Colosseum: A Multi-Agent LLM System for Autonomous Trading Strategy Discovery, Validation, and Deployment

Indie Hedge Fund

x.com/indiehedgefund

February 2026

Abstract

We present Colosseum, a production-grade multi-agent system that autonomously generates, critiques, validates, and deploys quantitative trading strategies using large language models (LLMs) as specialized reasoning agents. The system orchestrates nine distinct agent roles—including market regime detection, research analysis, strategy ideation, adversarial debate, backtesting, risk auditing, and production code generation—through a configurable pipeline that processes strategies across four distinct market domains: momentum/trend-following, mean reversion, factor rotation, and volatility mean reversion. Colosseum introduces several novel design contributions: (1) an adversarial debate mechanism where parallel bull and bear researcher agents present opposing cases to a facilitator agent, replacing traditional single-critic evaluation; (2) a template-first code generation approach where LLMs select and parameterize pre-tested strategy templates rather than generating executable code from scratch, achieving near-perfect execution reliability; (3) a multi-stage validation cascade combining walk-forward analysis with warmup-aware windowing, block bootstrap Monte Carlo significance testing, and independent risk auditing; and (4) a structured lessons database that enables regime-aware adaptive ideation by learning from past failures. We describe the system architecture, agent interaction patterns, prompt engineering methodology, and backtesting infrastructure in detail. We discuss design trade-offs encountered during development, including the tension between LLM creativity and execution reliability, cost management in multi-agent pipelines, and the challenges of preventing overfitting in autonomously generated strategies. The system has been deployed in production and operates continuously, generating and validating strategies every three hours.

Keywords: multi-agent systems, large language models, algorithmic trading, strategy generation, adversarial debate, backtesting, walk-forward validation

1 Introduction

The application of artificial intelligence to financial markets has a long history, from early expert systems and neural network-based prediction models to modern reinforcement learning agents (Bao et al., 2017; Fischer and Krauss, 2018). The recent emergence of large language models (LLMs) with strong reasoning, code generation, and structured output capabilities has opened new possibilities for automating not just individual predictions, but entire research workflows—from hypothesis generation through empirical validation to production deployment.

Traditional quantitative strategy development is a labor-intensive process. A human researcher identifies a market inefficiency hypothesis, designs trading rules to exploit it, implements backtesting code, validates results across multiple time periods, assesses robustness, and finally deploys the strategy with appropriate risk controls. Each step requires domain expertise and is prone to cognitive biases, particularly confirmation bias during strategy evaluation (Bailey et al., 2014a). The full cycle from idea to deployment typically requires days to weeks of focused effort.

We present **Colosseum**, a multi-agent system that automates this entire workflow using LLMs as specialized reasoning agents. The system operates as a continuous daemon, executing three-hour discovery cycles that generate strategy hypotheses, subject them to adversarial critique, validate them through rigorous backtesting, and produce deployment-ready trading scripts. The name evokes the adversarial arena in which strategy ideas must survive multiple rounds of critique and empirical validation before earning "production-ready" status.

1.1 Contributions

This paper makes the following contributions:

1. **System Architecture for Autonomous Strategy Discovery.** We describe a complete, production-deployed pipeline that orchestrates nine specialized LLM agents through an eight-stage strategy lifecycle. The architecture supports four distinct pipeline types with configurable thresholds, ticker universes, and validation criteria.
2. **Adversarial Debate Mechanism.** We introduce a three-agent debate system—bull researcher, bear researcher, and facilitator—that replaces traditional single-critic evaluation, producing more balanced strategy assessments.
3. **Template-First Code Generation.** We present an approach where LLMs select and parameterize pre-tested strategy templates rather than generating executable backtest code from scratch, achieving near-perfect execution reliability.
4. **Multi-Stage Validation Cascade.** We describe a five-gate validation system combining critic scoring, multi-period backtesting, walk-forward analysis with warmup-aware windowing, block bootstrap Monte Carlo significance testing, and independent risk auditing.
5. **Adaptive Learning from Failure.** We present a structured lessons database that captures failure patterns by edge family, market regime, and pipeline stage, enabling regime-aware adaptive ideation.

1.2 Paper Organization

Section 2 reviews related work. Section 3 presents the system architecture. Section 4 details agent design and interaction patterns. Section 5 describes backtesting and validation infrastructure. Section 6 walks through an illustrative pipeline cycle. Section 7 discusses design trade-offs and limitations. Section 8 outlines future work, and Section 9 concludes.

2 Related Work

2.1 Large Language Models in Finance

The application of LLMs to financial tasks has accelerated rapidly. **BloombergGPT** (Wu et al., 2023a) demonstrated that domain-specific pre-training on a 363-billion-token financial corpus improves performance on financial NLP tasks including sentiment analysis, named entity recognition, and question answering. **FinGPT** (Yang et al., 2023) took an open-source approach, providing a framework for fine-tuning LLMs on financial data with lightweight LoRA adaptation, reducing training costs from millions of dollars to under $300 per run. Lopez-Lira and Tang (2023) showed that GPT-4 can predict stock price movements from news headlines with significant accuracy, with forecasting ability emerging as a function of model scale.

More recently, **FinAgent** (Zhang et al., 2024) proposed the first multimodal foundation agent for financial trading that integrates numeric, textual, and visual market data with tool augmentation, achieving over 36% average improvement in profit across six financial datasets. **FinMem** (Yu et al., 2024a) introduced a memory-augmented LLM agent that converts raw financial inputs into hierarchical memories across different time horizons. **QuantAgent** (Wang et al., 2024a) focused specifically on using LLMs to generate and refine quantitative trading signals through an inner-loop/outer-loop architecture where a writer agent generates alpha factor scripts and a judge agent provides feedback. **Alpha-GPT** (Wang et al., 2023) explored human-AI interaction for alpha mining. A comprehensive survey by Ding et al. (2024) documents LLM trading agent architectures and reports annualized returns of 15–30% over strongest baselines across various agent designs.

Our work differs from these approaches in several ways. First, we orchestrate multiple specialized agents rather than using a single LLM for all tasks. Second, our system generates complete, executable trading strategies—not just signals or predictions—through a full lifecycle from ideation to deployment. Third, we emphasize rigorous empirical validation through multi-stage backtesting rather than relying on LLM-based evaluation alone.

2.2 Multi-Agent LLM Systems

The paradigm of multiple LLM agents collaborating on complex tasks has gained significant attention. **AutoGen** (Wu et al., 2023b) introduced a framework for building LLM applications via multi-agent conversation, supporting diverse interaction patterns including group chats and nested conversations. **MetaGPT** (Hong et al., 2023) demonstrated that encoding human-like Standard Operating Procedures (SOPs) into multi-agent frameworks reduces hallucination and improves coherence in software engineering tasks. Wang et al. (2024b) showed that a layered "Mixture-of-Agents" architecture exploiting the collaborativeness phenomenon of LLMs can achieve state-of-the-art performance on evaluation benchmarks. Dominguez et al. (2024) provide a comparative evaluation of AutoGen, CrewAI, and TaskWeaver on code generation tasks.

The concept of adversarial or debate-based evaluation in AI systems has theoretical roots in **AI Safety via Debate** (Irving et al., 2018), which proposed that two AI systems debating can help a human judge reach better decisions on questions that require superhuman knowledge. Du et al. (2024) demonstrated that multiple LLM instances proposing and debating responses over multiple rounds significantly enhances mathematical and strategic reasoning while reducing hallucinations. **ChatEval** (Chan et al., 2023) applied multi-agent debate to natural language

generation evaluation. Liang et al. (2023) identified the "Degeneration-of-Thought" problem in LLM self-reflection and showed that multi-agent debate with a judge can overcome it.

TradingAgents (Xiao et al., 2025) is the most directly related work, implementing a multi-agent LLM framework for stock trading inspired by trading firm structures, featuring specialized agents including bull/bear researchers and a risk management team. **FinCon** (Yu et al., 2024b) uses a manager-analyst communication hierarchy with verbal reinforcement for financial decision-making. **FinRobot** (Zhou et al., 2024) employs a multi-agent Chain-of-Thought system for equity research. However, these systems focus on real-time trading decisions or analysis rather than end-to-end strategy discovery and validation.

2.3 Automated Trading Strategy Generation

The automated generation of trading strategies has a substantial history predating LLMs. **Genetic Programming** (GP) has been extensively applied to evolve trading rules (Chen and Yeh, 2001; Koza, 1994), with mixed results regarding out-of-sample generalization. Brabazon and O'Neill (2006, 2008) extended these approaches with grammatical evolution and provided comprehensive coverage of natural computing in computational finance. Barbosa and Belo (2021) presented a GP framework for generating human-readable quantitative trading strategies, and Kampouridis et al. (2025) proposed strongly-typed GP combining sentiment and technical analysis.

More recent work has explored reinforcement learning for strategy discovery. **FinRL** (Liu et al., 2020) provided a framework for deep reinforcement learning in quantitative finance. Kou et al. (2024) proposed a three-stage LLM framework for automated alpha factor generation. Noguer i Alonso and Dupouy (2024) evaluated multiple LLMs specifically for generating Python trading strategy code, finding GPT-4-Turbo achieves the best results among tested models.

The key limitation of both GP and RL approaches is the difficulty of expressing complex, multi-condition trading logic and the tendency toward overfitting. Our template-based approach occupies a middle ground: the strategy structure is fixed by pre-tested templates, but the LLM selects which template to apply and how to parameterize it based on its understanding of the market hypothesis.

2.4 Backtesting Methodology and Overfitting

The risk of overfitting in backtested trading strategies is well-documented. Bailey et al. (2014a) introduced the concept of the "probability of backtest overfitting" and demonstrated that most backtested strategies are likely overfit when multiple strategy configurations are tested. Bailey and López de Prado (2014b) proposed the Deflated Sharpe Ratio to account for multiple hypothesis testing.

Walk-forward analysis (Pardo, 2008) addresses overfitting by splitting historical data into in-sample training and out-of-sample testing windows, advancing sequentially through time. Deep et al. (2025) recently proposed an interpretable hypothesis-driven walk-forward framework demonstrating realistic out-of-sample performance across 34 independent tests. **White's Reality Check** (White, 2000) and the **Superior Predictive Ability (SPA) test** (Hansen, 2005) provide formal statistical frameworks for evaluating whether a strategy's performance is genuine or an artifact of data mining. Sullivan et al. (1999) applied White's bootstrap methodology

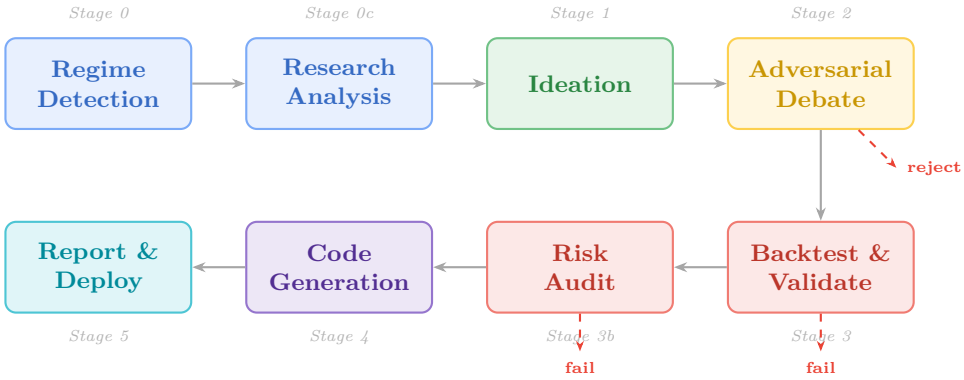

Figure 1: Pipeline architecture showing the eight-stage strategy lifecycle. Each stage acts as an independent gate; strategies may be rejected at any point (dashed red arrows).

to 100 years of technical trading rules, finding that performance degraded significantly out of sample.

Monte Carlo methods for strategy validation include permutation tests (Aronson, 2007; Masters, 2020) and bootstrap methods (Efron and Tibshirani, 1993). Politis and Romano (1994) introduced the stationary bootstrap for time series, which preserves temporal dependence—a critical requirement when testing momentum-based strategies.

2.5 Market Regime Detection

Regime-switching models have a long history in financial econometrics. Hamilton (1989) introduced the Markov-switching model, and subsequent work has applied hidden Markov models (Ang and Bekaert, 2002), change-point detection (Bai and Perron, 2003), and machine learning classifiers to identify market regimes. More recent work has applied hidden Markov models to factor investing (Nystrup et al., 2020) and examined regime changes across multiple asset classes (Ang and Timmermann, 2012; Guidolin and Timmermann, 2007). Kritzman et al. (2012) demonstrated that dynamic regime-aware asset allocation outperforms static allocation, especially for investors seeking to avoid large losses.

Our regime detector takes a deliberately simple, purely quantitative approach using six observable market factors. This deterministic classifier avoids LLM costs and ensures reproducibility, while providing sufficient regime granularity (five states) to inform strategy selection.

3 System Architecture

3.1 Overview

Colosseum is organized as a multi-stage pipeline that processes trading strategy hypotheses through successive gates of increasing rigor. The system operates as a continuous daemon, executing discovery cycles at configurable intervals (default: three hours). Figure 1 illustrates the eight-stage pipeline architecture.

Table 1: Summary of the four pipeline types with their distinguishing characteristics.

Pipeline	**Target Assets**	**Templates**	**Validation Focus**
Momentum/Trend	SPY, QQQ, TQQQ, sectors	SMA, dual momentum, breakout	Trend persistence, low turnover
Mean Reversion	SPY, QQQ, leveraged ETFs	RSI, Bollinger, consecutive down	Short holding, high win rate
Factor Rotation	Factor ETFs, sectors	Cross-sectional, calendar, spread	Multi-position, low correlation
Volatility	SVXY, UVXY, VXX, SPY	VIX regime, term structure	Tail risk, regime sensitivity

3.2 Four Pipeline Types

The system supports four distinct strategy families, each with its own ticker universe, template set, validation thresholds, and prompt variants (Table 1).

Each pipeline type defines its own ticker universe, template registry, preferred market regimes, validation thresholds (minimum Sharpe ratio, maximum drawdown, minimum trade count, walk-forward degradation limits, Monte Carlo significance levels), and prompt variants for each agent role.

This parameterization allows the system to apply appropriate rigor to each strategy family. For example, volatility strategies operate with relaxed walk-forward thresholds (95% maximum Sharpe degradation vs. 70% for momentum) because volatility clustering causes legitimate strategies to have highly variable out-of-sample performance.

3.3 Orchestration

The orchestrator (`Pipeline` class) manages the execution of pipeline cycles. In continuous mode, the system round-robins through enabled pipeline types, with an optional portfolio-aware selection mechanism that favors underrepresented strategy families.

Within each cycle, the orchestrator: (1) initializes market context by running the regime detector and research analyst; (2) generates up to 20 strategy hypotheses via ideation; (3) quick-filters to the top N ideas when more than N are generated; (4) evaluates hypotheses through adversarial debate with parallel execution across strategies; (5) backtests approved strategies using the template-based engine; (6) conducts independent risk audits; (7) generates production code and checks for duplicates; and (8) updates portfolio allocations and optionally breeds new strategies.

Parallelism is managed through a `ThreadPoolExecutor` with a configurable worker count (default: 4). The orchestrator provides budget-aware cancellation—if the cumulative LLM cost within a cycle exceeds the configured limit (default: $10 USD), pending agent calls are cancelled.

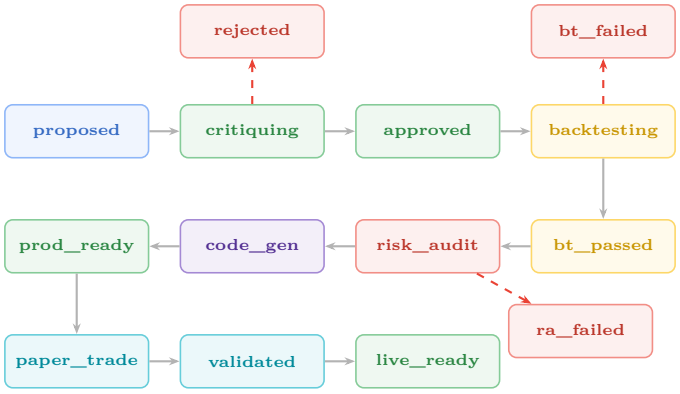

Figure 2: Strategy lifecycle state machine. Solid arrows represent normal progression; dashed red arrows represent rejection paths.

3.4 State Management

Strategy state is maintained through a dual-storage system: an **append-only JSONL log** serving as an immutable audit trail, and a **SQLite database** as the primary query interface with auto-migrating schema.

Each strategy progresses through a well-defined lifecycle, illustrated in Figure 2.

3.5 Cost Management

Operating a multi-agent LLM pipeline incurs non-trivial API costs. Colosseum implements cost management at multiple levels:

- **Per-call tracking:** Every LLM call records token counts mapped to provider-specific pricing tables using `Decimal` arithmetic to prevent floating-point drift.
- **Per-cycle budget cap:** A configurable maximum cost per cycle (default: $10 USD). Exceeding this raises a `BudgetExceededError` that cancels pending parallel tasks.
- **Temperature control:** Temperatures tuned per agent role—high (1.2) for creative ideation, low (0.1) for deterministic code generation—affecting output diversity and token consumption.
- **Zero-cost components:** The regime detector is purely quantitative (no LLM calls), and the template system eliminates LLM-generated backtest code.

4 Agent Design

4.1 Base Agent Framework

All agents inherit from a `BaseAgent` class that provides unified infrastructure:

LLM Interface. Three abstraction layers support different output requirements: `call_llm()` for raw text response with token usage tracking; `call_llm_json()` for automatic JSON parsing with markdown fence stripping and retry on parse failure (up to 2 attempts); and `call_llm_structured()` for Pydantic-validated typed output with automatic fallback to JSON parsing.

Table 2: Agent taxonomy with temperature settings and key roles.

Agent	Temp.	LLM	Role
Regime Detector	N/A	No	6-factor quantitative market regime classifier
Research Analyst	0.3	Yes	Synthesizes enriched market data into structured reports
Screener	0.3	Yes	Quantitative pre-filter on stock universe
Ideator	1.2	Yes	Generates 1–3 novel strategy hypotheses per cycle
Bull Researcher	0.9	Yes	Builds strongest case *for* the strategy
Bear Researcher	0.3	Yes	Stress-tests and finds weaknesses
Facilitator	0.3	Yes	Synthesizes debate; renders verdict
Backtester	0.2	Yes	Selects template, parameterizes, executes in sandbox
Risk Auditor	0.3	Yes	Independent post-backtest risk assessment
Code Generator	0.1	Yes	Produces standalone trading scripts

Retry Logic. All LLM calls use exponential backoff (3 attempts, 2-second base delay, 60-second maximum) via the `tenacity` library. JSON parse failures trigger an additional retry with a corrective prompt including the parse error.

Provider Abstraction. The base agent supports three LLM providers—Anthropic (Claude), Google (Gemini), and OpenRouter (multi-model gateway)—through a unified interface.

4.2 Agent Taxonomy

The system employs nine distinct agent roles, summarized in Table 2.

4.2.1 Regime Detector

The regime detector is a purely quantitative classifier that evaluates six market factors: (1) SPY 50-day SMA slope relative to 200-day SMA; (2) 20-day annualized realized volatility; (3) VIX level relative to historical percentiles; (4) VIX term structure (VIX3M/VIX ratio); (5) ADX trend strength; and (6) market breadth (percentage of sector ETFs above 200-day SMA). These factors map to five regimes: `BULL_LOW_VOL`, `BULL_HIGH_VOL`, `BEAR_HIGH_VOL`, `CHOP`, and `TRANSITION`.

4.2.2 Ideator

The ideation agent operates at the highest temperature (1.2), reflecting its creative role. It receives current regime and research context, a summary of existing strategies, recent rejection reasons and backtest failures, structured lessons learned (edge family success rates, regime-specific patterns), and a pipeline-specific prompt variant. The agent produces 1–3 strategy hypotheses as structured JSON objects.

4.2.3 Adversarial Debate System

The debate system is one of the system's primary contributions. Figure 3 illustrates its architecture.

Bull Researcher ($T = 0.9$): Explicitly tasked with building the strongest possible case *for* the strategy, identifying supporting historical evidence and favorable market conditions.

Bear Researcher ($T = 0.3$): Explicitly tasked with stress-testing the strategy, identifying failure modes, adverse conditions, implementation risks, and overfitting potential. The lower temperature encourages methodical, analytically rigorous critique.

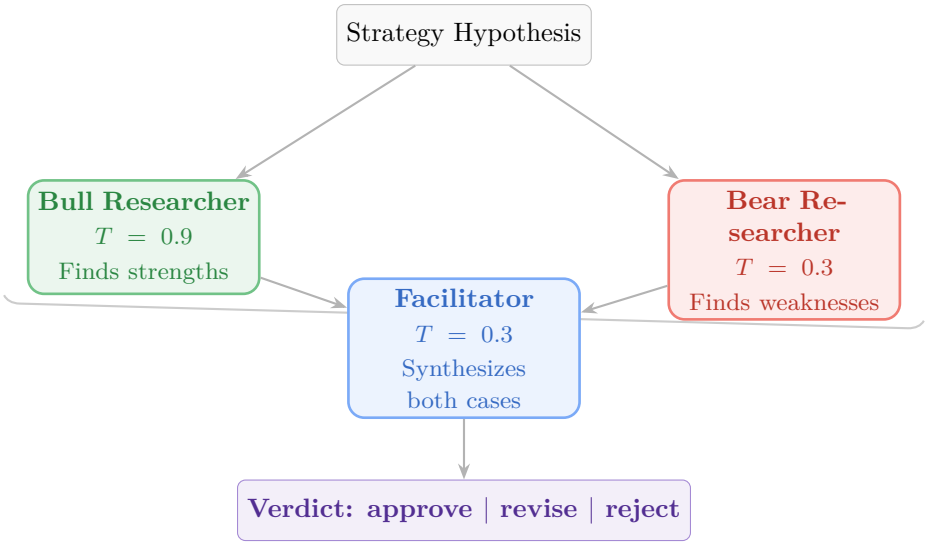

Figure 3: Adversarial debate architecture. Bull and bear researchers execute in parallel with asymmetric temperatures; the facilitator synthesizes their arguments into a single verdict.

Facilitator ($T = 0.3$): Receives both cases and synthesizes them into a balanced assessment, producing a final score (1–10) and a verdict: *approve*, *revise*, or *reject*.

A key design decision is the **revise threshold**: strategies receiving a facilitator score of ≥ 4.0 are approved for backtesting, even with a "revise" verdict. This reflects the principle that empirical validation should take precedence over pre-backtest theoretical assessment—a strategy with interesting characteristics but theoretical concerns should be tested rather than rejected a priori.

The asymmetric temperature assignment is deliberate. The bull researcher operates at higher temperature (0.9) to encourage creative identification of non-obvious strengths, while the bear researcher and facilitator operate at low temperature (0.3) for analytical rigor.

4.2.4 Backtester

Rather than generating backtest code directly (which exhibited ∼50% failure rate in early iterations), the backtester follows a **template-first** approach: (1) the LLM selects the most appropriate template from the pipeline's registry; (2) proposes parameter values within allowed ranges; (3) the template generates a complete strategy function; and (4) execution occurs in a sandboxed environment. This is detailed in Section 5.

4.2.5 Risk Auditor

The risk auditor serves as an independent post-backtest gate, evaluating benchmark correlation (high R^2 to SPY suggests no alpha), tail risk (CVaR at 95th and 99th percentiles), regime robustness, and return distribution characteristics. It produces a verdict of *pass*, *caution*, or *fail*. This stage is the pipeline's primary bottleneck, frequently rejecting strategies that passed backtesting on metrics alone.

Table 3: Template registry organized by strategy family.

Family	Count	Representative Templates
Momentum/Trend	7	Trend-following SMA, dual momentum, breakout volume, overnight premium
Mean Reversion	7	RSI reversion, Bollinger reversion, consecutive down days, pairs trading
Factor/Multi-Asset	6	Factor rotation, calendar effects, sector pair rotation
Structural	3	VIX regime rotation, volatility regime, cross-asset divergence

4.3 Prompt Engineering Architecture

The prompt system uses a **variant architecture** where each agent has a base prompt and optional pipeline-specific variants stored as Markdown files, enabling modification without code changes. The loading mechanism prioritizes pipeline-specific variants (e.g., `ideator_momentum_trend.md`) over base prompts (e.g., `ideator.md`), supporting 40+ prompt files across agent roles and pipeline types.

5 Backtesting and Validation Infrastructure

5.1 Template-Based Code Generation

The template system is a central architectural decision motivated by a practical observation: LLM-generated backtest code fails to execute approximately 50% of the time due to syntax errors, logic errors, undefined variables, infinite loops, or excessive memory consumption.

The template system separates the *what* (strategy logic) from the *how* (implementation). Twenty-three pre-built strategy templates are organized into four families (Table 3).

Each template is a Python function that generates a complete `run_strategy(data, initial_capital)` function given a parameter dictionary. Templates include indicator computation with configurable lookback periods, signal generation based on indicator values, position sizing with volatility scaling, risk management (stop losses, maximum holding periods, cooldown periods), and warmup handling (60–250 day lead-in).

The template approach preserves LLM creativity at the *strategic level* (which template and parameters best express this hypothesis?) while ensuring *implementation reliability*.

5.2 Sandbox Execution Environment

Strategy code executes in a sandboxed `exec()` environment with import stripping, whitelisted builtins only (safe Python builtins such as `range`, `len`, `zip`, `map`, `filter`, `all`, `any`, etc.), injected dependencies (`pandas`, `numpy`, commission/slippage parameters), and a 120-second `SIGALRM` timeout.

5.3 Multi-Period Backtesting

Each strategy is tested across multiple historical time periods capturing different market conditions (Table 4). A strategy must pass minimum thresholds in at least 2 of 5 periods (configurable per pipeline), reducing the probability that it is overfit to a single market regime.

Table 4: Backtest test periods and their market characteristics.

Period	Date Range	Market Character
Bull 2013–2015	Jan 2013 – May 2015	Low-volatility bull market
Bull 2017–2018	Jan 2017 – Dec 2018	Strong trend with late-cycle volatility
COVID	Jan 2019 – Dec 2020	Regime change: bull → crash → recovery
Post-COVID	Jan 2021 – Dec 2022	Inflation, rate hikes, growth-to-value rotation
Recent	Jan 2023 – present	AI-driven rally, narrow breadth

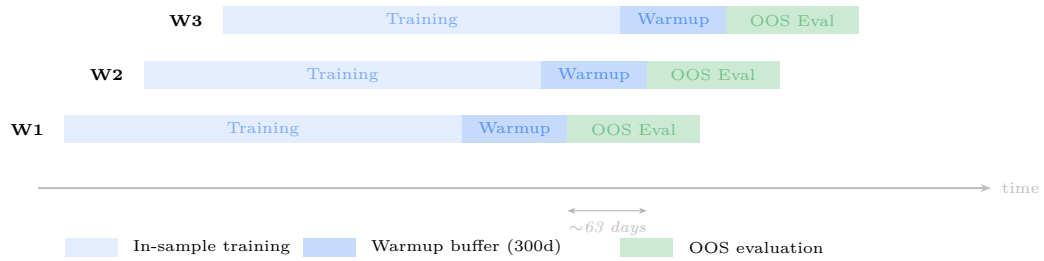

Figure 4: Walk-forward validation with warmup-aware windowing. Each window includes a warmup buffer (300 days) prepended to the out-of-sample period to ensure indicators are fully computed before performance evaluation begins.

5.4 Walk-Forward Validation

Walk-forward analysis provides the primary defense against overfitting by measuring the degradation between in-sample and out-of-sample performance.

The system uses a rolling walk-forward with a fixed 2-year training window advanced by approximately one quarter (63 trading days) per step. A critical innovation is **warmup-aware windowing**: each window begins `warmup_buffer_days` (default: 300) before the out-of-sample start, ensuring all indicators are fully computed when the evaluation period begins. This addresses a subtle but important issue: if a strategy uses a 200-day moving average and the out-of-sample window begins without a warmup period, the equity curve during the indicator computation phase is flat, potentially inflating or masking true performance.

Figure 4 illustrates the warmup-aware walk-forward window construction.

A strategy passes walk-forward validation if: (1) Sharpe ratio degradation from in-sample to out-of-sample does not exceed the pipeline-specific threshold (70–95%); (2) out-of-sample Sharpe exceeds the minimum threshold; and (3) the strategy is profitable in at least 50% of out-of-sample windows.

5.5 Monte Carlo Significance Testing

The Monte Carlo robustness test addresses the question: *Could this strategy's performance have arisen by chance?*

We use a **block bootstrap** rather than i.i.d. permutation because trading strategies—particularly momentum strategies—exploit serial correlation in returns. An i.i.d. shuffle destroys this temporal structure, creating an inappropriately easy null hypothesis.

The procedure: (1) extract daily returns from the strategy's equity curve; (2) for each of 500 simulations, randomly sample contiguous 20-day blocks (with replacement) to construct a synthetic return series of the same length; (3) compute the Sharpe ratio of each synthetic series;

Table 5: Monte Carlo significance thresholds by pipeline type.

Pipeline	Threshold	Rationale
Momentum/Trend	0.20	Momentum exploits serial correlation; moderate threshold
Mean Reversion	0.20	Short-term reversion well-documented; moderate threshold
Factor Rotation	0.25	Cross-sectional effects noisier; relaxed threshold
Volatility	0.30	Vol clustering creates naturally high variance; relaxed

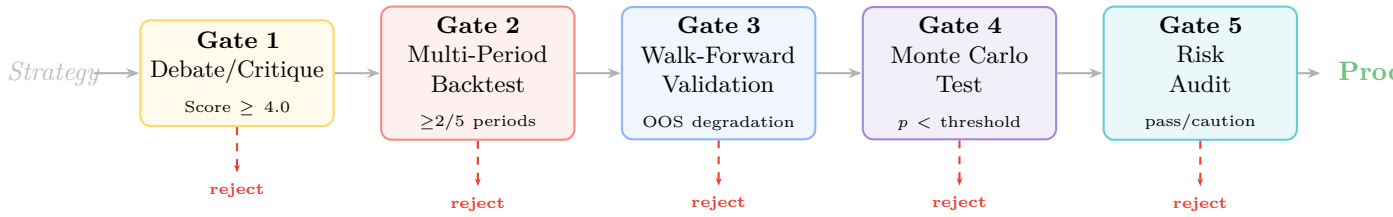

Figure 5: Five-gate validation cascade. Each gate is an independent rejection point with pipeline-specific thresholds. Only strategies surviving all five gates achieve "production-ready" status.

(4) calculate the p-value as the fraction of synthetic Sharpe ratios $\geq$ the observed Sharpe.

The 20-day block size preserves approximately one month of serial correlation structure. Pipeline-specific significance thresholds (Table 5) reflect different statistical properties of each strategy family.

5.6 Validation Cascade Summary

The complete validation cascade is illustrated in Figure 5. Strategies that survive all five gates advance to code generation and are granted "production-ready" status.

5.7 Performance Metrics

The system computes comprehensive metrics for each backtest: standard metrics (total return, CAGR, Sharpe ratio, Sortino ratio, maximum drawdown, Calmar ratio, win rate, profit factor, trade count); advanced risk metrics (CVaR at 95th and 99th percentiles, Ulcer index, return skewness and kurtosis); and statistical metrics (Bayesian Sharpe ratio with skeptical prior and 95% credible interval lower bound). All metrics are computed after stripping the warmup period to prevent dilution by zero-return days.

6 Illustrative Pipeline Cycle

To make the system's operation concrete, we trace a single cycle of the momentum/trend pipeline.

Stage 0: Regime Detection. The quantitative classifier evaluates current conditions: SPY above its 200-day SMA with positive slope, realized volatility below the 50th percentile, VIX at 16.5 in contango, ADX at 28, and 9/11 sector ETFs above their 200-day SMAs. Classification: `BULL_LOW_VOL`.

Stage 0c: Research Analysis. The research analyst identifies technology sector leadership (XLK +3.2% relative to SPY over 20 days), compressed credit spreads, rising leveraged ETF fund flows, and VIX term structure contango. Report highlights favorability for trend-following strategies.

Stage 1: Ideation. Informed by the regime, research report, and lessons database (showing trend-following SMA has 23% historical success rate in `BULL_LOW_VOL` vs. 8% for mean reversion), the ideator generates three hypotheses. One proposes a dual momentum strategy on QQQ with 20-day and 60-day momentum and a VIX contango filter.

Stage 2: Adversarial Debate. For the dual momentum strategy: the *bull researcher* highlights strong theoretical backing for momentum in low-vol regimes and regime-aware VIX filtering; the *bear researcher* notes momentum crowding risk in narrow-breadth markets and overly permissive VIX filter (contango is the base state ∼75% of the time); the *facilitator* synthesizes: score 6.5, verdict *revise*. The strategy advances.

Stage 3: Backtesting. The backtester selects `DUAL_MOMENTUM` with parameters: fast_period=20, slow_period=60, entry_threshold=0.02, stop_loss_pct=0.05. Execution across five test periods yields passing metrics in 3/5 periods. Walk-forward shows 45% Sharpe degradation (below the 70% threshold: pass). Monte Carlo yields $p = 0.12$ (below 0.20: pass).

Stage 3b: Risk Audit. R^2 to SPY is 0.35 (not a benchmark clone), CVaR-99 is −4.2% (acceptable), maximum consecutive losing days is 7. Verdict: *pass*.

Stage 4: Code Generation. A standalone Python script is produced with backtest, paper, and live trading modes. Deduplication confirms uniqueness. Status: `production_ready`.

7 Discussion

7.1 Design Trade-offs

LLM Creativity vs. Execution Reliability. The most significant design tension is between allowing creative freedom in strategy design and ensuring reliable execution. Early iterations with fully LLM-generated backtest code exhibited ∼50% failure rates. The template-first approach resolves this by constraining the strategy space to 23 pre-defined templates—an acceptable trade-off, as the templates cover the major quantitative strategy families and the LLM's creativity is redirected toward template selection, parameter optimization, and combination of conditions.

Single Critic vs. Adversarial Debate. The debate mechanism incurs ∼3× the LLM cost of a single critic but produces qualitatively different assessments. The single critic tended toward either excessive optimism or pessimism depending on the prompt, while the debate forces explicit consideration of both sides, producing more calibrated scores.

Validation Strictness vs. Discovery Rate. The five-gate cascade is deliberately strict, rejecting the majority of generated strategies. In quantitative finance, false positives (deploying an overfit strategy) are far more costly than false negatives (rejecting a genuinely profitable strategy that could be rediscovered later).

Deterministic vs. LLM Regime Detection. We chose a purely quantitative detector for reproducibility (same data → same classification), zero cost, and sub-second execution. An LLM-based detector could incorporate qualitative information but would introduce non-reproducibility and cost.

7.2 Limitations

Strategy Space Constraints. The template system limits the strategy space to pre-defined patterns. Truly novel strategies that don't fit existing templates cannot be discovered. Expanding the registry requires manual development and testing.

Backtest Realism. Despite realistic transaction costs, the engine does not model market impact, partial fills, or liquidity constraints. For liquid ETFs at $100K capital, these effects are likely small but would become significant at larger scales.

Single-LLM Dependency. All agents currently use the same model. Different agents might benefit from different models—more capable models for ideation and facilitation, faster models for template selection and code generation.

Survivorship Bias. The fixed ticker universe contains only currently listed securities. Historical backtests on these securities benefit from the implicit knowledge that they survived to the present.

No Transaction-Level Simulation. The engine operates at daily OHLCV resolution and cannot model intraday execution dynamics or order book effects.

7.3 Lessons Learned

Prompt Engineering is Iterative. The 40+ prompt variants represent months of iteration. Small changes in wording can dramatically affect agent behavior—particularly the ideator, where a single instruction can shift the distribution of generated strategies.

Error Messages are Gold. The backtester's error-aware retry loop, which feeds actual execution errors back to the LLM, was one of the most impactful improvements.

The Risk Auditor is the Bottleneck. Despite being conceptually simple, the risk auditor rejects the largest fraction of strategies that pass backtesting, suggesting it captures qualitative risk factors that quantitative metrics miss.

Structured Failure Analysis Improves Generation. The structured lessons database significantly improved ideation effectiveness. Early iterations produced many strategies in exhausted edge families; feeding success rates back to the ideator redirected generation toward productive areas.

8 Future Work

Several directions for future development are apparent:

Ablation Studies. Systematic comparison of pipeline variants: adversarial debate vs. single critic, template-first vs. LLM-generated code, walk-forward vs. simple train/test split. These ablations would quantify the contribution of each design decision.

Strategy Combination. More sophisticated ensemble methods—stacking strategy signals, Bayesian model combination, or metalearning to select strategies based on market conditions—could improve portfolio-level performance.

Multi-Model Agent Assignment. Assigning reasoning-heavy agents to more capable models while using efficient models for template selection and code generation could improve both quality and cost efficiency.

Formal Multiple Testing Correction. While the Monte Carlo test controls for individual strategy significance, a formal correction for the system-level multiple testing burden (e.g., Bailey and López de Prado, 2014b) would provide more rigorous assessment of the overall alpha discovery rate.

Reinforcement Learning from Pipeline Outcomes. Strategy outcomes could serve as reward signals to fine-tune agents, creating a closed-loop learning system.

9 Conclusion

We have presented Colosseum, a multi-agent LLM system that automates the full lifecycle of quantitative trading strategy discovery—from market analysis and hypothesis generation through adversarial critique, empirical validation, and production code generation. The system introduces several design contributions: an adversarial debate mechanism for strategy evaluation, a template-first approach achieving near-perfect execution reliability, a multi-stage validation cascade combining walk-forward analysis and Monte Carlo significance testing, and a structured lessons database for adaptive ideation.

The system has been deployed in production and operates continuously. Its architecture demonstrates that LLMs can serve effectively as specialized agents in complex, multi-stage reasoning pipelines—not merely as prediction engines, but as components of autonomous research workflows that generate, evaluate, and validate hypotheses with minimal human intervention.

We believe the design patterns presented here—particularly the adversarial debate mechanism, template-first code generation, and structured failure learning—are applicable beyond algorithmic trading to any domain where hypotheses must be generated, empirically validated, and deployed with high reliability.

References

Ang, A. and Bekaert, G. (2002). International asset allocation with regime shifts. *Review of Financial Studies*, 15(4):1137–1187.

Ang, A. and Timmermann, A. (2012). Regime changes and financial markets. *Annual Review of Financial Economics*, 4(1):313–337.

Aronson, D. R. (2007). *Evidence-Based Technical Analysis: Applying the Scientific Method and Statistical Inference to Trading Signals*. John Wiley & Sons.

Bai, J. and Perron, P. (2003). Computation and analysis of multiple structural change models. *Journal of Applied Econometrics*, 18(1):1–22.

Bailey, D. H., Borwein, J. M., López de Prado, M., and Zhu, Q. J. (2014a). Pseudomathematics and financial charlatanism: The effects of backtest overfitting on out-of-sample performance. *Notices of the American Mathematical Society*, 61(5):458–471.

Bailey, D. H. and López de Prado, M. (2014b). The deflated Sharpe ratio: correcting for selection bias, backtest overfitting, and non-normality. *Journal of Portfolio Management*, 40(5):94–107.

Bao, W., Yue, J., and Rao, Y. (2017). A deep learning framework for financial time series using stacked autoencoders and long-short term memory. *PLoS ONE*, 12(7):e0180944.

Barbosa, R. P. and Belo, O. (2021). Generating and optimizing human-readable quantitative program trading strategies through a genetic programming framework. *Procedia Computer Science*, 192:1026–1035.

Brabazon, A. and O'Neill, M. (2006). *Biologically Inspired Algorithms for Financial Modelling*. Springer.

Brabazon, A. and O'Neill, M., editors (2008). *Natural Computing in Computational Finance*, volume 100 of *Studies in Computational Intelligence*. Springer.

Chan, C.-M., Chen, W., Su, Y., Yu, J., Xue, W., Zhang, S., Fu, J., and Liu, Z. (2023). ChatEval: Towards better LLM-based evaluators through multi-agent debate. *arXiv preprint arXiv:2308.07201*.

Chen, S.-H. and Yeh, C.-H. (2001). Evolving traders and the business school with genetic programming. *Journal of Economic Dynamics and Control*, 25(3–4):363–393.

Deep, G., Deep, A., and Lamptey, W. (2025). Interpretable hypothesis-driven trading: A rigorous walk-forward validation framework. *arXiv preprint arXiv:2512.12924*.

Ding, H., Li, Y., Wang, J., and Chen, H. (2024). Large language model agent in financial trading: A survey. *arXiv preprint arXiv:2408.06361*.

Dominguez, V., O'Neill, J., and Zhan, H. (2024). Benchmarking large language models for multi-agent systems. In *Advances in Practical Applications of Agents, Multi-Agent Systems, and Digital Twins: The PAAMS Collection*, pages 39–51. Springer.

Du, Y., Li, S., Torralba, A., Tenenbaum, J. B., and Mordatch, I. (2024). Improving factuality and reasoning in language models through multiagent debate. In *Proceedings of ICML 2024*.

Efron, B. and Tibshirani, R. J. (1993). *An Introduction to the Bootstrap*. Chapman and Hall/CRC.

Fischer, T. and Krauss, C. (2018). Deep learning with long short-term memory networks for financial market predictions. *European Journal of Operational Research*, 270(2):654–669.

Guidolin, M. and Timmermann, A. (2007). Asset allocation under multivariate regime switching. *Journal of Economic Dynamics and Control*, 31(11):3503–3544.

Hamilton, J. D. (1989). A new approach to the economic analysis of nonstationary time series and the business cycle. *Econometrica*, 57(2):357–384.

Hansen, P. R. (2005). A test for superior predictive ability. *Journal of Business & Economic Statistics*, 23(4):365–380.

Hong, S., Zhuge, M., Chen, J., Zheng, X., Cheng, Y., Zhang, C., et al. (2023). MetaGPT: Meta programming for a multi-agent collaborative framework. *arXiv preprint arXiv:2308.00352*.

Irving, G., Christiano, P., and Amodei, D. (2018). AI safety via debate. *arXiv preprint arXiv:1805.00899*.

Kampouridis, M., Giagkiozis, I., and Sherkat, N. (2025). A novel strongly-typed genetic programming algorithm for combining sentiment and technical analysis for algorithmic trading. *Knowledge-Based Systems*, 309:112863.

Kou, Z., Yu, H., Luo, J., Peng, J., Li, X., Liu, C., Dai, J., Chen, L., Han, S., and Guo, Y. (2024). Automate strategy finding with LLM in quant investment. In *Findings of EMNLP 2024*.

Koza, J. R. (1994). Genetic programming as a means for programming computers by natural selection. *Statistics and Computing*, 4(2):87–112.

Kritzman, M., Page, S., and Turkington, D. (2012). Regime shifts: Implications for dynamic strategies. *Financial Analysts Journal*, 68(3):22–39.

Liang, T., He, Z., Jiao, W., Wang, X., Wang, Y., Wang, R., Yang, Y., Tu, Z., and Shi, S. (2023). Encouraging divergent thinking in large language models through multi-agent debate. In *Proceedings of EMNLP 2024*.

Liu, X.-Y., Yang, H., Chen, Q., Zhang, R., Yang, L., Xiao, B., and Wang, C. D. (2020). FinRL: A deep reinforcement learning library for automated stock trading in quantitative finance. *arXiv preprint arXiv:2011.09607*.

Lopez-Lira, A. and Tang, Y. (2023). Can ChatGPT forecast stock price movements? Return predictability and large language models. *arXiv preprint arXiv:2304.07619*.

Masters, T. (2020). *Permutation and Randomization Tests for Trading System Development*. Self-published.

Noguer i Alonso, M. and Dupouy, H. (2024). Evaluating LLMs in financial tasks: Code generation in trading strategies. *SSRN Working Paper No. 4752797*.

Nystrup, P., Madsen, H., and Lindström, E. (2020). Regime-switching factor investing with hidden Markov models. *Journal of Risk and Financial Management*, 13(12):311.

Pardo, R. (2008). *The Evaluation and Optimization of Trading Strategies*. John Wiley & Sons.

Politis, D. N. and Romano, J. P. (1994). The stationary bootstrap. *Journal of the American Statistical Association*, 89(428):1303–1313.

Sullivan, R., Timmermann, A., and White, H. (1999). Data-snooping, technical trading rule performance, and the bootstrap. *The Journal of Finance*, 54(5):1647–1691.

Wang, G., Yang, G., Du, Z., Fan, L., and Li, X. (2023). Alpha-GPT: Human-AI interactive alpha mining for quantitative investment. *arXiv preprint arXiv:2308.00016*.

Wang, G., Zhao, S., Yang, G., Du, Z., Fan, L., and Li, X. (2024a). QuantAgent: Seeking Holy Grail in trading by self-improving large language model. *arXiv preprint arXiv:2402.03755*.

Wang, J., Wang, J., Athiwaratkun, B., Zhang, C., and Zou, J. (2024b). Mixture-of-Agents enhances large language model capabilities. In *Proceedings of ICLR 2025*.

White, H. (2000). A reality check for data snooping. *Econometrica*, 68(5):1097–1126.

Wu, S., Irsoy, O., Lu, S., Dabravolski, V., Dredze, M., Gehrmann, S., Kambadur, P., Rosenberg, D., and Mann, G. (2023a). BloombergGPT: A large language model for finance. *arXiv preprint arXiv:2303.17564*.

Wu, Q., Bansal, G., Zhang, J., Wu, Y., Li, B., Zhu, E., et al. (2023b). AutoGen: Enabling next-gen LLM applications via multi-agent conversation. In *Proceedings of COLM 2024*.

Xiao, Y., Sun, E., Luo, D., and Wang, W. (2025). TradingAgents: Multi-agents LLM financial trading framework. *arXiv preprint arXiv:2412.20138*.

Yang, H., Liu, X.-Y., and Wang, C. D. (2023). FinGPT: Open-source financial large language models. In *FinLLM Symposium at IJCAI 2023*.

Yu, D., Li, K., Wang, Y., Lukasiewicz, T., and Shi, Z. (2024a). FinMem: A performance-enhanced LLM trading agent with layered memory and character design. In *ICLR Workshop on LLM Agents, 2024*.

Yu, Y., Yao, Z., Li, H., Deng, Z., Jiang, Y., Cao, Y., et al. (2024b). FinCon: A synthesized LLM multi-agent system with conceptual verbal reinforcement for enhanced financial decision making. In *Advances in Neural Information Processing Systems (NeurIPS 2024)*.

Zhang, Z., Zheng, H., Zhao, Y., Zhang, R., and Wu, D. J. (2024). FinAgent: A multimodal foundation agent for financial trading using tool-augmented reasoning. In *Proceedings of KDD 2024*.

Zhou, T., Wang, P., Wu, Y., and Yang, H. (2024). FinRobot: AI agent for equity research and valuation with large language models. In *1st Workshop on LLMs and Generative AI for Finance at ICAIF 2024*.