ISTANBUL TECHNICAL UNIVERSITY ★ GRADUATE SCHOOL

AN AGENT-BASED FRAMEWORK FOR SYSTEMATIC GENERATIVE FEATURE ENGINEERING IN FINANCIAL MARKETS

M.Sc. THESIS

Mustafa Enes KARAMAN

Department of Data Engineering and Business Analytics

Big Data and Business Analytics Programme

JUNE 2026

ISTANBUL TECHNICAL UNIVERSITY ★ GRADUATE SCHOOL

AN AGENT-BASED FRAMEWORK FOR SYSTEMATIC GENERATIVE FEATURE ENGINEERING IN FINANCIAL MARKETS

M.Sc. THESIS

Mustafa Enes KARAMAN
(528231076)

Department of Data Engineering and Business Analytics

Big Data and Business Analytics Programme

Thesis Advisor: Prof. Dr. Alp ÜSTÜNDAĞ

JUNE 2026

ISTANBUL TEKNİK ÜNİVERSİTESİ ★ LİSANSÜSTÜ EĞİTİM ENSTİTÜSÜ

FİNANSAL PİYASALARDA SİSTEMATİK ÜRETİCİ DEĞİŞKEN MÜHENDİSLİĞİ İÇİN YAPAY ZEKÂ AJANLARI TEMELLİ BİR YAKLAŞIM

YÜKSEK LİSANS TEZİ

Mustafa Enes KARAMAN
(528231076)

Veri Mühendisliği ve İş Analitiği Anabilim Dalı

Büyük Veri ve İş Analitiği Programı

Tez Danışmanı: Prof. Dr. Alp ÜSTÜNDAĞ

HAZİRAN 2026

Mustafa Enes KARAMAN, a M.Sc. student of İTU Graduate School student ID 528231076, successfully defended the thesis/dissertation entitled "AN AGENT-BASED FRAMEWORK FOR SYSTEMATIC GENERATIVE FEATURE ENGINEERING IN FINANCIAL MARKETS", which he prepared after fulfilling the requirements specified in the associated legislations, before the jury whose signatures are below.

Thesis Advisor :	**Prof. Dr. Alp ÜSTÜNDAĞ** İstanbul Technical University	.....
Jury Members :	**Prof. Dr. Emre ÇEVİKCAN** İstanbul Technical University	.....
	Prof. Dr. Selim ZAİM Ibn Haldun University	.....

Date of Submission : 04 June 2026
Date of Defense : 17 June 2026

To my parents, for their unwavering support,

FOREWORD

As I conclude this academic journey, I wish to express my sincere gratitude to all who have supported me throughout my education and thesis process. Their trust, encouragement, and understanding during challenging periods have been instrumental in my academic development and personal growth. Knowing that I was supported at every stage was vital in sustaining my motivation and determination.

I extend my heartfelt appreciation to my family, particularly my mother and father, whose unwavering patience, belief in me, and support have been a constant source of strength. Without their encouragement and sacrifices, the completion of this academic process and my current achievements would not have been possible. Their dedication and perseverance have been fundamental in shaping my character.

Finally, I sincerely thank my thesis advisor, Prof. Dr. Alp ÜSTÜNDAĞ, for his invaluable guidance throughout my thesis. His academic insight, experience, and consistent support allowed me to progress confidently in this new academic field. Beyond the thesis process, his constructive feedback, encouragement, and mentorship were essential to the successful completion of this study and to my overall academic growth.

June 2026 Mustafa Enes KARAMAN

TABLE OF CONTENTS

Page

FOREWORD ..... **ix**
TABLE OF CONTENTS ..... **xi**
ABBREVIATIONS ..... **xvii**
LIST OF TABLES ..... **xix**
LIST OF FIGURES ..... **xxi**
SUMMARY ..... **xxiii**
ÖZET ..... **xxvii**
1. INTRODUCTION **1**
1.1 Motivation 1
1.2 Limitations of Traditional Feature Engineering 2
1.2.1 Manual design and human bias 2
1.2.2 Scalability challenges 3
1.2.3 Lack of systematic evaluation 4
1.3 Emergence of Generative and Agent-Based Systems 4
1.4 Research Gap 5
1.5 Research Questions 6
1.6 Contributions of the Thesis 6
1.7 Thesis Organization 8
2. LITERATURE REVIEW **11**
2.1 Feature Engineering in Financial Time Series 11
2.1.1 Technical indicators and price-based features 12
2.2 Automated Feature Engineering 13
2.2.1 Rule-based and transformation approaches 13
2.2.2 Genetic programming for alpha factor discovery 13
2.2.3 AutoML frameworks and neural architecture search 14
2.3 Large Language Models in Quantitative Finance 15
2.4 Agent-Based Learning Systems 16
2.4.1 Multi-agent architectures and coordination 17
2.4.2 Memory mechanisms and reflection 18
2.4.3 Feedback-driven learning and self-improvement 19
2.4.4 Alpha mining and signal generation 20
2.5 Prompt Engineering and Conditioning Strategies 20
2.5.1 Zero-shot and few-shot prompting 20
2.5.2 Chain-of-thought and structured reasoning 21
2.5.3 ReAct: Reasoning and acting frameworks 21
2.6 Security Considerations in Multi-Agent Systems 22

2.7 Research Gaps and Opportunities 22
3. THEORETICAL BACKGROUND OF AGENTIC AI SYSTEMS 23
3.1 Foundations of Large Language Models 23
3.1.1 What are large language models? 23
3.1.2 Emergent capabilities and fundamental limitations 24
3.2 Generative AI as Precursor to Agency 25
3.3 AI Agents: Definition, Properties, and Architecture 26
3.3.1 Distinguishing characteristics of AI agents 27
3.3.2 Agent architectures 27
3.3.3 Architectural components of an AI Agent 28
3.4 The Conceptual Leap: From AI Agents to Agentic AI 29
3.4.1 Defining Agentic AI 30
3.5 Architecture of Agentic AI Systems 30
3.5.1 The orchestration layer 31
3.5.2 Specialized agent ensembles 31
3.6 LangGraph and StateGraph: Technical Architecture 32
3.6.1 StateGraph fundamentals 32
3.6.2 Node communication and message passing 33
3.6.3 StateGraph implementation patterns 33
3.6.4 Checkpointing and persistence 34
3.6.5 Comparison of agent orchestration frameworks 34
3.7 Memory Architecture in Agentic Systems 35
3.7.1 Short-term memory (working memory) 36
3.7.2 Long-term memory (knowledge base) 36
3.7.3 Episodic memory (interaction history) 37
3.7.4 Context management and integration 38
3.8 Learning Paradigms in LLM-Powered Agents 38
3.8.1 Zero-shot learning 38
3.8.2 Few-shot learning 39
3.8.3 In-context learning 39
3.8.4 Human-in-the-loop (HITL) learning 40
3.9 Feedback Mechanisms in Agentic Systems 40
3.9.1 Performance feedback 41
3.9.2 Validation feedback 41
3.9.3 Category-aware feedback 42
3.10 Tool Integration and Function Calling 43
3.10.1 Tool invocation mechanisms 43
3.10.2 Types of tools 44
3.11 Reasoning and Planning in Agentic Systems 44
3.11.1 Reasoning and acting 45
3.11.2 Chain-of-thought prompting 45
3.11.3 Reflexion and self-critique 45
3.11.4 Tree-of-thoughts planning 46

3.12 Planning Algorithms in Agentic Systems 46
3.12.1 Structured planning approaches 46
3.13 Multi-Agent Systems and Orchestration 47
3.13.1 The coordinator-worker-delegator (CWD) model 47
3.13.2 Communication and coordination protocols 48
3.13.3 Orchestration frameworks 49
3.14 Sequential and Parallel Processing Patterns 49
3.14.1 Sequential processing 50
3.14.2 Parallel processing 50
3.14.3 Performance optimization 50
3.15 Reflection and Meta-Reasoning in Agents 50
3.16 Challenges and Limitations of Agentic AI 51
3.16.1 Hallucination and inherited LLM limitations 52
3.16.2 Coordination failure and error cascades 52
3.16.3 Emergent behavior and unpredictability 52
3.17 Solution Strategies and Architectural Best Practices 53
3.18 Situating the Study Within the Agentic AI Framework 54
3.18.1 Agentic approach to feature engineering 54
4. METHODOLOGY AND EXPERIMENTAL RESULTS 57
4.1 Case 0: Traditional Feature Engineering (Baseline) 57
4.1.1 Data sources and feature set 58
4.1.2 Asset universe and data configuration 58
4.1.3 Data preprocessing pipeline 59
4.1.4 Correlation-based feature elimination 60
4.1.5 XGBoost feature importance scoring 61
4.1.6 Rolling window training and validation 62
4.1.7 Performance metrics 62
4.2 Case 0: Experimental Results and Analysis 64
4.2.1 Post-selection category distribution 64
4.2.2 Overall performance summary 65
4.2.3 Model accuracy distribution analysis 65
4.2.4 Return distribution analysis 66
4.2.5 Top-performing individual features 66
4.2.6 Category-level performance comparison 67
4.2.7 Long-Short directional asymmetry 68
4.2.8 Summary of key findings 69
5. CASE 1: AGENT-BASED GENERATIVE FEATURE ENGINEERING 71
5.1 System Architecture 72
5.1.1 Graph-based agent orchestration 73
5.1.2 Node configuration and LLM assignment 74
5.2 Multi-LLM Provider System 75
5.2.1 Provider detection and configuration 76
5.2.2 Evaluated LLM configurations 76

5.2.3 Token tracking and cost management 76
5.3 Feature Generation Pipeline 76
5.3.1 Structured output schema 77
5.3.2 Prompt engineering strategy 78
5.3.2 Code safety and execution constraints 79
5.4 Creativity Enhancement System 79
5.4.1 Mathematical concept seeds 80
5.4.2 Difficulty levels and temperature modulation 81
5.4.3 Dynamic category system 81
5.5 Four-Layer Deduplication System 82
5.5.1 Layer 1: Name check 82
5.5.2 Layer 2: Code hash 83
5.5.3 Layer 3: LLM self-verification 83
5.5.4 Layer 4: LLM semantic validation 84
5.6 Memory System Architecture 84
5.6.1 Short-term memory (state dictionary) 85
5.6.2 Long-term memory (SQLite database) 85
5.6.3 Episodic memory (performance feedback) 85
5.6.4 State restoration and checkpointing 86
5.7 Data Manipulation and Preprocessing 86
5.7.1 Feature execution 86
5.7.2 Look-ahead bias prevention 86
5.7.3 Data cleaning 86
5.8 Training and Evaluation Pipeline 87
5.8.1 XGBoost training configuration 88
5.8.2 Per-asset evaluation 88
5.8.3 Feature selection and best features update 88
5.9 Feedback Loop Mechanism 88
5.10 Experimental Protocol 89
5.10.1 Execution parameters 89
5.10.2 Controlled variables 89
5.10.3 Database isolation 90
5.11 Summary 90
6. COMPARATIVE ANALYSIS, KEY FINDINGS, AND CONCLUSION 93
6.1 Analysis Overview and Data Preparation 93
6.2 Overall LLM Performance Comparison 94
6.2.1 Model accuracy 95
6.2.2 Model return 95
6.2.3 Multi-metric radar analysis 96
6.2.4 Detailed metric comparison 97
6.3 Performance Heatmap and Ranking Analysis 98
6.4 Feature Performance Distributions 98
6.4.1 Accuracy distribution 99
6.4.2 Return distribution 100

6.4.3 F1 score distributions 100
6.5 Feature Category Analysis 100
6.5.1 Category-level performance 101
6.5.2 Category distribution by LLM 102
6.6 Metric Correlation Analysis 102
6.7 Winner Analysis and Composite Ranking 104
6.7.1 Metric wins 105
6.7.2 Composite performance score 105
6.7.3 Head-to-head comparison 105
6.8 Top-Performing Individual Features 106
6.9 Key Findings 107
6.10 Conclusion 108
6.11 Future Works 111
REFERENCES 115
CURRICULUM VITAE 119

ABBREVIATIONS

AI	**:** Artificial Intelligence
API	**:** Application Programming Interface
AutoML	**:** Automated Machine Learning
CoT	**:** Chain-of-Thought
CWD	**:** Coordinator-Worker-Delegator
DAG	**:** Directed Acyclic Graph
FE	**:** Feature Engineering
GP	**:** Genetic Programming
GPT	**:** Generative Pre-trained Transformer
HITL	**:** Human-in-the-Loop
ICL	**:** In-Context Learning
LLM	**:** Large Language Model
MCTS	**:** Monte Carlo Tree Search
ML	**:** Machine Learning
NLP	**:** Natural Language Processing
OHLCV	**:** Open-High-Low-Close-Volume
ReAct	**:** Reasoning and Acting
S&P	**:** Standard & Poor's
SHA-256	**:** Secure Hash Algorithm 256-bit
SQL	**:** Structured Query Language
ToT	**:** Tree-of-Thoughts
XGBoost	**:** eXtreme Gradient Boosting

LIST OF TABLES

Page

Table 3.1 : Comparative taxonomy of Generative AI, AI Agents, and Agentic AI.. **26**
Table 3.2 : Comparison of agent orchestration frameworks. **35**
Table 3.3 : Comparison of primary reasoning frameworks used in Agentic AI. **35**
Table 3.4 : Comparative overview of major Agentic AI orchestration frameworks. **49**
Table 3.5 : Key challenges in Agentic AI systems **51**
Table 4.1 : Feature distribution by category and source library..... **58**
Table 4.2 : Feature selection pipeline stages and dimensional reduction **61**
Table 4.3 : XGBoost classifier hyperparameter configuration **62**
Table 4.4 : Performance metric definitions..... **63**
Table 4.5 : Post-selection category distribution compared to original..... **64**
Table 4.6 : Descriptive statistics for Case 0 selected features **65**
Table 4.7 : Distribution of model accuracy across 40 selected features. **66**
Table 4.8 : Distribution of model returns across 40 selected features. **66**
Table 4.9 : Top 10 features ranked by model accuracy **67**
Table 4.10 : Category-level average performance metrics **68**
Table 4.11 : Long versus Short directional performance comparison **68**
Table 4.12 : Summary of key findings from Case 0 baseline analysis **69**
Table 5.1 : Multi-agent node configuration (Claude Opus 4.6 preset) **74**
Table 5.2 : Evaluated LLM configurations for feature generation **76**
Table 5.3 : Selected creativity seeds from the pool of 22 mathematical concepts.... **80**
Table 5.4 : Difficulty levels with temperature boost and instruction..... **81**
Table 6.1 : Mean performance metrics across top 20 features per LLM source..... **95**
Table 6.2 : Performance metrics by feature category..... **101**
Table 6.3 : Top 10 features by model accuracy across all 120 features..... **106**

LIST OF FIGURES

Page

Figure 3.1 : LangGraph agent-tools workflow..... **34**
Figure 3.2 : Working memory class, structure, and session lifecycle **36**
Figure 3.3 : Episodic memory class, structure and retrieval mechanism **37**
Figure 5.1 : Structural comparison between Case 0 (Traditional) and Case 1 (Agent-based) **72**
Figure 5.2 : Complete system architecture showing eight specialized agents **73**
Figure 5.3 : Multi-LLM provider system with automatic detection and configuration **75**
Figure 5.4 : Feature generation pipeline showing preparation and creativity stages. **77**
Figure 5.5 : Creativity enhancement pipeline with seed injection and difficulty modulation **80**
Figure 5.6 : Four-layer deduplication system with cascading validation **82**
Figure 5.7 : Three-tier memory architecture: short-term, long-term, and episodic .. **84**
Figure 5.8 : Training and evaluation pipeline with rolling window XGBoost **87**
Figure 6.1 : Executive dashboard summarizing LLM performance across metrics. **94**
Figure 6.2 : Normalized radar chart comparing six approaches across ten metrics.. **96**
Figure 6.3 : LLM performance across all metrics with best performer highlighted. **97**
Figure 6.4 : Performance heatmap showing absolute values and rankings **98**
Figure 6.5 : Feature performance distributions by LLM source across six metrics.. **99**
Figure 6.6 : Feature category analysis return, accuracy, and count distribution..... **101**
Figure 6.7 : Pearson correlation matrix across ten performance metrics **103**
Figure 6.8 : Winner analysis — metric wins, composite scores, and head-to-head results **104**

AN AGENT-BASED FRAMEWORK FOR GENERATIVE FEATURE ENGINEERING IN FINANCIAL MARKETS

SUMMARY

One of the biggest problems in quantitative finance is automating feature engineering. Traditional practitioners relied on curated libraries of carefully constructed technical indicators. These indicators are mathematical transformations of price and volume data. They have been refined over decades of market experience. Although traditional indicator sets are interpretable and grounded in theory, they are only as creative as their creators. Their dynamic-to-static nature provides insufficient flexibility to address the challenges posed by market conditions. This thesis proposes and evaluates a radically different approach. It introduces an autonomous multi-agent system powered by large language models. This solution reduces feature engineering to a generative discovery problem. New mathematical transforms of open-high-low-close-volume data can be systematically generated. The predictive power is validated through rolling-window out-of-sample testing.

To illustrate the survey, the research is structured around two experimental cases regarding a common prediction task, namely predicting whether each S&P 100 equity, selected for their high mean daily trading volume, will move directionally open-to-close on any given day. The target to be predicted is therefore a binary classification problem, given the cross-section, with all features lagged by one trading day to remove look-ahead bias. Each feature is evaluated using an XGBoost classifier trained on a rolling six-month window and tested in the preceding month. The above process repeats for twelve consecutive periods and includes all assets within the specified universe.

The first experimental case simply establishes a "traditional" baseline based on a complete set of technical indicators from the factored library in Microsoft Qlib Alpha158 and Achelis's Technical Analysis from A to Z. These indicators can take many forms, such as Trend, Momentum, Volatility and Volume. The analysis uses a multi-stage dimensionality reduction pipeline, consisting of Pearson correlation filtering and iterative gain-based elimination using XGBoost. This method substantially reduces the size of the initial indicator pool to less than ten percent of what it began with, keeping only the most relevant features for inclusion in the final set. These few chosen features are the basis for evaluating any agentic system.

These baseline results confirm that classical technical indicators selected in a controlled way do have, albeit limited, predictive power. This performance holds true across many assets and time frames; however, it sits right at the edge of no better than chance. Interestingly, Volume Rate of Change indicators consistently rank at the very top, whilst Volatility features represent the most significant category after removing redundancy. However, a major directional asymmetry emerges as the core structural limitation, with much better reliability in the positive direction vs. the negative, and thus a negative mean return for shorts. This long bias not only stems from the evaluation period's bullish nature but also underscores the inherent difficulties in

predicting downward moves using classical methods. As a result, it creates an important standard that the agentic system is meant to surpass.

In the second experimental case, we introduce an eight-node multi-agent setting implemented with LangGraph's StateGraph. These specialized nodes handle unique tasks such as workflow routing, quote assignment, and market data retrieval, feature code synthesis, uniqueness checking, and deduplication of new features, execution of the code created in step 3 to generate signals/payment, as shown in steps 4 & 5, which are then trained with quotes for performance evaluation. This is an important difference, as the Feature Generator node uses a large language model with a higher sampling temperature to generate Python code that computes new OHLCV transformations. It runs in an iterative process over 30 loops, generating a candidate feature set at each iteration. This iterative approach allows the performance feedback of earlier loops to inform the creative direction of later ones.

To ensure that the generated feature pool does not include duplicates, I employ a four-layer deduplication mechanism at increasingly deeper levels: exact-name matching; hash-based syntactic camouflage detection; LLM-powered mathematical uniqueness verification; and semantic similarity analysis in the expression space. Every generation prompt includes randomly chosen mathematical concepts sampled from a broad space of subject areas, including fractal geometry, information theory, and spectral analysis, so that the system does not hone in on familiar feature archetypes over successive iterations. This gradation of difficulty also applies to the sampling temperature and novelty requirements, meaning that the system is forced to explore genuinely new mathematical territory throughout later iterations.

As the Feature Generator, five large language model configurations are evaluated under similar experimental conditions, testing with ChatGPT 4o (with and without its mini version), ChatGPT 5, ChatGPT 5.2, and Claude Opus 4.6. The cross-source deduplication step selects the features with the largest scores from each configuration, which are then combined with the Case 0 baseline set to provide results for comparative evaluation. All features are evaluated in the same environment to ensure that any discrepancies between types of features observed come from the quality of generated features and not variance in methodology.

These configurations all improve upon the classification accuracy and simulated model return achieved by Case 0, confirming the main hypothesis of this thesis. ChatGPT 4o outperforms in the majority of evaluation metrics, producing features that yield the best mean accuracy and highest economic returns. The single best feature it produces reflects accuracy and return figures that far exceed those in the traditional library. ChatGPT 4o mini is hot on its heels, delivering the best directional profile across all configurations. The other configurations show gradual, but still positive, improvements over the baseline. Interestingly, ChatGPT 4o beats both of the latest GPT models, suggesting that creative diversity in the context of generation is more important than model size for this task.

Although the magnitude varies, there is a systematic long bias across all configurations. Claude Opus 4.6 stands out for a strong directional bias toward upward predictions and weak performance in providing a reliable signal for short positions. Conversely, ChatGPT 4o mini provides the most balanced predictions and demonstrates the most powerful short-side performance across all LLM configurations. Meanwhile, momentum features are continually hailed as the best overall performers across the top-ranked models, corroborating the well-established

fact that opinion is expressed in monetary investments through long or short signals, and that such a generative system overall provides such evidence without explicit domain guidance. Additionally, different models exhibit distinct preferences for categories, indicating substantial disparities in their use of training data. The main caveats of the study are that it examines large-cap U.S. equities in a generally bull market, has not accounted for transaction costs in the performance numbers, and that LLM reasoning is opaque, making formal attribution of predictive power to particular mathematical mechanisms difficult.

Keywords: Agent-Based Feature Engineering, Large Language Models, Quantitative Finance, Technical Analysis, XGBoost, Multi-Agent Systems, LangGraph, Generative AI, Stock Price Prediction, S&P 100

FİNANSAL PİYASALARDA SİSTEMATİK ÜRETİCİ DEĞİŞKEN MÜHENDİSLİĞİ İÇİN YAPAY ZEKÂ AJANLARI TEMELLİ BİR YAKLAŞIM

ÖZET

Finansal piyasalarda özellik mühendisliğinin otomasyonu, niceliksel finansın tarihsel olarak çözümsüz kalan temel problemlerinden birini oluşturmaktadır. Alanda çalışan pratisyenler, ham fiyat ve hacim verisinin önceden tanımlanmış matematiksel dönüşümlerine dayanan, elle hazırlanmış teknik gösterge kütüphanelerine uzun süredir güvenmektedir. Bu klasik yaklaşım yorumlanabilirlik ve teorik sağlamlık açısından değer taşısa da, içerdiği dönüşümler yazarların hayal gücüyle sınırlıdır ve değişen piyasa koşulları karşısında büyük ölçüde statik kalmaktadır. Bu tez, temelden farklı bir yaklaşım önermekte ve değerlendirmektedir: büyük dil modelleriyle desteklenen özerk bir çok etmenli sistem, özellik mühendisliğini sabit bir katalogdan seçim problemi olarak değil, sistematik bir keşif süreci olarak ele almakta; açılış-yüksek-düşük-kapanış-hacim verisinin özgün matematiksel dönüşümlerini otuz yinelemeli döngüde icat etmekte ve bunların öngörücü değerini kayan pencere örnek dışı testlerle ölçmektedir.

Araştırma, ortak bir tahmin görevi üzerinde yapılandırılmış iki deneysel vakadan oluşmaktadır: ortalama günlük işlem hacmine göre seçilmiş yüksek likiditeli S&P 100 hisselerinin günlük açılış-kapanış fiyat yönü tahmini. Yüksek likiditenin tercih edilmesinin başlıca nedeni, bu tür varlıkların gün içi fiyat hareketlerinin daha belirgin olması ve veri kalitesi sorunlarına karşı daha az duyarlı bulunmasıdır. Tahmin hedefi ikili sınıflandırma olarak tanımlanmakta olup, ileriye dönük önyargıyı önlemek amacıyla tüm özellik değerleri bir işlem günü öne kaydırılmaktadır. Her özellik, altı aylık kayan pencereyle eğitilen ve ardından gelen ay üzerinde doğrulanan bir XGBoost ikili sınıflandırıcısıyla değerlendirilmekte; bu süreç on iki ardışık dönem ve tüm varlık evreni üzerinde tekrarlanmaktadır. Kayan pencere yaklaşımının seçilmesinin ardındaki temel gerekçe, modelin farklı piyasa rejimlerinde nasıl davrandığını anlamak ve performans değerlendirmesinin belirli bir zaman dilimine özgü koşullardan kaynaklanmamasını güvence altına almaktır.

Birinci deneysel vaka, iki köklü kaynaktan derlenen kapsamlı bir teknik gösterge havuzuyla geleneksel referans noktasını oluşturmaktadır: Microsoft'un Qlib Alpha158 kütüphanesi ve Achelis'in A'dan Z'ye Teknik Analiz eseri. Bu göstergeler birlikte Trend, Momentum, Volatilite ve Hacim ailelerini kapsamaktadır. Söz konusu kaynakların tercih edilmesinin temel nedeni, her ikisinin de niceliksel araştırmada yaygın biçimde kullanılan ve matematiksel açıdan iyi tanımlanmış göstergeler sunmasıdır; bu özellik, sonuçların yorumlanabilirliği ve yeniden üretilebilirliği açısından önem taşımaktadır. Pearson korelasyonunu filtrelemesi ve yinelemeli XGBoost kazanım tabanlı elmayı birleştiren çok aşamalı bir boyut azaltma hattı, bu havuzu yüzde doksanın üzerinde bir oranla küçülterek sıkıştırılmış bir nihai sete indirgemektedir. Bu indirgeme süreci yalnızca teknik bir zorunluluk değil, aynı zamanda hangi gösterge türlerinin öngörücü açıdan hayatta kaldığını anlamak için de

bilgilendirici bir analiz niteliği taşımaktadır. Hayatta kalan bu özellikler, ajansal sistemin karşılaştırılacağı referans özellik setini oluşturmaktadır.

Referans sonuçları, dikkatli seçime tabi tutulmuş klasik teknik göstergelerin gerçek ama mütevazı bir öngörücü bilgi içerdiğini ortaya koymaktadır. Performans varlıklar ve dönemler genelinde tutarlı olmakla birlikte rastlantısal tahminin hemen üzerinde seyretmektedir; bu durum, finansal piyasalarda tahmin görevinin doğasında var olan zorluğu yansıtmaktadır. Artıklık eliminesinin ardından Volatilite özellikleri hayatta kalan en büyük kategoriyi oluştururken, Hacim Değişim Oranı göstergeleri bireysel performans sıralamasında belirgin biçimde öne çıkmaktadır. Bu bulgu, yüksek işlem hacmine sahip hisselerde hacim dinamiklerinin kısa vadeli fiyat yönü üzerindeki belirleyici rolüne işaret etmektedir. Bununla birlikte belirgin bir yönsel asimetri, referans noktasının temel yapısal sınırlaması olarak ortaya çıkmaktadır: model, aşağı yönlü hareketlere kıyasla yukarı yönlü hareketleri çok daha güvenilir biçimde tahmin etmekte ve kısa pozisyonlarda negatif ortalama getiri üretmektedir. Bu Uzun önyargısı, değerlendirme döneminin hâkim yükseliş karakterini ve klasik sinyallerle aşağı yönlü hareket tahmininin yapısal zorluğunu yansıtmakta; aynı zamanda ajansal sistemin aşması beklenen temel kıyaslamayı belirlemektedir.

İkinci deneysel vaka, LangGraph'ın StateGraph yapısı kullanılarak uygulanan sekiz düğümlü bir çok etmenli çerçeveyi tanıtmaktadır. Özelleşmiş düğümler sırasıyla iş akışı yönlendirmesini, piyasa verisi alımını, özellik kodu sentezini, özgünlük doğrulamasını, tekilleştirmeyi, kod yürütmeyi, model eğitimini ve performans değerlendirmesini üstlenmektedir. Özellik Oluşturucu düğümü, yüksek örnekleme sıcaklığında çalışan büyük bir dil modeli aracılığıyla özgün OHLCV dönüşümlerini hesaplayan çalıştırılabilir Python kodu üretmektedir. Sistem otuz döngü boyunca yinelemeli olarak çalışmakta; önceki döngülerin performans geri bildirimi, sonraki döngülerin yaratıcı yönelimini şekillendirmektedir. Bu yinelemeli tasarım, sistemin yalnızca özellik üretmekle kalmayıp kendi geçmiş performansından öğrenerek zamanla daha bilgilendirilmiş bir arama stratejisi geliştirmesine olanak tanımakta ve bunu doğrudan çerçevenin mimari özüne yerleştirmektedir.

Üretilen özellik havuzunun bütünlüğü, giderek derinleşen soyutlama düzeylerinde çalışan dört katmanlı bir tekilleştirme mekanizmasıyla güvence altına alınmaktadır: tam ad eşleştirmesi, söz dizimsel kılık değiştirmeyi tespit eden karma tabanlı karşılaştırma, büyük dil modeli destekli matematiksel özgünlük doğrulaması ve ifade uzayında anlamsal benzerlik analizi. Bu dört katmanlı yapı, mevcut literatürdeki pek çok otomasyon çerçevesinin yalnızca yüzeysel düzeyde kontrol ettiği artıklık sorununu daha derinlemeli biçimde ele almakta ve kabul edilen her özelliğin gerçek bir matematiksel katkı teşkil etmesini güvence altına almaktadır. Sistemin otuz döngü boyunca tanıdık özellik arketiplerine yakınsamasını önlemek amacıyla her üretim istemi, fraktal geometri, bilgi teorisi ve spektral analiz gibi geniş bir alandan rastgele seçilen matematiksel kavram tohumlarıyla beslenmektedir. Kademeli zorluk düzeyleri örnekleme sıcaklığını ve yenilik gereksinimlerini eş zamanlı olarak modüle etmekte; bu sayede sistemin geç döngülerde bile gerçekten yeni matematiksel bölgeler keşfetmeyi sürdürmesi sağlanmaktadır. Bu yaratıcılık geliştirme mekanizması, üretken yapay zekâ sistemlerinde sıkça gözlemlenen üretim çöküşü sorununa — sistemin zamanla tekrarlayan ve öngörülebilir çıktılar üretmeye başlaması durumuna — karşı geliştirilmiş yapısal bir yanıt niteliği taşımaktadır.

Beş büyük dil modeli yapılandırması, diğer koşullar sabit tutularak Özellik Oluşturucu konumunda değerlendirilmektedir: ChatGPT 4o, ChatGPT 4o mini, ChatGPT 5,

ChatGPT 5.2 ve Claude Opus 4.6. Bu yapılandırmaların yalnızca Özellik Oluşturucu konumunda değiştirilmesi ve altyapının sabit tutulması, gözlemlenen farklılıkların oluşturucu modelin doğasına atfedilmesini kolaylaştırmakta; diğer sistem bileşenlerinden kaynaklanan karıştırıcı etkileri büyük ölçüde devre dışı bırakmaktadır. Birden fazla yapılandırmada bağımsız olarak ortaya çıkan özellikleri kaldırmaya yönelik çapraz kaynak tekilleştirmesinin ardından, her yapılandırmanın en iyi özellikleri seçilerek Vaka 0 referans setiyle birleştirilen nihai karşılaştırmalı veri seti oluşturulmaktadır. Tüm özellikler özdeş koşullar altında değerlendirilmekte; böylece gözlemlenen farklılıkların metodolojik varyasyondan değil, üretilen özelliklerin kalitesinden kaynaklandığı güvence altına alınmaktadır.

Beş büyük dil modeli yapılandırmasının tamamı, hem sınıflandırma doğruluğu hem de simüle edilmiş model getirisi açısından Vaka 0 referansını geride bırakmıştır; bu bulgu, ajansal özellik mühendisliğinin klasik teknik göstergelerden üstün öngörücü güce sahip özellikler ürettiği yönündeki temel hipotezi doğrulamaktadır. ChatGPT 4o, değerlendirme metriklerinin büyük bölümünde öne çıkmakta; hem en yüksek ortalama doğruluğu hem de en büyük ekonomik getirileri sunan özellikler üretmekte ve en iyi bireysel özelliği geleneksel kütüphanedeki her göstergeyi büyük bir farkla geride bırakmaktadır. ChatGPT 4o mini tüm yapılandırmalar arasında en dengeli yönsel profiliyle ikinci sıraya yerleşmektedir. Kalan yapılandırmalar sırasıyla azalan ama tutarlı olarak pozitif iyileştirmeler sunmaktadır. Dikkat çekici bir bulgu olarak ChatGPT 4o'nun daha yeni GPT versiyonlarını da geride bırakması, üretim bağlamı içindeki yaratıcı çeşitliliğin bu görevde ham model kapasitesi ve güncelliğinden daha belirleyici olabileceğini düşündürmektedir. Bu sonuç, büyük dil modellerinin niceliksel finans uygulamalarında değerlendirmesinde model büyüklüğünü tek ölçüt olarak alan yaklaşımları sorgulamakta ve üretken sistemlerin tasarımında çeşitlilik ile keşif kapasitesini ön plana çıkarmaktadır.

Yapılandırmalar arasındaki performans farklılıkları yalnızca ortalama değerler düzeyinde değil, bireysel özellik dağılımları açısından da incelendiğinde ilgi çekici örüntüler ortaya çıkmaktadır. ChatGPT 4o, en geniş özellik kalitesi dağılımını sergilemekte; hem son derece güçlü bireysel özellikler hem de görece zayıf olanlar üretmektedir. Bu geniş dağılım, modelin yaratıcı çeşitlilik kapasitesinin bir yansıması olarak değerlendirilebilir: arama alanı geniş tutuldukça istisnai keşifler mümkün olmakta, ancak aynı genişlik kaçınılmaz olarak daha az başarılı denemeleri de beraberinde getirmektedir. Buna karşın ChatGPT 5 ve 5.2, çok daha sıkı kümelenmiş dağılımlar sergilemekte; tutarlı ama daha az farklılaşmış özellikler üretmektedir. Bu gözlem, özellik keşfi bağlamında üretken yapay zeka sistemlerinin tasarımı açısından önemli bir değiş tokuşa işaret etmektedir: yüksek ortalama tutarlılık ile aykırı değer üretme kapasitesi arasında bir denge kurulması gerekmekte ve hangi uç noktanın daha değerli olduğu nihai hedefin niteliğine göre farklılaşmaktadır.

Tüm yapılandırmalarda sistematik bir Uzun önyargısı gözlemlenmekle birlikte şiddeti yapılandırmalar arasında belirgin biçimde farklılaşmaktadır. Claude Opus 4.6, yukarı yönlü hareket tahmininde güçlü bir performans sergilerken güvenilir kısa yönlü sinyaller üretememekte ve en belirgin yönsel dengesizliği ortaya koymaktadır. ChatGPT 4o mini ise en simetrik tahminleri sunan ve büyük dil modeli yapılandırmaları arasında en güçlü kısa yönlü performansı gösteren yapılandırma olarak öne çıkmaktadır. Bu bulgu, pratik ticaret stratejileri açısından önemli sonuçlar doğurmaktadır: yalnızca uzun pozisyonlara odaklanan stratejiler için neredeyse tüm yapılandırmalar kullanılabilirken, iki yönlü pozisyon almayı gerektiren stratejiler için yapılandırma seçimi kritik bir tasarım kararı haline gelmektedir. Momentum

özellikleri tüm yapılandırmalarda üst sıralara hâkim olmakta ve özellik setinin en büyük payını oluşturmaktadır; bu durum, hisse senedi piyasalarındaki yerleşik momentum anomalisini bağımsız biçimde doğrulamakta ve üretken sistemin bu anomaliyi açık alan talimatı olmaksızın yeniden keşfetme kapasitesini göstermektedir. Farklı büyük dil modellerinin belirgin kategori tercihleri sergilemesi de dikkat çekmekte ve her modelin eğitim bilgisini nasıl farklı biçimlerde aktardığına işaret etmektedir.

Çalışmanın temel sınırlamaları birkaç başlık altında ele alınabilir. Her şeyden önce, değerlendirme ağırlıklı olarak yükseliş eğilimli bir dönemde büyük ölçekli ABD hisse senetleriyle sınırlı kalmaktadır; bu durum, tespit edilen Uzun önyargısının bir bölümünün üretken yaklaşımın kendisinden değil yapısal piyasa koşullarından kaynaklanıyor olabileceğini düşündürmektedir. Farklı piyasa rejimlerinde — özellikle uzun süreli düşüş dönemlerinde veya yüksek volatilite ortamlarında — benzer bir çerçevenin nasıl davranacağı, ileriki çalışmalar için açık kalan önemli bir soru olarak durmaktadır. Öte yandan raporlanan getiriler işlem maliyetlerini, alım-satım farkını ve emir yürütme gecikmesini dışarıda bırakmaktadır; dolayısıyla bu değerler gerçekçi bir ticaret stratejisinde elde edilebilecek performansın üst sınırını temsil etmektedir. Son olarak, büyük dil modellerinin kara kutu niteliği, üretilen özelliklerin matematiksel gerekçesinin yalnızca modellerin sağladığı doğal dil açıklamaları kadar şeffaf olduğu anlamına gelmekte ve belirli matematiksel mekanizmalara resmi performans atamayı güçleştirmektedir. Bu şeffaflık eksikliği yalnızca akademik bir sınırlama değil, aynı zamanda gerçek dünya uygulamalarında model denetimi ve uyumluluk gereksinimleri açısından da pratik bir zorluk oluşturmaktadır.

Gelecek araştırma yönleri açısından en doğrudan uzantı, özellik üretim sürecinin küratörlüğü yapılmış finansal alan literatürüyle temellendirilmesidir. Bu yaklaşım, büyük dil modelinin yalnızca genel eğitim bilgisine değil, belirli bir teknik analize veya akademik literatüre erişerek özellik üretmesini sağlayabilir ve böylece hem üretilen özelliklerin kalitesini hem de yorumlanabilirliğini artırabilir. Bunun yanı sıra geleneksel ve büyük dil modeli kaynaklı özellikleri bir arada kullanan topluluk stratejilerinin araştırılması, iki yaklaşımın tamamlayıcı güçlerini birleştirme potansiyeli taşımaktadır. Çerçevenin emtia, sabit gelir ve çapraz varlık uygulamalarına genişletilmesi ise hem sistemin genellenebilirliğini sınamak hem de teknik analizin daha az olgunlaşmış olduğu piyasalarda görece avantajın daha belirgin hale gelebileceğini araştırmak açısından değerli bir yön oluşturmaktadır.

Anahtar Kelimeler: Ajans Tabanlı Özellik Mühendisliği, Büyük Dil Modelleri, Niceliksel Finans, Teknik Analiz, XGBoost, Çok Etmenli Sistemler, LangGraph, Üretken Yapay Zeka, Hisse Senedi Fiyatı Tahmini, S&P 100

1. INTRODUCTION

Financial markets are a peculiar space where the abundance of data and forecasting challenges persist as challenges for quantitative science. Although research efforts have been extensive and numerous technical indicators have been developed over several decades, the question of reliably forecasting short-term price movements remains unanswered. The problem isn't the lack of data; it's in knowing which version or versions of that data are truly predictive. This is new research because, instead of focusing on the choice of existing indicators, it examines the potential of large language models to create indicators. In subsequent chapters, a multi-agent framework based on this novel approach is described and evaluated against a strong classical baseline; the results are interpreted to reveal the properties and limitations of LLM-based feature engineering in financial markets.

1.1 Motivation

Quantitative researchers interested in developing systematic approaches to extracting predictive signals from historical price data have long gravitated to financial markets. One major obstacle in this endeavor is feature engineering: the process of transforming raw open, high, low, close, and volume (OHLCV) data into structured inputs that reliably signal price-change behavior. This transformational process has been guided over the past few decades by a growing body of literature on technical analysis, in which the indicators described serve as the primary toolbox of quantitative practitioners.

The rationale for this thesis stems from the fact that conventional approaches to engineering features reach a fundamental ceiling. The Microsoft Qlib platform (Yang et al., 2020) offers a library of both calculated and custom-calculated assets, known as Alpha158, comprising trend, momentum, volatility, and volume indicators; in total, 431 indicator types, including those that were originally cataloged by Achelis (2001). All these indicators have been well studied, are interpretable, and validated by professionals. Yet, they all have one key limitation in common: their formulation

encodes the distilled knowledge of humans at a particular moment in time, and the space of possible mathematical transformations they explore is vastly more limited than the potential space of transformations that could yield useful insights.

That is the motivation behind this thesis: as both LLMs and deep learning models have ingested a vast body of quantitative finance literature, academic papers across various sciences, and programming source code, they can be prompted to construct mathematical transformations of OHLCV data that outperform classical indicators created by expert human minds. To address this question, the experimental work presented here aims to contribute towards an autonomous multi-agent framework that encourages novel feature generation through structured cooperation between LLMs and the subsequent evaluation of those features under rigorous rolling-window validation conditions.

The larger context for this work is the rapid progress seen in agentic AI systems — architectures in which multiple specialized language model nodes operate autonomously, cooperating to perform tasks that an individual LLM cannot accomplish on its own. Frameworks such as LangGraph play a critical role in DAG workflows, where each node has a specific purpose, the output from one node determines the behavior of the next, and feedback from downstream evaluations is used to improve upstream generation. This architecture is applied to the original feature series to produce a self-improving pipeline that generates novel characteristics in financial markets, thereby enhancing the significance of the generated features at each iteration.

1.2 Limitations of Traditional Feature Engineering

Despite the technical indicators discussed in the introductory section and their sophistication, they exhibit three structural limitations that impose a ceiling on feature engineering in financial time series. These limitations both shape the research design and highlight the unique contributions of this thesis.

1.2.1 Manual design and human bias

Nearly two centuries ago, classical technical indicators were invented by practitioners who blended their intuition about the market with mathematical formalization. Murphy (1999) and Achelis (2001) enumerated hundreds of these, but the design process is

ultimately governed by the conceptual categories available to human intermediaries, such as trend, momentum, oscillation, and volume. Because of this cognitive frame, the resulting indicators are biased towards particular classes of transformations and leave a considerable part of the mathematical landscape unexplored. Properties that lie outside these learned conceptual partitions, no matter how statistically significant their effect may be, are unlikely to be reached through manual design.

Confirmation bias exacerbates the situation. Practitioners tend to prefer indicators whose behavior is consistent with one or more prevailing theories of price behavior (mean reversion, trend continuation, etc.) and which are generally validated against historical periods that reinforce the anticipated behaviors at the indicator level. As a result, the process yields a set of metrics that reinforces existing assumptions about market behavior rather than encouraging an objective pursuit of predictive formulations.

1.2.2 Scalability challenges

Even after removing such human bias, manually designing features remains a non-scalable process. An independent researcher may create hundreds of indicators as part of a programmatic research effort; the Alpha158 library itself is the product of years of community collaboration. By contrast, the list of mathematical transformations that can be applied to a multi-dimensional time series is practically infinite. Automated feature generation methods have demonstrated that this large space can be traversed far more efficiently than traditional techniques allow. For example, Zhang et al. (2020) developed a platform called AutoAlpha that used hierarchical genetic algorithms to generate 434 unique formulaic alpha factors, achieving 12 times the coverage of traditional programming-based systems.

When applied to a system of 20 commonly traded S&P 100 equities with a daily prediction horizon, the time required to evaluate the introduction of each candidate feature through rolling-window training is substantial yet feasible. This challenge stems less from computation than from creativity: the need to develop a sufficiently varied set of candidate transformations. LLM-driven generation circumvents this limitation by leveraging the mathematical knowledge encoded in a pre-trained model's parameters.

1.2.3 Lack of systematic evaluation

Most technical analysis libraries are not built according to a systematic test of their predictive power. Indicators are generally included based on theoretical justification or historical application, not because they have been empirically demonstrated to provide better predictions than other potential indicators in standardized prediction tasks. Such an approach yields a set of highly redundant features; many indicators are correlated and convey essentially the same information. The baseline analysis described in this thesis clearly illustrates this point: Pearson correlation elimination reduces the initial set of 431 indicators to approximately 280. Applying XGBoost feature importance scores to the full set of 42 indicator datasets further reduces the list to 40 indicators, suggesting that most traditional indicators are either redundant or irrelevant to the target prediction task.

In contrast, a systematic evaluation framework uses a common objective criterion for each candidate feature: for example, how well the addition of a new feature improves the accuracy of a binary classifier predicting the daily open-to-close price direction on held-out data. This criterion is obvious, relevant in multiple fields, and closely related to the economic objective of directional trading.

1.3 Emergence of Generative and Agent-Based Systems

The recent introduction of advanced reasoning capabilities for large language models (LLMs) opens new opportunities for automating knowledge-intensive tasks. Scaling LLMs may simply lead to in-context learning and multi-step reasoning (Brown et al., 2020). These capabilities emerge not from explicit programming, but rather from the optimization objective Driving next-token prediction at Scale. These emergent properties have profound ramifications for feature engineering; for example, a sufficiently trained LLM that has absorbed many books on quantitative finance can exploit this information to recommend mathematical transformations without needing a formula.

Modern generative AI are intrinsically reactive; they generate responses from prompts, but do not store information from previous interactions to improve their outputs autonomously. Moving from generative AI to goal-seeking, or agentic AI requires one more structural layer, an additional mechanism whereby state can be retained across

multiple LLM calls and outputs are directed through specialized nodes and incorporated with feedback loops that allow the system to improve its performance over time (Bornet et al., 2025). One such important infrastructure is provided by LangGraph in the form of the StateGraph abstraction, which allows complex workflows to be modeled as directed graphs in which each node can read from and write to a shared state dictionary.

Recently, there has been a noticeable uptick in interest in applying LLM architectures to quantitative finance. Ding et al. (2024) conducted a systematic survey of LLM-powered trading agents, showing that they produced average annualized returns per tradable asset over each backtest interval that were roughly 15%–30% above those of traditional benchmark methods. However, they warned that these models might be overfitting to historical data. Li et al. (2025) described a novel Research and Development Agent (RD-Agent) system designed to automate research on financial predictive factors. Their results showed that LLM agents are able to independently generate hypotheses and test them with little human guidance. In contrast, Cao et al. (2025) introduced the Chain-of-Alpha framework to bridge LLM-based reasoning pathways with interpretable alpha factors. This thesis aims to fill this gap in the existing research and present a systematic empirical investigation of different LLM setups within a controlled environment.

1.4 Research Gap

Despite the growing body of literature on automated feature engineering and LLM-based financial agents, a key issue remains unaddressed. Current research mostly focuses either on benchmarking single trading agents or on validating automatically generated alpha factors. However, there is currently no study comparing the feature engineering capabilities across LLM providers and model scales under controlled experimental conditions. For instance, it remains unclear whether larger and more advanced LLMs reliably produce more predictive features than smaller ones, whether different model families exhibit different biases in the types of transformations they prefer, and whether the LLM-based generation approach consistently outperforms traditional technical indicators across the entire distribution of generated features rather than only at the tail end.

This thesis fills these gaps through a controlled experiment in which five LLM variants, ChatGPT 4o, ChatGPT 4o mini, ChatGPT 5, ChatGPT 5.2, and Claude Opus 4.6 act as feature generators within the same multi-agent architecture. Each configuration is evaluated against a fixed classical baseline composed of 431 indicators from Alpha158 and Achelis, using the same XGBoost classification model, a given universe of 20 assets, and a standardized rolling validation process covering the past 12 months. This design effectively isolates the influence of the LLM configuration while controlling for ancillary differences in evaluation methodology.

1.5 Research Questions

The thesis is structured around four research questions that together address the main contribution of this work:

RQ1. Can an autonomous multi-agent framework leveraging large language models produce novel mathematical operations as features for stock price direction prediction that outperform a comprehensive set of classical indicators across a range of predictive metrics spanning different classes?

RQ2. How do the quality, consistency, and distribution of features generated for prediction vary across different LLM configurations differing in model scale, provider, or architecture?

RQ3. What are the structural properties of the top-performing LLM-generated features compared to the classical baseline, and do they exhibit systematic biases in LLM-based feature generation, such as directional asymmetry between long and short predictive power?

RQ4. Does the quality of LLM-generated features increase proportionally with model scale, or do smaller, cheaper models produce competitive results?

1.6 Contributions of the Thesis

This thesis makes several important contributions to the domains of automated feature engineering and quantitative finance.

First, it presents a novel and comprehensive multi-agent feature engineering paradigm that has been carefully designed, implemented, and evaluated. This framework

introduces an original LangGraph StateGraph system, which leverages an eight-node state graph to implement structured output generation, a four-layer deduplication mechanism, and iterative feedback from rolling-window XGBoost evaluations. Importantly, this framework is provider-agnostic, supporting both the OpenAI and Anthropic model families through an automatic, prefix-based routing system. This architecture allows for the direct comparison of different providers within a single architectural context.

Second, this thesis provides an empirical benchmark comparing five LLM configurations against a classical baseline comprising 431 indicators on the task of predicting the daily open-to-close price direction for 20 S&P 100 stocks. As illustrated in Figure 6.1, all five LLM configurations outperform the traditional baseline in terms of model accuracy; among them, ChatGPT 4o achieves the highest mean return of 2.792, more than five times that of the baseline (0.523). This represents the first systematic comparison of multiple LLMs in a generative feature engineering setting for financial prediction.

Third, looking at feature category distributions in detail shows that momentum-type transformations account for ~48% of the top-performing features across LLMs. This indicates that LLMs use more momentum-driven financial understanding when generating out-of-sample signals. Furthermore, it is worth noting that LLM-based features are systematically long-biased: while both accuracy and Long-side returns show great improvements relative to the baseline, the Short-side performance remains an Achilles' heel across most configurations.

Fourth, the thesis introduces a systematic approach to agent-based generative feature engineering that generalizes beyond its application to the prediction task studied in this work. This combination of an iterative creativity-seeding mechanism, an organized Pydantic output schema, cross-asset importance averaging and a multi-layer deduplication protocol serves as a reusable experimental design. This can be adapted to specific assets, forecast horizons or evaluation criteria.

1.7 Thesis Organization

The rest of this thesis is organized as follows. Chapter 2 conducts an exhaustive survey of the relevant literature across four domains: feature engineering for financial time series; automated approaches to feature engineering, including genetic programming and AutoML; large language models applied within quantitative finance; and agent-based learning systems. Chapter 3 provides a theoretical overview of agentic AI, covering its key concepts, the architectural differences between generative AI and agentic systems, reasoning frameworks, orchestration frameworks that enable coordination among multiple agents, and related research.

Chapter 4 is divided into two parts that provide an overview of the methodology and experimental results. Part 1 (Case 0) details the traditional feature engineering pipeline, including data sourcing, the construction of a 431-indicator feature space, correlation-based filtering, XGBoost importance scoring, and rolling-window validation. Part 2 (Case 1) explores the multi-agent framework, discussing its eight-node architecture, large language model (LLM) configuration, feature generation pipeline, deduplication methods, and iterative evaluation protocol. It also reports the results for each LLM configuration. A subsequent comparative evaluation section measures performance across all six approaches using ten evaluation metrics, including directional analysis and feature category distribution.

Chapter 5 contributes the main methodology of this thesis: a multi-agent architecture that treats feature engineering as a generative problem. Under this paradigm, a large language model is prompted to generate new mathematical transformations of OHLCV data, rather than infer from an existing library. The chapter explains the LangGraph-based architecture, composed of eight domain-specific nodes and a provider-agnostic routing mechanism supporting five different LLM configurations. It also describes a structured feature generation implemented that uses round-robin difficulty levels and 22 mathematical creativity seeds. The chapter further expands on the four-layer cascading deduplication scheme that ensures feature uniqueness. It describes a three-tier memory architecture that enables cross-loop learning and establishes a standardized experimental protocol comprising 30 loops, each generating 50 features per LLM. The methodology is tested using XGBoost rolling-window training on 20 S&P 100 assets.

Chapter 6 compares and contrasts two experimental setups, synthesizing key findings from the analysis of 120 features across 10 metrics derived within a uniform XGBoost rolling-window validation framework. All five LLM configurations perform better than the traditional baseline, with ChatGPT 4o achieving the highest accuracy. It is worth noting that this yields a return more than five times that of the best-case model in Case 0. Moreover, distributional and categorical analyses of the output indicate a striking momentum dominance, unique creative signatures among the LLMs, and a prevalent long-side bias. Finally, the chapter reiterates the agent-based framework as an important breakthrough over classical feature engineering. It also defines some restrictions, such as the asset universe and transaction costs. Additionally, it summarizes paths ahead, the development of a dedicated database agent, a continuous deployment system, retrieval-augmented domain specialization, and multi-LLM debate protocols.

2. LITERATURE REVIEW

This chapter examines the literature supporting the research presented in this thesis. The review is structured around five main themes, which together form the theoretical basis of the proposed framework. The first two sections focus on feature engineering in financial time series and the shift toward automated methodologies, beginning with hand-crafted indicators developed by domain experts and moving to approaches based on genetic programming and AutoML. The third section discusses large language models in quantitative finance and their applications, covering core architectures, domain-specific adaptations, and their use in direct prediction tasks. The fourth section presents agent-based learning systems and addresses challenges in multi-agent coordination, as well as general architectures for memory and reflection, feedback-driven self-improvement, and specialized alpha mining agents. The fifth section covers advanced topics such as prompt engineering (e.g., structured prompting) and reasoning strategies for language models, including chain-of-thought prompting and the ReAct framework, which support modern agent-based pipelines. Finally, the chapter identifies key research gaps that motivate the development of the agent-based feature engineering framework introduced in the following chapters.

2.1 Feature Engineering in Financial Time Series

A huge amount of data is produced in financial markets that must be systematically transformed into exploitable signals. A fundamental challenge in quantitative finance is transforming raw price and volume data into structured features. In classical literature on financial time series, feature engineering relies heavily on domain knowledge, with practitioners in finance creating indicators based on recognized market theories and their trading intuition (Murphy, 1999; Achelis, 2001). While traditional techniques are easily interpretable and well-grounded in financial theory, they fail to scale efficiently or adapt over market regimes.

Histories of quantitative investment systems have evolved through distinct paradigms. Guo et al. (2022) track this evolution from Quant 1.0, where small expert teams

deployed interpretable mathematical models, to Quant 2.0, with massive factor mining/large-scale machine learning and low manual control. In addition, with the shift to Quant 3.0, deep learning techniques were incorporated for more complete predictions. In which the authors introduce Quant 4.0 as a combination of Automated AI, Explainable AI, and Knowledge-Driven AI to overcome obstacles such as model opacity, overreliance on data, and the inability to generalize across diverse market regimes.

2.1.1 Technical indicators and price-based features

Feature engineering in finance heavily relies on technical analysis, such as using price, volume, and derived statistical measures to find possible trading signals. Most practitioners use different indicators, which can be categorized as follows: trend-following indicators, momentum oscillators, volatility indicators, and volume-based indicators. Kakushadze (2016) documented a factor set of 101 formulaic expressions, which has become one of the leading benchmarks in quantitative research. These are factors that apply and manipulate mathematical functions across open, high, low, close, and volume (OHLCV) data to ensure they represent different market dynamics.

Yang et al. (2020) have built Qlib, with an expression engine that allows practitioners to easily build novel factors using simple mathematical expressions. This makes a whole lot of sense, since it's an architecture that natively supports both built-in asset technical indicators and the automation of custom features. Simple, systematic feature engineering approaches can pay off in practice. Mobilizing heterogeneous data sources on top of a distributed file system, the platform itself showed significant improvements by millions of folds when contrasted with traditional database solutions, reinforcing the need for infrastructure in quantitative research at scale. Following this endeavor, Sun et al. (2023) released TradeMaster, a holistic reinforcement learning framework that not only integrates Alpha158 indicators from Qlib but also proposes the PRUDEX-Compass evaluation mechanism for systematic algorithm evaluation.

FinRL, introduced by Liu et al. (2021), is the first open-source deep reinforcement learning framework for quantitative finance. Its architecture consists of three main layers—Application, Agent, and Environment—that together form a complete pipeline from data preprocessing to backtesting. Notably, the framework supports multiple deep reinforcement learning algorithms, such as DQN, DDPG, PPO, and

SAC, as well as ensemble strategies. As a result, it achieves an annualized return of 52.61% with a Sharpe ratio of 2.81 on the Dow 30 index, significantly outperforming classical buy-and-hold benchmarks. This robust infrastructure, therefore, lays the foundation for automated trading systems, which have been further advanced by subsequent agent-based approaches.

2.2 Automated Feature Engineering

This appears to be a significant step forward for the field as it automates the feature engineering stage of machine learning pipelines. Mumuni and Mumuni (2024) conducted a comprehensive analysis and also presented an organized categorization of automated data processing techniques into three main types: pre-processing, augmentation, and feature engineering. Keyword-based recurrent neural networks outperform traditional keyword methods by at least 10%, and their model beats multiple state-of-the-art systems across common benchmark datasets.

2.2.1 Rule-based and transformation approaches

Automated feature engineering emerged from its early stages, when systematic rule-based strategies generated candidate features using mathematical operators. The expansion-reduction paradigm (Mumuni and Mumuni, 2024) describes the process of expanding features via random transformations and selecting a subset based on performance metrics. So this one illustrates a systematic approach that gives equal weight to exploration and exploitation. In addition, the deep feature synthesis algorithm proposed by Kanter and Veeramachaneni (2015) was transformative for automating feature engineering from relational databases, demonstrating that it is indeed possible to create many features automatically with both speed and performance.

2.2.2 Genetic programming for alpha factor discovery

Availability of evolutionary optimization approaches such as genetic programming have made this a dominant paradigm for automated generation of alpha factors, evolving mathematical expressions. Recent work by Zhang and colleagues (2020) presented AutoAlpha, which is a hierarchical genetic algorithm framework that was able to successfully discover 434 diverse and effective formulaic alpha factors

outperforming traditional genetic algorithms with twelve times speedup. Their study showed that successful high-depth alphas frequently contain effective low-depth root genes, which provides a theoretical basis for using hierarchical search strategies throughout the large expression space.

Yu et al. (2023a) proposed a reinforcement-learning-based framework to generate complementary alpha collections. AlphaGen went even further with this line of research by using Proximal Policy Optimization, which is more efficient because it builds upstream models that incorporate downstream model performance across all asset classes, rather than training separate points. It can be seen that substituting the independent factor generation with an algorithm that considers member diversity, such as an ensemble-aware optimization method, results in a significant increase in the Information Coefficient (IC) value calculated on A-share market data (e.g., CSI300 index IC = 0.0725, compared to 0.0404 for XGBoost-based methods).

Shi et al. (2025) proposed the AlphaForge framework, which merges neural network-based genetic programming with an attention-based time-varying alpha combination mechanism. Such method is designed to overcome the failure modes of exploration and exploitation on large-dimensional expression spaces. AlphaForge improves adaptability over time by allowing factors to change their temporal weightings dynamically. In empirical experiments on the Chinese stock markets, AlphaForge outperformed both traditional genetic programming approaches as well as alpha mining methods based on neural networks.

2.2.3 AutoML frameworks and neural architecture search

This has made many advanced feature engineering techniques far more accessible and opened up multi-model experimentation to AutoML frameworks. According to a survey by Mumuni and Mumuni (2024), Neural Feature Search techniques utilize reinforcement learning-based controllers to determine effective feature transformation methods. DIFER is an instance of differentiable automated feature engineering that embeds discrete features into continuous vector spaces via encoder-predictor-decoder architectures. This also highlights an important similarity between Neural Architecture Search and automated feature engineering methods.

2.3 Large Language Models in Quantitative Finance

The transformer architecture proposed by Vaswani et al. (2017) is a milestone in deep learning for NLP that introduces self-attention as an alternative to recurrence to better capture long-range dependencies. This key invention set up the advance of cutting-edge language models, for example, bidirectional designs exemplified by BERT and autoregressive models like GPT. Despite the self-attention architecture being widely applied across asset domains, its practical attractiveness for financial prediction, which illustrates fundamental temporal dependencies and multi-asset correlations, was conferred by its intrinsic flexibility. Such flexibility aligns well with the complex, multidimensional relationships in financial data.

The use of large language models in finance has grown dramatically, with domain-specific architectures demonstrating performance well above that of general-purpose ones. Lee et al. (2024) provide an extensive review of Financial Large Language Models (FinLLMs), ranging from domain-specific pre-trained language models such as FinBERT to instruction-tuned architectures such as BloombergGPT, FinMA, and FinGPT. The five independent training strategies covered in their study are continual pre-training, domain-specific pre-training, mixed-domain pre-training, adding prompt engineering with large models, and instruction fine-tuning through parameter-efficient methods.

Mixed-domain pre-trained models have shown strong results across six financial NLP tasks assessed against a common benchmark, achieving the best performance especially on sentiment analysis, text classification, and named entity recognition. On the other hand, architectures specializing in specific tasks still dominate in more complex reasoning tasks like question answering and stock movement prediction (Lee et al., 2024). BloombergGPT, which was trained on a 363 billion token financial corpus, demonstrated that performance at scale can be misleading for financial tasks: the type of training and quality of data used are more important than size alone.

The FinQA dataset, a benchmark for numerical reasoning over financial data, was introduced by Chen et al. (2021). This work highlights the challenges of hybrid tabular and textual understanding in financial settings, requiring models to execute multi-step mathematical calculations from heterogeneous sources. Domain-specific languages

built for reasoning with programs provide a framework for producing structured outputs in financial feature engineering applications.

New strategies, such as parameter-efficient adaptation, allow the deployment of large language models in resource-constrained financial environments. Low-Rank Adaptation (LoRA), proposed by Hu et al. (2021), showed that using 10,000 times fewer trainable parameters can still achieve results similar to full fine-tuning. This progress enables affordable customization of models for specialized financial tasks by mitigating the computational burden and privacy constraints within financial institutions.

In a more recent investigation, the capacity of large language models (LLMs) in predicting financials is directly looked into. Lopez-Lira and Tang (2023) showed that ChatGPT may be able to predict stock price movements based on news headlines, as sentiment scores generated by the model can have predictive power for next-day returns. The implications of these results are that LLMs hold within them an implicit understanding of finance, which can be mined for quantitative tasks beyond traditional natural language processing application domains.

2.4 Agent-Based Learning Systems

Autonomous agents powered by large language models have been an active area of research for carrying out more complicated tasks. Ding et al. (2024) surveyed approaches to LLM-powered trading agents, presenting a taxonomy dichotomizing architectures between LLM as Trader and LLM as Alpha Miner. They showed that agent-based systems achieved annualized returns of 15–30% above baseline methods in the backtesting phase, although with somewhat short backtesting durations (on average 1.3 years).

The same year, Wang et al. (2025) gave an overview of LLM-based autonomous agents and proposed a unified framework composed of four fundamental modules: the Profiling module, which specifies the agent's role; the Memory module, which manages both short-term and long-term memories; the Planning module for task decomposition and reasoning chains; and the Action module, which transforms plans into actual actions. Such an architectural pattern has been widely used in domains like social simulation, scientific discovery, and software engineering.

2.4.1 Multi-agent architectures and coordination

Multi-agent systems represent a further step from single-model deployment to cooperative intelligence. Tran et al. (2025) surveyed 166 references on LLM-based multi-agent collaboration mechanisms. They distinguish collaboration at five levels: actors, types (cooperation, competition, coopetition), structures (centralized, decentralized, hierarchical), strategies (rule-based, role-based, or model-based), and coordination protocols. The study shows that cooperation mechanisms, where agents align individual goals with group objectives, outperform competitive approaches in complex problem-solving configurations.

Li et al. (2023) proposed CAMEL, a framework for communicative agents based on role-playing activities. Their inception prompting method enables human-level control of agent behaviour through system prompts, after which agents interact and prompt each other without human guidance. In evaluations, solutions produced by CAMEL were compared to those of GPT-3.5-turbo in single-shot mode; autonomous multi-agent collaboration outperformed the single-shot method 76.3% of the time according to human evaluation.

AutoGen, a framework for multi-agent conversations over a wide range of applications, was introduced by Wu et al. (2023). The conversable agent abstraction and auto-reply mechanism decouple the control flow via decentralization, enabling programmer-extensibility around customizable combinations of human participants, LLMs, and tools. In experiments on mathematical problem-solving, success was achieved in 69.78% of cases, a significant improvement over GPT -4's baseline performance of 55.18%. Supporting a modular architecture for the development and testing of specialized agents, it provides the most generalizable framework to bring these advancements into practice in any given domain.

A novel hierarchical multi-agent workflow framework called HAWK, containing five layers — User, Workflow, Operator, Agent, and Resource — was put forward by Cheng et al. (2025), providing inter-layer standard interfaces as well. In their comparative analysis, they showed that AutoGen, LangGraph, and MetaGPT are not flexible enough to combine heterogeneous systems; that the scheduling policy within these frameworks is static instead of dynamic; and that state synchronization in existing frameworks is far from sufficient. The operator layer modules, which handle

task management, memory, and inference, serve as an architectural reference point on how to construct custom financial agents.

MetaGPT is a framework that integrates Standardized Operating Procedures (SOPs) derived from human organizational workflows into the multi-agent coordination process (Hong et al., 2024). It mitigates cascading hallucinations and incoherent reasoning using document-based communication, shared message pools via publish-subscribe mechanisms, and executable feedback loops. The framework designates specialized roles — Product Manager, Architect, Project Manager, Engineer, and QA Engineer — applying role-based decomposition to organize complex tasks. MetaGPT also achieved state-of-the-art performance on the HumanEval benchmarks and provides architectural patterns that may be directly applied to financial feature engineering pipelines.

2.4.2 Memory mechanisms and reflection

In an example, Park et al. (2023) propose Generative Agents consisting of memory stream, retrieval and reflection components that combine to enable realistic human behavior simulation. The memory stream retains a time-ordered log of agent observations in natural language, and the retrieval process pulls these memories using a weighted sum of recency, query relevance, and importance. Ablation studies established that all three components are necessary for agent believability, as the reflection component enables higher-order abstractions based on past experience. In this work, the architecture sets forth guidelines for learning iteratively about specific domains.

At a high level, memory mechanisms underpin agent learning and adaptation. FINMEM was introduced by Yu et al. (2023b) as a layered memory architecture guided in part by insights from cognitive psychology. The system comprises a working memory modelled on the levels-of-processing theory developed by Craik and Lockhart (1972), where long-term storage occurs at several different layers according to how relevant these memories are deemed by the system. The experimental results showed that FINMEM significantly outperformed deep reinforcement learning agents and other LLM-based systems across all five stock assets.

2.4.3 Feedback-driven learning and self-improvement

Self-improving agent systems can improve and better with each iteration, by using feedback from their environment. QuantAgent, a closed-loop framework for autonomous alpha mining developed by Wang et al. (2024a), has two layers: the inner reasoning loop is composed of the knowledge base, context buffer, writer and judge components that allow for repeated updates to the output; conversely, the outer feedback loop allows for validation of outputs by backtesting against historical market data. The framework is presented as a Markov Decision Process, with theoretical guarantees of provable efficiency via sublinear Bayesian regret.

Chain-of-Alpha (Cao et al., 2025) is then proposed to format a dual-chain for formulaic alpha mining. The Factor Generation Chain produces new seed factors through self-evolving features, and the Factor Optimization Chain runs an iterative process for refining these factors based on backtesting feedback. To ensure not only quality but also overall complementary qualities, a four-dimensional evaluation framework was created to analyze Strength (RankIC), Consistency (RankICIR), Efficiency (Turnover), and Diversity. A 13.24% annualized return and a 1.42 information ratio were achieved on the CSI 500 index, with experimental results beating classical factor libraries and tree-based LLM, respectively.

FinAgent, the first financial trading multimodal foundation agent, was proposed by Zhang et al. (2024). It utilizes a market intelligence module to synthesize numeric, textual, and visual data, and uses reflection mechanisms at two levels: low level for analysis of the relationships between market prices; high level for evaluation of decisions. It has a diversified retrieval strategy: short-, medium-, and long-term context is thoroughly leveraged. This architecture yielded 84% greater annualized risk-adjusted returns compared to standard reinforcement learning techniques.

Li et al. (2025) proposed Co-STEER, a framework for agent-based code generation in collaborative data science research. The framework combines DAG-based task scheduling with an evolving knowledge repository and filters out factors correlated above 0.99 through IC-based deduplication to keep the factor library diverse. As a hypothesis-driven factor generation method, Co-STEER offers a glimpse into how automated research workflows might be structured for quantitative finance.

2.4.4 Alpha mining and signal generation

An example of a targeted LLM agent use case is the Alpha Miner paradigm, which focuses on quantitative factor discovery. Alpha-GPT, proposed by Wang et al. (2023), defines a new way to perform interactive workflows with humans and AI tailored for alpha research. This system uses large language models (LLMs) as natural middlemen between quantitative researchers and algorithmic search methods to translate natural language trading ideas into formalized representations, while dynamically delivering interpretable feedback about the performance of those factors. By streamlining the traditional alpha design process, this method lowers the barrier for technically skilled personnel while maintaining control over the creative process in researcher hands.

Ding et al. (2024) survey described memory and reflection mechanisms that help ameliorate hallucination threats, encourage more sophisticated understanding of the environment. In the FinMem framework, memories are simple layers of abstraction that first convert raw inputs into memories; when a new observation occurs relevant memories are retrieved and used to inform reflections. This motivates for a cognitively inspired architecture that emphasizes structured information processing to achieve generalizability in financial reasoning tasks.

2.5 Prompt Engineering and Conditioning Strategies

Promoting outputs of LLMs on specialized tasks is heavily dependent upon the prompting strategies used. Wei et al. (2022) showed that chain-of-thought prompting invokes a multi-step reasoning capability in LLMs and achieves impressive results on mathematical and logical reasoning benchmarks. That is, it encourages models to write out the intermediary reasoning steps clearly, which is especially relevant for financial feature engineering where clear tracing shows logical interpretability.

2.5.1 Zero-shot and few-shot prompting

Zero-shot prompting utilises the knowledge that is part of the pretraining of a model without specific examples for any one task, with minimal explicit guidance, whereas few-shot methods offer actual demonstration instances to organize both output structure and reasoning procedures. Khattab et al. (2023) introduced DSPy, a declarative statistical programming framework that translates calls to language models into self-improving pipelines. The teleprompter mechanism enables automatic prompt

optimization through iterative bootstrap demonstrations, achieving further performance boosts of 25% or more over normal few-shot prompting approaches. This systematic approach in prompt engineering reduces the brittleness associated with manually crafted templates when deployed across different models and domains.

2.5.2 Chain-of-thought and structured reasoning

Chain of thought prompting enables models to decompose a complicated problem into a series of chained reasoning steps. While the use of self-evolving chains that build based on previous outputs has been previously demonstrated in a factored generation scenario (i.e., Chain-of-Alpha; Cao et al., 2025), its two-part structure effectively separates generation and optimization, allowing for parallelization of optimization chains at a combined computation cost O(Km) instead of the O(b^d) offered by tree-based exploration methods. This would particularly benefit alpha mining applications at a large scale.

2.5.3 ReAct: Reasoning and acting frameworks

Specifically, Yao et al. (2023) boldly pointed out that the traditional way of treating reasoning traces, i.e. as a process independent from task-specific actions, is not appropriate to go beyond imitation in reinforcement learning. They proposed the ReAct (Reasoning + Action) framework by unifying reasoning and actions into a single prompting paradigm. In contrast to conventional chain-of-thought methods that create reasoning separate from the environment, ReAct combines words and actions in a cycle by swapping between verbal reasoning steps and concrete interactions taken within an external environment such as searching for information, calling tools, and updating observations. It lets models dynamically change the reasoning they have been doing in response to the feedback from the external environment, which can cut down hallucination rates that isolated chains of reasoning are susceptible to. Across question answering, fact verification, and interactive multi-agent decision-making benchmarks, experimental evaluations exhibited clear improvements of ReAct over chain-of-thought and action-only baselines. Specifically for HotpotQA and ALFWorld tasks a boosted score was reached. The ReAct paradigm then served as a foundation for contemporary agentic system design, inspiring orchestration frameworks like LangGraph, AutoGen, and CrewAI that feature node-level orchestration where agents alternate between structured reasoning and tool-mediated action execution..

2.6 Security Considerations in Multi-Agent Systems

Security in multi-agent AI systems is a fundamental challenge for financial applications dealing with sensitive information. Louck et al. (2025) examined agent-to-agent communication protocols for vulnerabilities, discovering eight major security holes in the A2A protocol used by Google, including lack of control over token lifetimes and overly permissive access scopes. For instance, the DirectDataFlowController mechanism that they proposed managed to make the data leaking rate 0 percent in their experimental adversarial prompt injection tests while without this approach baseline attack success rates were greater than 60%. These outcomes highlight the necessity of incorporating security mechanisms at the protocol layer in agent-based feature engineering systems.

2.7 Research Gaps and Opportunities

With more exciting developments in individual research areas, there are still large gaps in their integration. The general mechanization of technical indicators from OHLCV data falling within adaptive agent-based frameworks that leverage elements of iterative learning has not been extensively examined (Ding et al., 2024). In addition, the wide range of features generated by large language models (LLMs) is limited not only in scope but also in theoretical grounding, due to the absence of structured evaluation frameworks that correlate these produced indicators to known technical analysis categories.

Research focused on multi-agent collaboration mechanisms specific to financial feature engineering remains limited. General-purpose frameworks like AutoGen, CAMEL or LangGraph are multi-agent systems (Wu et al., 2023; Li et al., 2023; Tran et al., 2025), but it is not yet known how these might be adapted to systematic technical indicator generation, where clear roles can be assigned for aspects such as data preparation, feature generation, evaluation, and feedback. This thesis addressed the major research challenge of designing agent-based pipelines that embed iterative feedback loops within structured feature-evaluation frameworks tailored to financial applications.

3. THEORETICAL BACKGROUND OF AGENTIC AI SYSTEMS

This chapter provides a comprehensive theoretical background to the foundation of Agentic AI, which is at the heart of this thesis. It first covers the basics of large language models and generative AI, then progresses through the definition and architecture of single-agent systems. It ends with a detailed breakdown of the various parts that make up multi-agent Agentic AI, including its architecture, forms of coordination, reasoning methods, orchestration tools to implement it in practice, applications to date, and known downsides. Following the explanation of these theoretical ingredients, the last subsection directly links them to the experimental setup in this thesis. All technical details, codes mentioned, and architectural patterns shown are from the original literature source assuming a general audience without background knowledge of agent-based AI.

3.1 Foundations: Large Language Models

The modern research on Agentic AI is more or less inseparable from the development of large language models (LLMs), which are basically what serves as a mental engine behind almost all current agent architectures. On this basis, a more detailed understanding of what LLMs are, how they work and their basic strengths and weaknesses is arguably a prerequisite for any sound analysis of agentic systems.

3.1.1 What are large language models?

A large language model is a statistical model that assumes probabilities over sequences of tokens, with a token generally being defined as either a word or portion of a word. The model samples from a distribution over the next token given some context consisting of previous tokens. When generalized to models with hundreds of billions of parameters and trained on trillions of tokens, this simple goal leads to a set of impressive capabilities that are not explicitly coded: translation, summarization, question answering, code generation, logical reasoning and even simple arithmetic (Brown et al., 2020).

The most widely used architecture for current large language models (LLMs) is the Transformer, developed by Vaswani et al. (2017). It is based on a self-attention mechanism which models dependencies between all positions of the input in a given layer. Unlike recurrent networks, which process inputs one at a time in order, the Transformer processes all of the context window in parallel, enabling more efficient large-scale training and allowing for better modeling of long-distance linguistic dependencies. One basic limitation on the performance of LLMs is that their context window, the maximum number of tokens a model can attend to at any point in time, determines how much previous context the model has access to during inference.

It takes two different phases of training. In pre-training, the model is deployed on large text corpora and optimised to predict masked or following tokens using self-supervised learning without any human-labeled data for guidance. During this stage the model is instilled with parametric representations of knowledge pertaining to the world. During the fine-tuning stage, the model is adapted to specific downstream tasks or behavioral specifications. Supervised Fine–Tuning (SFT) is done on the expert full instruction-following datasets, where we use limited yet carefully created data that instructs the model to generate human-like formatted questions. Reinforcement Learning from Human Feedback (RLHF) improves alignment with humans by training a reward model on relative human ratings, which is then used to optimize policy (Achiam et al, 2023). The result is a model that can hold coherent, contextually appropriate conversations and follow sophisticated directions, traits critical for agentic deployment.

3.1.2 Emergent capabilities and fundamental limitations

An important finding in LLM literature is that some abilities appear discontinuously in large models: they are non-existent below a certain model size or compute used for training (Brown et al., 2020). These capabilities include multi-step arithmetic, logical reasoning, in-context learning, the ability to generalize to new tasks given a few examples specified in the prompt, and chain-of-thought reasoning. The most impactful emergent property in the context of agent design is instruction-following: the model's ability to process a long, multi-step instruction and write an output that satisfies each component one at a time. Such capability is a necessary condition both to orchestrate tool usage and coordinate with co-agents.

There are some very impressive capabilities these LLMs can do; and still, they face essential shortcomings that serve as the foundation for the agentic paradigm. They are stateless: without explicit management of the context window, each call to inference has no representation of prior calls. They are knowledge-locked: the parametric details frozen into their architecture reflect only the training data up to a particular cut-off date and cannot be changed in real-time without re-training. Besides, they are subject to hallucinations; output that is within the congruence of the training distribution but factually incorrect becomes a significant problem when LLM outputs begin to be used for decisions affecting the real world (Bornet et al., 2025). In fact, these constraints are just the limitations that agent architectures seek to counteract.

3.2 Generative AI as Precursor to Agency

Even before AI Agents drew attention, the paradigm that dominated after LLMs were introduced to the public was generative AI, systems designed to generate original content (text, code, images, sounds) based on user prompts. Generative AI systems, represented by GPT-4 (Achiam et al., 2023), Gemini, and Claude, are powerful content generation engines, but they suffer from an essential design flaw that restricts their ability to perform autonomous tasks: they function reactively and statelessly.

The interaction model of a generative AI system is as strict as follows: Prompt - Model - Output. It simply waits for an input, runs it through a single forward pass and produces an output. It cannot take actions on its own, watch its environment, call external services, retain memories of previous interactions, or pursue goals across a sequence of exchanges without being re-prompted for that goal explicitly. After a user completes their session, every context is wiped clean. To summarize, this stateless, reactive architecture is sufficient for content-creation tasks like writing an email or explaining a concept, but it utterly fails as soon as the task needs to maintain persistent interactions with the external world, aggregate intermediate results, or adjust its decisions over time.

Generative AI is in fact most accurately described as the cognitive substrate on top of which we can construct agentic systems. The LLM provides language understanding, reasoning, and code generation capabilities that enable agents to perform their function. The agent architecture provides all the underlying plumbing (memory, tools, feedback loops and orchestration) that allows one to convert a reactive content

generator into an autonomous executor of tasks. Table 3.1 illustrates this various paradigms graphically by aligning the three paradigms for fundamental operational dimensions based on hierarchical reality relation.

Table 3.1: Comparative taxonomy of Generative AI, AI Agents, and Agentic AI across key operational dimensions. Adapted from Acharya et al. (2025) and Bornet et al. (2025)

	Generative AI	**AI Agent**	**Agentic AI**
Primary Goal	Create content on demand	Execute a bounded task via tools	Orchestrate complex multi-step workflow
Autonomy	None – prompt-driven	Medium – tool-assisted execution	High – self-managing across agents
Memory	Stateless; no persistence	Optional short-term buffer	Persistent episodic + semantic
Planning	None	Single-step task planning	Long-horizon, multi-step, adaptive
Collaboration	None	Isolated – no inter-agent comm.	Multi-agent coordination & negotiation
Architecture	Single LLM + prompt	LLM + tool chain	Orchestrator + specialized sub-agents
Learning	Static pretrained	Heuristic / feedback loop	Meta-learning across task cycles
Error Handling	None	Retry on tool failure	Self-correction + inter-agent validation
Examples	GPT-4, DALL·E 3, Gemini	AutoGPT, LangChain Agent	AutoGen, CrewAI, MetaGPT, ChatDev

3.3 AI Agents: Definition, Properties, and Architecture

The idea of an intelligent agency existed before the arrival of large language models. According to Wooldridge and Jennings (1995, p. 116), an agent is defined as follows: "an autonomous computer system situated within some environment capable of independent action in order to meet its design objectives." This communication-oriented definition implies three key aspects: (1) it is situated (the agent exists within and observes its surroundings), (2) it is autonomous, initiating actions without a direct

human command, and (3) it is goal-directed, meaning that its behavior seeks to achieve particular goals. The appearance of large language models itself did not change this definition, but it surely increased the number of systems that had a chance to meet those criteria.

3.3.1 Distinguishing characteristics of AI agents

Three key characteristics distinguish modern LLM-powered AI agents from classical rule-based systems and also from purely generative AI models (Acharya et al., 2025; Biswas & Talukdar, 2025):

Autonomy: Once an AI agent is deployed, it can work with less or even no human involvement. After being initialized with a specific objective and appropriate permissions, the agent detects inputs, uses them to reason and performs actions in a self-sustained iterative loop. Autonomy is limited (to some extent) by a set of rules and permissions, but within those rules the agent does not need approval to take any particular action. That quality allows agents to scale out to problems that would be infeasible with human-in-the-loop workflows.

Task-Specificity: AI agents are optimized for specific, narrowly defined tasks. They are specialized for easily repeatable tasks in a narrow domain like filtering emails, querying a database, or coordinating calendars. This investment in a narrow focus and specialization provides a better mechanism for efficiency, interpretability and accuracy to automate functions where far-reaching generalization would be wasteful or ineffective.

Reactivity and Adaptation: AI agents add primitive ways to deal with live inputs, so they can respond to the input in real-time (such as a user request or external APIs, or thanks to the change of state in software environments). Some systems include basic learning options via feedback loops, heuristics, or dynamically updated context buffers that can improve behavior in discrete steps (think of personalized recommendations or the ability to keep a conversation flowing).

3.3.2 Agent architectures

The literature identifies three principal architectural paradigms for intelligent agents (Biswas & Talukdar, 2025):

Deliberative Architecture: Deliberative architectures use explicit world models and plan action sequences in advance using symbolic reasoning. The promise of these architectures regarding problems that require complex inference, elaborate reasoning, and sophisticated planning is compelling. They use methods such as rule-based reasoning, constraint satisfaction, and heuristic search to efficiently navigate complex problem domains. The sense–plan–act (SPA) cycle consists of perceiving the environment, updating an internal model of the environment, reasoning over that model, and subsequently acting.

Reactive Architecture: In reactive architectures, the stimulus-response mappings are direct and do not depend on an explicit representation of the environment. It highlights quick responses to environmental changes as it works best in dynamic environments, an environment in which immediate action is required. However, these architectures do not have the ability to plan or reason about future consequences while they consistently fail on some tasks which require abstract reasoning and generalization.

Hybrid Architecture: A hybrid architecture is a layered architecture that combines deliberative and reactive components. The reactive layer is confined to short-term, ground-level reactions to environmental triggers, and the deliberative layer presides over long-range thinking, planning and choice. This integration allows for both complex reasoning and real-time response, making hybrid architectures particularly viable in domains that require both capabilities at the same time.

3.3.3 Architectural components of an AI agent

The internal architecture of an AI Agent can be decomposed into four primary subsystems that form a closed-loop operational cycle (Acharya et al., 2025; Bornet et al., 2025):

Perception Module: The perception module takes environmental inputs and represents them in a format compatible with the reasoning of the agent. This module handles tasks such as prompt templating, context wrapping, or document chunking for LLM-powered agents. It also encodes images or audio signals in multimodal agents. An agent's ability to correctly perceive the context of its task is directly dependent on how well the perception module works.

Knowledge Representation and Reasoning (KRR) Module: The KRR module is instantiated directly by the LLM at the core of the agent. This module uses the

statistical and semantic knowledge of the model to parse input, perform multi-step inferences, evaluate plans for executing actions, and select an appropriate next action. This functionality is enabled using structured prompting in contemporary LLM-powered agents — for instance, using Chain-of-Thought (CoT) signalling to nudge the model into responding with coherent, logistically reasoned outputs that are easier for humans to read (Wei et al., 2022), few-shot examples, and structured output schemas.

Action Selection and Execution Module: The Action Selector is the module that translates KRR decisions into concrete actions to be taken in the environment. Among these actions are sending a message to another agent, invoking an external API endpoint, making database queries (or writes), writing to a file, executing some code block, and providing a structured reply to facilitate the following process. The action library that the agent has access to defines the range of actions it can actually perform.

Learning and Adaptation Module: Common learning mechanisms for single AI agents are limited to individual heuristics for changing one parameter, remembering the context of past actions when responding to a user state, or scoring various approaches that lead to higher ordering of tool selection in a ladder-like manner. Some systems include a very simple form of memory buffer, which can save the last few user inputs or agent outputs, in order to provide basic personalization. Unlike agentic AI systems, single agents typically have no meta-learning abilities towards various types of tasks, cross-session memory architectures, or inter-agent knowledge transfer mechanisms.

3.4 The Conceptual Leap: From AI Agents to Agentic AI

Even though an improvement over the passive generative systems, AI agents are still architecturally limited which inevitably causes more difficulty when task complexity increases. Bornet et al. (2025) pinpoint the main limitation: just one agent in a limited context window, with a static tool library optimized for a singular task type, cannot consistently decompose and execute objectives that require long-term reasoning across divergent domains or derive value from executing independent subtasks in parallel. These constraints have led to the creation of Agentic AI, a paradigm in which agents specialized in their capabilities interact through structured communication, shared memory and co-ordinated orchestration to achieve high-level complex goals.

3.4.1 Defining agentic AI

Acharya et al. (2025) define what they called Agentic AI as systems built from multiple individual AI agents that have been designed for collaboration, coordination, and achieving complex goals using procedures such as goal decomposition, dynamic role assignment between team members, communications between the agents involved in a task, and memory persistence. This spotlights four key elements that may define Agentic AI as distinct from a classic single-agent model.

Multiplicity: An agentic AI system consists of several agents, each with a specialized function. Those agents team up to finally achieve global goals not possible for one agent alone. Such a multiplicity enables parallel processing, role-based cognitive division of labour and emergence of system-level intelligence that exceeds the combined capacity of individual agents.

Goal Decomposition: General high-level goals are candidly broken down into sequences of tractable subtasks, with each targeting a specialist agent that corresponds most effectively to the task at hand. This decomposition is not static, rather, the system can change how it breaks down the task into steps based on what happens in between steps by observing intermediate results over time rather than following a pre-defined rigid workflow.

Persistent Memory and Shared Context: While in single-agent systems context is lost between sessions, Agentic AI sustains persistent memory stores that can be accessed by all agents. This embedding context enables agents to collaboratively construct upon each other's outputs, re-use previous computations, and enable continuity across multi-session workflows.

Coordinated Autonomy: Individual agents act at a more local level within their respective roles but are coordinated by an orchestration layer that provides dependency management, conflict resolution, and task output routing between agents. Such a structure yields what is termed a distributed system; there's no single point of failure, and it offers coherence, as it collectively pursues a common goal.

3.5 Architecture of Agentic AI System

The architecture of a full Agentic AI system can be decomposed into five interdependent layers: the orchestration layer, the specialized agent ensemble, the

memory subsystem, the reasoning and planning framework, and the tool and environment interface.

3.5.1 The orchestration layer

The orchestration layer is by far the most unique architectural element of Agentic AI and is also what turns multiple independent agents into a single well-organized domain. The orchestrator (or meta-agent or controller) is itself an agent, powered by an LLM, but manages instead of being a domain expert. It takes the high-level goal, breaks it down into subtasks, assigns each task to a suitable specialized agent, tracks progress, makes sure agents do not block each other's work due to dependencies on individual parts, and finally resolves conflicts when agents return imperfect answers in forms that are conflicting.

From the literature, two main orchestration topologies are identified: centralized and decentralized. In centralized orchestration, one meta-agent remembers the full task state and sends commands to child agents. While this structure results in closely coordinated and integrated planning, it also embeds a single point of failure. On the other hand, decentralized orchestration follows peer-to-peer communication protocols to make agents decide what task to play and a global authority is never exercised from a single central controller over other individual agents. While this increases robustness to failure of any given agent, design and debugging are more complicated because coordination emerges in less predictable ways (Acharya et al., 2025).

3.5.2 Specialized agent ensembles

Instead of putting all cognitive processing into a single agent, Agentic AI systems distribute reasoning across a set of specially designed agents. Every agent in this ensemble has been designed and prompted to be a specialist at one of the following tasks: data retrieval, extraction, preprocessing, generating candidate solutions, validating outputs, evaluating performance, and so on. This split in labor provides the following architectural benefits:

Cognitive focus for the first: a prompt and tool set of a specialized agent is closely matched to its intended use case, which enables it to deliver a higher quality output in that area than a generalist agent trying to juggle everything at once. Parallelism: independent subtasks can be run in parallel by separate agents; running a single agent

involving multiple subtasks adds significant latency to the total workflow. Third, fault isolation: failure in one specialized agent doesn't take down the system, as long as the orchestrator detects the failure and either retries the agent or delegates to a backup. Fourth, interpretability: each agent has a clear purpose, operates on structured inputs and outputs therefore their rationale can be analyzed separately from the system itself thus guaranteeing that the entire system is more debuggable than its counterpart monolithic architecture.

3.6 LangGraph and StateGraph: Technical Architecture

LangGraph represents, to the best of current knowledge, the most advanced framework for organizing agents by introducing a graph-based approach to orchestrating long-form (multiple sequential steps) workflows. At its core, LangGraph enables the representation of agent workflows as directed graphs where nodes represent computation units and edges describe how control or data moves from one compute unit to another (Jay, 2024; Biswas & Talukdar, 2025). The important generalization that is presented based on LangChain is the concept of directed state graph, a mathematical construct in which nodes represent agents or processing stages and directed edges represent control-flow logic.

3.6.1 StateGraph fundamentals

At the core of LangGraph-based applications is a StateGraph. In addition, it provides a declarative framework for describing the structure and behavior of agentic workflows using a state machine model. Each node interacts with a shared state object that is passed between nodes to keep data in sync throughout the execution pipeline. A StateGraph is composed of a few key elements:

Nodes: They are the basic computation units in the graph, called nodes. Every node represents an operation (such as invocation of an LLM, execution of a tool, or running the data through a process). Nodes are given the current state input and return an updated state output. The node function signature usually takes in a state dictionary and returns how to change that state.

Edges: They connect nodes and specify the execution order. LangGraph has two edge types: unconditional edges, which always move to a target node, and conditional

edges, which evaluate a routing function to determine the next node from the current one.

State Schema: The state schema defines the structure of data that flows through the graph. Using typed state classes (such as MessagesState for conversational agents), developers can ensure type safety and clear documentation of the information available at each step of the workflow.

3.6.2 Node communication and message passing

In LangGraph, nodes communicate through a shared state mechanism rather than direct function calls. This design pattern provides several advantages for agentic systems:

Immutable State Transitions: Each node receives a snapshot of the current state and returns only the changes to be applied. The framework handles merging these changes with the existing state, ensuring that state transitions are atomic and traceable.

Message Accumulation: For conversational agents, the MessagesState class automatically accumulates messages. Each interaction adds to a growing list of messages, maintaining the full conversation history without requiring explicit management by individual nodes.

Conditional Routing: The add_conditional_edges method allows nodes to dynamically determine the next step based on the current state. This enables sophisticated control-flow patterns, such as loops, branches, and early termination, based on runtime conditions.

3.6.3 StateGraph implementation patterns

Conditional Routing: The add_conditional_edges method allows nodes to dynamically determine the next step based on the current state. This enables sophisticated control-flow patterns, such as loops, branches, and early termination, to be implemented based on runtime conditions.

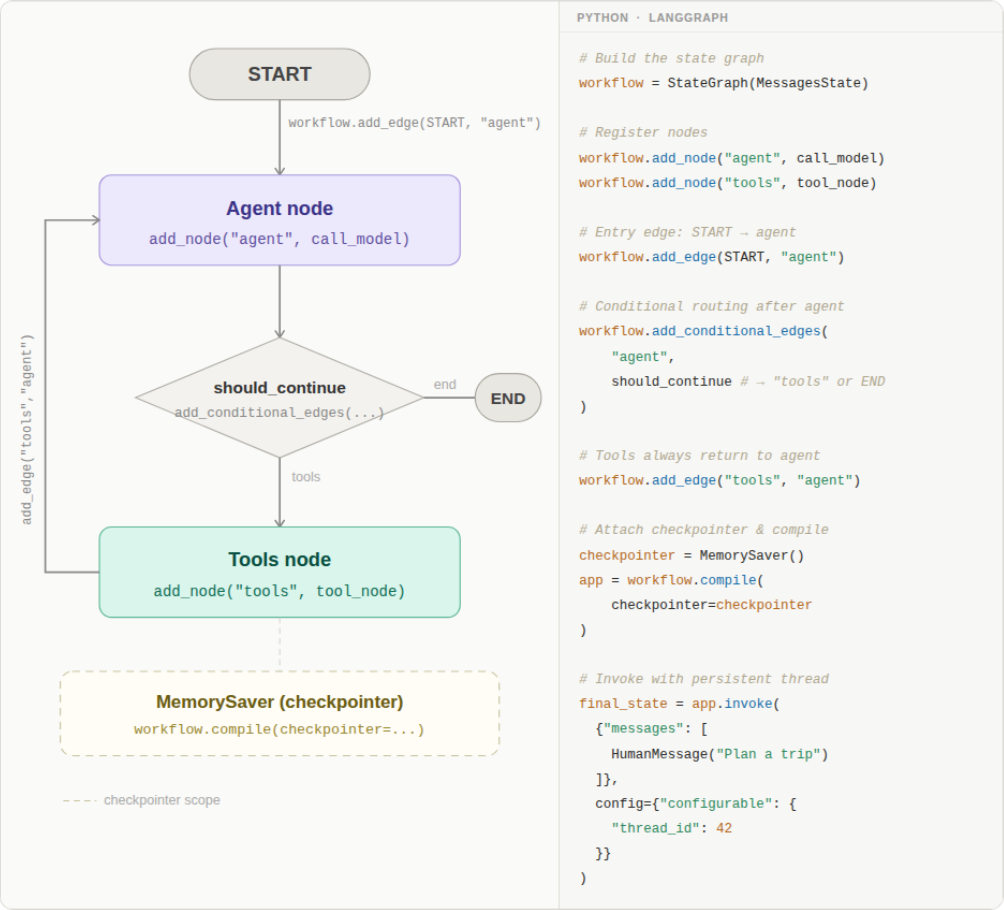

Figure 3.1: LangGraph Agent-Tools Workflow

3.6.4 Checkpointing and persistence

LangGraph already has native storage for state persistence using its checkpointing mechanism. Here are some of the critical utilities that this provides: in-memory conversation persistence across sessions so users can navigate back into conversations seamlessly, continuation and failure recovery so workflows can be restarted from the last successful execution checkpoint instead of from the start, maintaining audit trails so causes behind state transitions can be debugged (and necessary compliance measures maintained), support for human-in-the-loop workflows where execution can be stopped at any point while waiting on a human to evaluate the result and then continuing with the same context values.

3.6.5 Comparison of agent orchestration frameworks

Table 3.2 compares LangGraph with other popular agent orchestration frameworks, highlighting key differences in their approaches to state management, tool integration, and flow control (Biswas & Talukdar, 2025).

Table 3.2. Comparison of Agent Orchestration Frameworks

Aspect	LangGraph	CrewAI	AutoGen
State Management	Explicit state management via StateGraph	Through agent instances and task context	Through group chat message history
Tool Integration	Managed through dedicated tool node	Decorator-based tool definition with direct agent association	Associates tools directly with specific agents
Flow Control	Graph-based workflow with conditional edges	Hierarchical task decomposition or sequential flow	Round-robin turn-taking between agents

3.7 Memory architecture in agentic systems

Memory architecture represents a key distinction that sets agentic systems apart from stateless generative AI. Strong memory mechanisms enable agents to retain context across interactions, generalize from prior use cases and respond in a user-specific fashion. Relevant to agentic frameworks, the literature can be divided into a few different types of memory (see Table 3.3).

Table 3.3. Comparison of primary reasoning frameworks used in Agentic AI

Memory Type	Description	Role in Agentic System
Sensory / In-Context	The live content of the LLM context window is active for the duration of a single inference call	Immediate task instructions, tool outputs, and inter-agent messages received in the current cycle
Short-Term / Working	Temporary storage maintained within a session; cleared on session end	Agent scratchpad for intermediate reasoning steps and partial computations
Episodic	Structured log of past agent actions, observations, and outcomes across sessions	Enables agents to recall prior decisions, avoid repeating errors, and build on past work
Semantic	Persistent store of factual, domain, or procedural knowledge	Provides agents with reusable domain expertise; underpins knowledge retrieval chains
Parametric	Knowledge encoded in the LLM model weights during pretraining	Provides implicit world knowledge; cannot be dynamically updated without fine-tuning

3.7.1 Short-term memory (working memory)

The context of short-term memory (also known as working memory) is retained in current situations where it may still be relevant for ongoing actions. It includes the ongoing conversation thread, intermediate reasoning results, live search queries, and temporary preferences predicted via user text input. Its working memory is transient and unique to an input session, wiping clean after each iteration. Its execution typically follows the framework (Biswas & Talukdar, 2025):

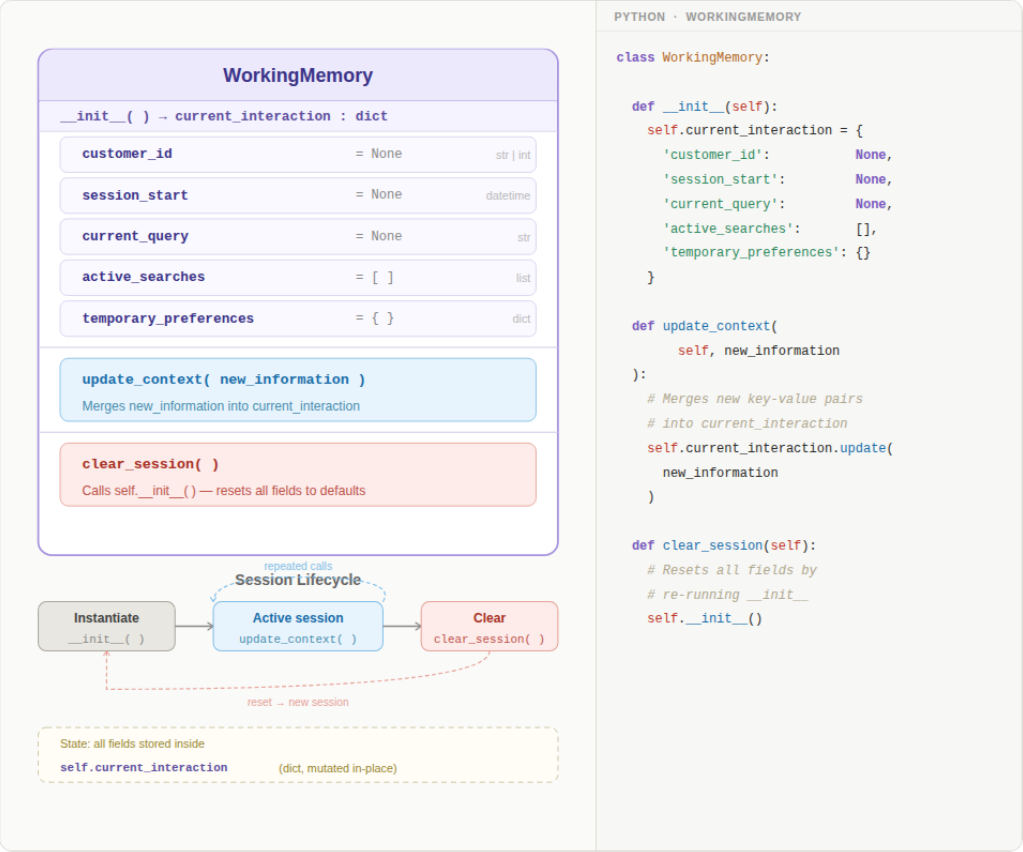

Figure 3.2: Working Memory Class – Structure and Session Lifecyclew

3.7.2 Long-term memory (knowledge base)

Long-term memory serves as the agent's lasting knowledge base, retaining information that is relevant and valuable over multiple interactions and sessions. This storage, unlike the short-term memory, is intended for information that must be kept and retrievable for long timeframes. It is also the very foundation of the knowledge, learned behaviors, and relations that an agent has built up. Usually, long-term memory consists of customer profiles and preferences, domain knowledge covering facts, rules,

and relationships between them; learned patterns such as strategies that work or fail often; and lessons on optimizing performance from operating experience.

3.7.3 Episodic memory (interaction history)

Episodic memory is an event-specific type of memory that encapsulates past interactions, their course and consistency. This type of memory enables an agent to benefit from past experiences, leveraging those prior interactions to help succeed in the future. Episodic memory (in contrast to general long-term memory) emphasizes when an event occurred and the context details surrounding that action, making it highly relevant for identifying patterns in behavior/outcomes.

For episodic memory there are many functions: pattern recognition, finding out what patterns work well or identifying common patterns of failure; contextual learning that assists with giving more contextually relevant responses; and personalization that serves increasingly personalized responses using historical data from past interactions (Biswas & Talukdar, 2025).

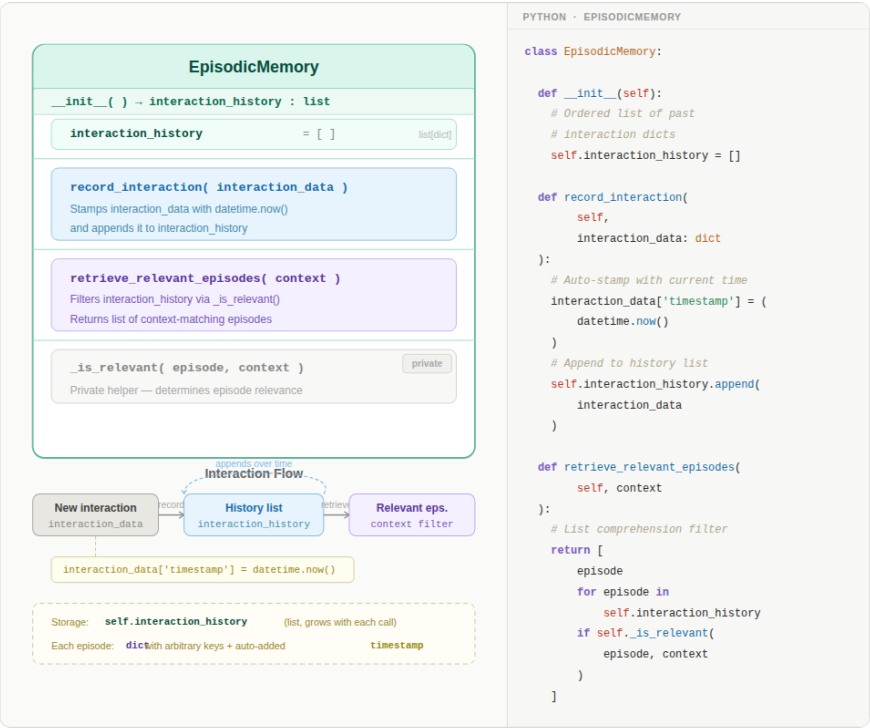

Figure 3.3: Episodic Memory Class – Structure and Retrival Mechanism

3.7.4 Context management and integration

Good context management facilitates the agent to maintain a correct sense of both current state and relevant historical information. The biggest challenge has to do with orchestrating information flow between the three memory systems, so that at decision time the right information is available. At the core of these systems is a single memory manager communicating across multiple modules: relevance filtering, which retrieves important historical information related to what the LLM appeared to ask for at the moment; memory consolidation, which moves crucial short-term data into long-term storage; and context window management, which handles the limited word capacity available within each LLM by splitting context windows amongst relevant ongoing tasks while keeping a reserve of recent work in case it needs to refer back.

3.8 Learning Paradigms in LLM-Powered Agents

LLM-powered agents are also critically dependent on their capabilities of gaining knowledge and using this competency to solve new tasks. Unlike the traditional machine learning systems where reliance on huge training datasets becomes a core limitation, modern language models exhibit aspects of flexibility in their learning mechanisms, they can operate with little or no examples for a specific paradigm. In this section, zero-shot learning, few-shot learning, in-context learning, and human-in-the-loop are discussed as the core agentic AI paradigms (Biswas & Talukdar, 2025).

3.8.1 Zero-shot learning

Zero-shot learning is a prime example of a basic representational form of emerging capabilities from large language models: the ability to do tasks without any input-specific training data. In this setup, the model relies only on knowledge it has learned in pre-training and the task description provided in the prompt to generate appropriate responses. The model extrapolates from its huge training corpus in order to understand and follow instructions it has never been trained on before (Brown et al., 2020).

The zero-shot capabilities of agentic systems are especially valuable in situations where labeled data is difficult to obtain or where rapid generalization to new domains is critical. For example, zero-shot learning allows an agent facing a new type of document or a new type of query to produce reasonably accurate outputs without additional training. However, the usefulness of zero-shot performance is limited by

what the model already knows and may be ineffective for niche or highly specialized tasks, such as technical documentation.

3.8.2 Few-shot learning

Few-shot learning extends zero-shot abilities by providing the model with a small number of input-output examples showing how to perform the intended task. Moreover, studies have shown that this approach is very practical as it requires only one to five examples for considerable performance improvements (Brown et al., 2020). These examples act as a sort of implicit instruction, allowing the model to identify patterns from those and apply them to new inputs.

In agentic architectures, few-shot learning allows customization almost instantaneously without having to retrain on their home datasets. A small, well-chosen set of examples can improve task performance significantly more than zero-shot methods when deploying an agent within a new enterprise environment. This method is particularly useful for tasks with repetitive structural operations, such as data extraction, formatting conversions, or domain classification. According to Biswas and Talukdar (2025), for many applications, few-shot learning addresses a core problem in deploying AI: the need for large labeled datasets makes it less useful in settings where collecting sufficient training data may be impractical.

3.8.3 In-context learning

ICL refers to the more general phenomenon where LLMs change their behavior based on the signals present in the current prompt, including instructions and examples as well as other contextual information. ICL acts only at inference time using the context window as a temporary scratchpad, completely avoiding any parameter updates that occur in conventional fine-tuning. This allows models to show specific task behavior without the need for any gradient updates (Brown et al., 2020).

This has meaningful implications for agentic systems. With ICL, agents are able to optimize their answers based on the preferences of the user, multi-turn domain-specific requirements or constraints that evolve over time, all specified in natural language within the prompt. This allows for very configurable agents to be created, where the prompts guide them rather than having to retrain the base model. However, ICL has its own limitations in that the context window is a predefined size and the learned

behavior is highly dependent on both prompt design and relevance/quality of other contextual information.

3.8.4 Human-in-the-loop (HITL) learning

Human-in-the-loop (HITL) learning, where human feedback directly influences the agent's actions in real time. The systems continuously incorporate human examples to adapt decision-making, correct mistakes, and validate outputs, rather than relying on pre-training or fixed datasets. Bornet et al. (2025) reiterate the need for this method at least during the early deployment of agentic AI, comparing it to an intelligent new employee that likewise must be trained and overseen.

There are several different forms of HITL mechanisms in agentic architectures:

Explicit Feedback: Users provide direct ratings, affirmations or corrections to agent outputs, which the system stores and uses to adjust future conduct.

Approval Gates: For some business-critical actions, human verification is required before execution, ensuring that decisions with a significant impact remain human-controlled.

Interactive Refinement: Users repeat the chat to guide how an agent produces a response, which results in step-by-step finalization.

Human Supervision: Human supervisors review ongoing agent activity, stepping in where appropriate to deliver corrective signals that influence the behavior of agents down the line.

This has serious implications for narrow AI (or generative AI) as it can have grave consequences when a single mistake is made by an autonomous agent in fields like healthcare, finance and legal consultation. Having humans involved at critical junctures allows organizations to enjoy the benefits of these agents while maintaining accountability and control.

3.9 Feedback Mechanisms in Agentic Systems

It is through feedback mechanisms that the adaptive engine of intelligent agentic systems can begin to close the loop between action and evaluation — evaluating some past actions against a future state and adjusting accordingly. Unlike static AI systems which operate within the confines of fixed behavioral schemas (i.e., instruction

manuals), feedback-enabled agents learn how to do a task through experience on the job, enabling them to become more effective with time. Bornet et al. (2025) state that feedback loops are the basis for adaptive systems; they allow AI agents to refine how they make decisions, learn from mistakes and gradually achieve more of their objectives.

3.9.1 Performance feedback

Performance feedback, on the other hand, contains mechanisms that measure agent outputs against quantitative measures of success. These metrics generally include task completion rates, response accuracy, processing time, resource utilization, and resultant business outcomes. Through some defined indicators, the system tracks financial and operational impacts of its actions, producing a steady stream of performance signals that drive behavioral changes.

In practical implementations, performance feedback operates through a structured cycle:

Observation: The agent gathers data from interactions, including user responses, task success rates, and implicit behavioral cues.

Evaluation: The system assigns reward scores—positive or negative—based on outcome assessment against defined objectives.

Adjustment: The agent refines future responses by updating its decision-making processes, prompt templates, or workflow configurations.

Deployment: The improved agent continues operation, generating new performance data that feeds subsequent learning cycles.

Performance feedback shows particularly high effectiveness in quantifiable domains, such as quality control in manufacturing. Bornet et al. (2025) report a 32% gain in accuracy and a 45% reduction in inspection time through feedback-driven optimization of agents.

3.9.2 Validation feedback

Validation feedback is about validating the correctness and appropriateness of agent outputs before executing them or showing them to users. This is especially critical

given that the large language models (LLMs) that underlie agentic systems are prone to hallucinating. There are various channels to gather validation feedback:

Explicit User Validation: Agents solicit direct confirmation from users regarding output quality, asking questions such as "Did this answer help?" or "Is this information correct?" Negative responses trigger logging and behavioral adjustment.

Automated Verification: Computational checks validate outputs against known constraints, schemas, or reference data. For example, a data extraction agent might verify that extracted values conform to expected formats and ranges.

Cross-Agent Verification: In multi-agent systems, agents assess their peers' outputs, establishing robust validation methods to detect potential hallucinations or errors (Bornet et al., 2025).

Validation feedback creates a quality assurance layer that catches errors before they propagate through the system. When validation fails, the agent can either retry with modified approaches or escalate to human oversight, depending on the task's criticality.

3.9.3 Category-aware feedback

Category-aware feedback represents a specialized form of evaluation that considers the semantic and functional categories of agent outputs. Rather than treating all outputs uniformly, this mechanism applies different evaluation criteria based on output type, enabling more nuanced assessment and targeted improvement.

In the context of feature engineering, the focus of this thesis, category-aware feedback proves particularly valuable. Generated features may belong to different functional categories (trend indicators, momentum oscillators, volatility measures, volume metrics), each with distinct quality criteria and evaluation standards. A momentum indicator should exhibit statistical properties different from those of a volatility measure, and category-aware feedback ensures that the evaluation reflects these domain-specific requirements.

The implementation of category-aware feedback typically involves:

Output Classification: Each agent output is tagged with its functional category based on content analysis or explicit metadata.

Category-Specific Metrics: Evaluation criteria are tailored to each category, applying appropriate statistical tests, validity checks, or domain constraints.

Targeted Adjustment: Feedback signals are used to improve category-specific generation capabilities, allowing the agent to develop specialized expertise across multiple output types.

This approach enables agents to maintain high quality across diverse output categories while avoiding the limitations of one-size-fits-all evaluation schemes.

3.10 Tool Integration And Function Calling

To remedy the limitations built into generative-only systems (for example, hallucination, knowledge cutoffs that can be static and protocols that limit interaction scopes), a new paradigm was proposed where AI agents were augmented with tools. Such systems introduce external tools, APIs and computational platforms into the reasoning of the agent in real time (i.e., enabling information access, code execution and interaction with dynamic data environments) (Biswas & Talukdar, 2025; Schick et al., 2023).

3.10.1 Tool invocation mechanisms

If an agent finds a need that cannot be satisfied by its internal knowledge, for example, querying the stock price of Tesla at this time, or to get some relevant news article, or to run a particular script, it generates structured function calls (or API requests). In general, these calls are in JSON format, SQL or a Python dictionary depending upon the needs of whatever service is on the other end and they go through an orchestration layer to execute.

Tool invocation follows a series of predefined stages: (1) Need Identification, the reasoning of the agent identifies that external information or an intervention is needed; (2) Tool Selection, all available tools and their capabilities are considered by the agent, which then selects the best-suited tool for this task; (3) Parameter Formatting, inputs need to be properly formatted to fit the input parameter required by tool invocation; (4) Execution, the orchestrator transfers request to final completion with selected tool, while waiting for a response from it; (5) Result Integration, tool output will be parsed and integrated into agent context as new facts in support of further reasoning.

3.10.2 Types of tools

Agentic systems employ various categories of tools to extend their capabilities:

Information Retrieval Tools: Web search APIs, database query interfaces, and knowledge base retrieval systems that provide access to external information sources.

Computational Tools: Code interpreters, calculators, and data processing utilities that enable complex computations beyond the LLM's native capabilities.

Action Tools: APIs that enable real-world actions such as sending emails, creating calendar events, or executing financial transactions.

Integration Tools: Connectors to enterprise systems, third-party services, and specialized domain applications.

3.11 Reasoning and Planning in Agentic Systems

Reasoning and planning are the cognitive capabilities that enable agents to move from high-level goals to concrete executable actions through a structured sequence of inferential steps. Table 3.3 provides a comparative summary of the primary reasoning frameworks used in Agentic AI.

Table 3.3. Comparison of primary reasoning frameworks used in Agentic AI. Sources: Wei et al. (2022), Yao et al. (2023a), Yao et al. (2023b).

Dimension	Chain-of-Thought (CoT)	ReAct	Tree-of-Thoughts (ToT)
Core Idea	Elicit step-by-step verbal reasoning before answer	Interleave reasoning with external actions	Explore multiple reasoning branches simultaneously
Tool Use	None – internal reasoning only	Central – actions invoke external tools	Optional – used within branch evaluation
Feedback Loop	None	Yes – observation from environment updates state	Yes – branch scoring prunes search space
Multi-Agent Use	Individual agent reasoning	Standard pattern for tool-using agents	Complex planning requiring path evaluation

3.11.1 Reasoning and acting

The ReAct (Reasoning and Acting) framework combines reasoning via Chain-of-Thought prompting with action using a tool to show how tool-augmented agents are architected. In this model, large language models (LLMs) switch between internal cognition and communication with the outside world (Yao et al., 2023a). The ReAct pattern creates a feedback loop in which agents: (1) Reason, evaluate the current state and how to best follow through with an action, (2) Act, call a tool/API to enact this action, (3) Observe, interpret the feedback or results from the tool, and (4) Repeat, continue doing this until complete or until some exit condition has been reached.

The key innovation of ReAct is that reasoning and action-taking are not performed in stages, one after another: rather than reasoning in advance and then performing one action, the agent reasons step by step, acting per decision as it helps answer the question. This interleaving allows the agent to adapt in real-time. If a tool call produces surprising results, the next Thought step now has factual knowledge from that experience and updates its course of action as needed.

3.11.2 Chain-of-thought prompting

Chain-of-Thought (CoT) prompting was introduced by Wei et al. (2022), eliciting intermediate reasoning steps before a large language model (LLM) generates an answer. Where traditional prompting forces the model to jump straight from a complicated single question to its answer (which often fails in multi-step problems), CoT prompting emphasizes step-by-step deduction, providing a final response composed of understandable intermediate conclusions. CoT is particularly important for agent-based applications because it makes an agent's reasoning transparent and inspectable, enabling debugging and auditing.

3.11.3 Reflexion and self-critique

Reflexion (Shinn et al., This step is added after a task has been done, where the framework (2023) trains from completing or failing tasks with self-evaluation in verbal form [11]. Rather than going directly to the next task, the agent generates a natural language form of self-feedback on its performance: What went wrong? What needs to be done is different? This verbally self-criticised message is then saved in episodic memory and retrieved at the start of the next attempts when running a task, thus

allowing the agent to iteratively get better without any update in weights based on gradients.

Reflexion extends naturally to inter-agent critique in multi-agent systems: a dedicated verifier or evaluator agent evaluates the output of a generator agent and outputs an organized critique, specifying errors, suggestions for fixes, or certifying output quality, which is then passed as feedback to the generator. This adds a quality control layer that can reduce hallucination and error propagation in workflows that focus on generation.

3.11.4 Tree-of-thoughts planning

Tree-of-Thoughts (ToT), proposed by Yao et al. (2023b), introduces a search over multiple reasoning paths, extending the Chain-of-Thought (CoT) framework from a linear chain of reasoning steps to a tree-structured search. The agent creates several candidate next steps at each node of this tree, scores the potential for each candidate using a scoring function (which could also be implemented as an LLM inference), and prunes low-scoring branches before expanding more promising paths. This shifts LLM reasoning from a greedy, single-path approach to a more systematic search of the space of possible ways to reason.

3.12 Planning algorithms in agentic systems

Planning algorithms derive sequences of actions for an agent to take to achieve its goals from a given initial state. The choice of planning algorithm significantly impacts an agent's ability to handle complex, multi-step tasks effectively (Biswas & Talukdar, 2025).

3.12.1 Structured planning approaches

Graph-Based Planning: Graph-based planning represents a planning problem as a graph, where nodes correspond to possible states or configurations, and edges represent actions or transitions that can move between states. Two broad categories include graph search algorithms (such as depth-first search, breadth-first search, and Dijkstra's algorithm) that find paths from an initial to a goal state, and optimal path-finding algorithms that find the best path according to some criteria.

Hierarchical Task Network (HTN) Planning: Hierarchical planning approaches break down complex problems into a hierarchy of higher-level tasks and subtasks that achieve those objectives. This hierarchical decomposition allows reasoning about problems more abstractly and the reuse of solutions to common subproblems. Core advantages include computational efficiency through reusing subplan solutions, knowledge representation at multiple levels of abstraction, and increased scalability to handle complex problems.

Monte Carlo Tree Search (MCTS): The principles of MCTS consist in building an asymmetric search tree over many random simulations (playouts) starting from the current state. The results of these simulations guide the growth of the majority potential branches. This strategy is particularly useful in high uncertainty, large state space domains, where it still results in performance when compute is limited.

Constraint Satisfaction: In constraint satisfaction problems (CSPs), the problem is reformulated as a set of constraints that must be satisfied concurrently and then various forms of constraint propagation are applied to eliminate inconsistent options from the search space. CSPs offer a solid framework to handle several problems such as scheduling, resource allocations, puzzle solving and configuration tasks.

3.13 Multi-Agent Systems and Orchestration

Single-agent systems perform well for bounded, relatively simple tasks but many real-world problems require multiple specialized agents to coordinate in order to solve it. A Multi-agent system (MAS) consists of multiple intelligent agents that are capable of interaction, cooperation and sometimes competition in order to fulfil their individual or joint goals. Transitioning from single-agent to multi-agent architectures represents a major change in system architecture that enables distributed problem-solving, distribution of specific roles, and collaborative behaviors (Biswas & Talukdar 2025).

3.13.1 The coordinator-worker-delegator (CWD) model

The CWD model provides a structured approach to multi-agent collaboration, defining clear roles and responsibilities for different types of agents within the system. This hierarchical structure enables efficient task distribution while maintaining coherent overall coordination (Biswas & Talukdar, 2025).

Coordinator Agent: The coordinator plays the role of plan logics in multi-agent systems. It supervises the whole workflow to have a global perspective of the entire business initiative, makes executive decisions about resource allocation and task prioritization, maintains an up-to-date knowledge of what all agents are doing and how far they are from finishing each job, and reconciles conflicts when worker agents create incompatible suggestions.

Worker Agents: Worker agents act as specialized executors that are in turn responsible for parts of the whole job. Every agent has vast experience in its own specialization and focuses on its task areas. Some examples for better illustration are flight booking agents, hotel reservation agent, activity planning agents. data analysis agent etc.

Delegator Agent: The delegator serves as the critical link between strategy and execution. It excels at task prioritization and resource allocation, understands each worker agent's capabilities and current workload, and ensures optimal task distribution and workflow management.

3.13.2 Communication and coordination protocols

The emergent intelligence of an Agentic AI system is fundamentally a product of how its constituent agents communicate and coordinate. Three primary coordination patterns appear in the literature:

Shared Memory Coordination: Agents coordinate via shared memory in the sense that they all read and write to a mutual persistent memory store (db, key-value store, structured document collection). The agents themselves do not directly talk to one another; they put their outputs into the shared store and other agents read them when needed. That approach is very similar to the blackboard architecture from classical AI.

Message-Passing Coordination: Message-Passing Coordination: In message-passing coordination, agents send and receive structured messages explicitly through well-defined communication channels typically instantiated as message queues, publish-subscribe topics or point-to-point links. Every message contains a sender ID, recipient ID, type of message, and payload. An example of a conversational variant of this message-passing approach is AutoGen (Wu et al., 2023).

Role-Based Hierarchical Coordination: Role-Based Hierarchical Coordination, as shown by MetaGPT (Hong et al., 2024) and ChatDev (Qian et al., 2023), coordinates agents in a well-structured organizational hierarchical system based on human professional systems. For each role, there are one or more responsibilities, a determined scope of action and communication patterns that explain to whom the role reports, what reports it receives and what artefacts it consumes and produces.

3.13.3 Orchestration frameworks

Table 3.4 provides a comparative overview of major Agentic AI orchestration frameworks.

Table 3.4. Comparative overview of major Agentic AI orchestration frameworks. Sources: Wu et al. (2023), Hong et al. (2024), Qian et al. (2023), Pandya & Holia (2023).

Framework	Coordination Model	Memory	Specialization	Primary Use Case
AutoGen	Conversational multi-agent	Scratchpad + shared	Role-defined agents	Code generation, research pipelines
LangGraph	Directed state graph	Persistent graph state	Node-level specialization	Cyclic, conditional workflows
CrewAI	Role-based crew assignment	Shared task context	Crew roles with goals	Team-based task automation
MetaGPT	Corporate hierarchy	Shared workspace	Dept. role agents (CEO, Dev, QA)	Software development lifecycle
ChatDev	Sequential dialogue chains	Chat history	Functional dept. agents	End-to-end software projects

3.14 Sequential and Parallel Processing Patterns

Agentic systems employ various execution patterns to optimize performance and manage complex workflows. Understanding these patterns is essential for designing efficient multi-agent systems.

3.14.1 Sequential processing

Sequential processing processes tasks in a pre-defined order. This pattern is applicable if the tasks depend on each other, for example, when output from one task becomes an input of the next and in case debugging and traceability are needed. Since execution follows a straight line, it allows tracing back errors and better understanding how the processing is taking place in the system.

3.14.2 Parallel processing

Parallel processing enables multiple tasks to execute simultaneously, improving throughput and reducing total execution time. This pattern is effective when tasks are independent and do not share state, when time-to-completion is critical, and when computational resources are available to support concurrent execution.

3.14.3 Performance optimization

There are a number of techniques that improve the efficiency of agentic systems: caching, which stores results from expensive-to-compute operations so they do not pay for themselves multiple times; load balancing, where work is distributed across available resources to better utilize overall throughput; speculative execution, which executes operations likely to be required before their need is confirmed; and batching, where similar requests are combined in order to reduce overhead.

3.15 Reflection and Meta-Reasoning in Agent

Reflection and introspection are critical capabilities that enable intelligent agents to analyze their own reasoning processes, learn from experience, and continuously improve their performance. These mechanisms distinguish truly adaptive agents from systems that simply execute predefined workflows (Biswas & Talukdar, 2025).

Meta-Reasoning: Meta-reasoning includes processes that guide and control reasoning, allowing agents to introspect their own cognitive performance or action selection and modify them when required. In agentic systems, meta-reasoning enables self-evaluation, where agents evaluate their outputs and identify areas for improvement, strategy selection, which allows them to select among possible methods based on previous performance, and resource allocation, informing how much computation an agent should allocate between different components of a problem.

Self-Explanation: Self-explanation capabilities enable agents to articulate the reasoning behind their decisions, supporting transparency (users and developers can understand why agents take specific actions), debugging (explicit reasoning traces facilitate identification of errors and inconsistencies), and trust (clear explanations help build user confidence in agent recommendations).

Self-Modeling: Error detection is also discussed as internal models of the agent's capabilities and limitations, that is, capability assessment (knowing which tasks it can execute consistently and which ones it cannot), uncertainty quantification (recognizing when confidence in outputs needs to be diminished), and adaptive behavior (adjusting strategies based on some self-appraisal of performance).

3.16 Challenges and Limitations of Agentic AI

The promise of Agentic AI is accompanied by well-documented challenges that limit its reliability, scalability, and deployability in high-stakes environments. Table 3.5 summarizes the primary challenges and their mitigation strategies.

Table 3.5. Key challenges in Agentic AI systems. Based on Acharya et al. (2025) and Bornet et al. (2025).

Challenge	Description	Mitigation Strategies
Hallucination	LLMs generate plausible but factually incorrect outputs; propagates through multi-agent pipelines	Self-verification loops, cross-agent validation, ground-truth anchoring
Prompt Brittleness	Minor prompt variations cause large output divergences, reducing reproducibility	Programmatic prompt templates, few-shot examples, structured output schemas
Coordination Failure	Agents pursue misaligned sub-goals, create race conditions, or enter deadlocks	Centralized orchestrator, explicit dependency graphs, timeout mechanisms
Error Cascades	Faulty output from one agent corrupts all downstream agents' reasoning	Inter-agent output validation, checkpoint rollbacks, reflexion loops
Scalability	Each LLM call incurs latency and cost; exponential with agent count	Asynchronous execution, batched API calls, lightweight sub-agents

Explainability	Tracing the causal chain of decisions across distributed agent logs is non-trivial	Structured audit logs, agent-level reasoning traces, timeline visualizations
Causal Reasoning	Agents detect statistical correlations but cannot model cause-effect relationships	Integration of causal inference frameworks, Bayesian layers, STRIPS planning

3.16.1 Hallucination and inherited LLM limitations

Error detection is also discussed as internal models of the agent's capabilities and limitations, that is, capability assessment (knowing which tasks it can execute consistently and which ones it cannot), uncertainty quantification (recognizing when confidence in outputs needs to be diminished), and adaptive behavior (adjusting strategies based on some self-appraisal of performance).

3.16.2 Coordination failure and error cascades

Multi-agent coordination failure is concerned with the situation where individual agents take locally rational strategies, which in combination lead to collectively suboptimal results, a closely related phenomenon studied in game theory and mechanism design. Performance failures occur in practice in a few ways: agents generate duplicate or contradictory outputs; agents are stuck waiting indefinitely for inputs from others that have failed completely or crashed; semantic ambiguities within common communication protocols can lead to misinterpretations of output by other agents; and the orchestrator does not recognize that a subtask has failed, therefore continuing downstream execution based on erroneous intermediate results.

3.16.3 Emergent behavior and unpredictability

Among the most scientifically fascinating and practically significant properties of agentic AI systems is that system-level behaviors arise that are not explicitly coded in any given agent. The behavior of a multitude of autonomous agents with advanced LLM-based reasoning capabilities communicating over shared memory and message-passing channels can be very different from that of any single agent. This divergence is creative when agents discover new solution methods or self-destructive by trapping the system in positive feedback loops that become ever more divergent, or establishing coordination patterns that eschew relevant information from the common context.

3.17 Solution Strategies and Architectural Best Practices

The Agentic AI literature has converged on a set of architectural best practices that collectively address the challenges outlined in Section 3.16. These practices are not mutually exclusive; production-grade Agentic AI systems typically implement all of them.

The Agentic Loop with Self-Critique: The number one best practice is to have a set of iterative Reason-Act-Observe-Critique loops at every level of the agent hierarchy. For each task done by individual agents, the ReAct framework allows anchoring the reasoning on observing the environment directly; the Reflexion mechanism allows triggering a self-critique step between every task so that the episodic memory of the agent gets up-to-date with feedback gained.

Programmatic Prompt Engineering: Prompt engineering relies on a fragile manual process, which does not scale well and is difficult to reproduce reliably. By contrast, programmatic prompt engineering replaces hand-crafted prompts with template-based systems that procedurally assemble a prompt from fixed parts: system instructions, task context, memory inspiration and structured formats of the output and roles.

Structured Output Schemas: Unstructured natural-language outputs from one agent pose parsing challenges for downstream agents. Structured output schemas—JSON schemas, Pydantic models, or XML templates—address this by constraining the LLM's output to a predefined machine-readable format.

Role Isolation and Authority Boundaries: Each agent in the system should be assigned a clearly defined role with explicit authority boundaries: which actions it is permitted to take, which memory stores it is permitted to read and write, and which agents it is permitted to communicate with.

Deduplication and Diversity Enforcement: In generation-heavy agentic workflows, the natural tendency of LLMs to exploit statistical patterns in their training data produces systematically homogeneous outputs. Deduplication mechanisms, hash-based exact duplicate detection, embedding-space near-duplicate detection, correlation-based functional duplicate detection, and removing redundant outputs from the candidate pool.

3.18 Situating the Study Within the Agentic AI Framework

The above theoretical background is now applied in the context of automated feature engineering for financial market prediction. Multi-agent systems using the foundational StateGraph architecture can derive features through coordinated data manipulation and feature generation pipelines. Since the focus is primarily on small data bodies, it is more than reasonable to create specialized agents for self-verification, protocol execution, training, and evaluational tasks.

There are several reasons why the agentic approach is especially appropriate for this domain:

First, feature engineering is an inherently iterative process that benefits from a reasoning, action, observation cycle. This allows agents to suggest candidate features, score their performance and incrementally improve their strategies through feedback.

Second, the domain requires diverse capabilities such as data processing, mathematical operations, model training and performance evaluation that are well suited to specialized agents acting in a coordinated multi-agent system.

Third, the self-verification criteria are strongly aligned with recent reflection and meta-reasoning capabilities of agentic systems, allowing the framework to assess a feature or set of features for goodness and soundness prior to deployment.

Fourth, due to the complexity and size of this task (i.e., generating a diverse set of features across technical categories that are not redundant), multi-agent frameworks can orchestrate the exploration process via parallelization while maintaining synchronization through a shared state.

3.18.1 Agentic approach to feature engineering

The 8-node multi-agent pipeline developed in this thesis instantiates the core architectural principles of Agentic AI. Cognitive division of labor across specialized agents ensures that each component of the feature engineering task is handled by an agent whose prompt, tool access, and evaluation criteria are optimized for that specific component: data ingestion and preprocessing are handled by the Data Agent; creative feature ideation by the Feature Generator Agent; code correctness validation by the Self-Verification Agent; training pipeline management by the Feature Trainer Agent; and performance evaluation by the Feature Evaluator Agent. The Orchestrator Agent,

implemented through LangGraph's StateGraph, manages the coordination of these specialized functions across iterative generation cycles.

Permanent episodic memory, exemplified as a structured SQLite database layer that documents the complete history of all features generated (including code implementations, mathematical formulations, SHA-256 hash signatures, and real-world performance metrics), serves as the basis for a deduplication mechanism through successive generations. Just before every generation cycle, the previously generated feature signatures are requested from memory, and in case certain conceptual spaces have been explored, the system informs the Feature Generator Agent not to explore those specific areas again, given their unique fingerprints.

4. METHODOLOGY AND EXPERIMENTAL RESULTS

This chapter describes the experimental methodology used in the two case studies presented in this thesis. Case 0 is a standard feature engineering baseline that creates a pipeline of 431 technical indicators taken from tried and true quantitative finance libraries. These indicators are selected through a multi-step process and it results in a set of 40 features which is tested by a rolling-window XGBoost classification framework. The second case (Case 1) explores the agentic framework proposed in this study, where a multi-agent system based on large language models creates original alpha factors. These are then analysed under similar experimental conditions to the baseline. Uniformity in terms of the data configurations, asset universes, and performance metrics across both cases allows for comparing standard feature engineering techniques against the generative, agent-based approach head-to-head within controlled conditions.

4.1 Case 0: Traditional Feature Engineering (Baseline)

This chapter describes the experimental methodology used in the two case studies presented in this thesis. Case 0 is a standard feature engineering baseline that creates a pipeline of 431 technical indicators taken from tried and true quantitative finance libraries. These indicators are selected through a multi-step process and it results in a set of 40 features which is tested by a rolling-window XGBoost classification framework. The second case (Case 1) explores the agentic framework proposed in this study, where a multi-agent system based on large language models creates original alpha factors. These are then analysed under similar experimental conditions to the baseline. Uniformity in terms of the data configurations, asset universes, and performance metrics across both cases allows for comparing standard feature engineering techniques against the generative, agent-based approach head-to-head within controlled conditions.

The Case 0 pipeline has six main steps: (1) fetching data and calculating traditional features, (2) preprocessing data and creating output labels, (3) removing redundant

features based on correlation, (4) using XGBoost to rank feature importance and narrowing down to 40 features, (5) training with a rolling window and validating over 12 months, and (6) reporting and analyzing performance. Each step helps refine the features to ensure the process remains statistically sound and avoids look-ahead bias.

4.1.1 Data sources and feature set

The baseline feature set is composed of 431 technical indicators obtained from two main sources. Of these, 179 algorithmic features are derived from Microsoft's Qlib Alpha158 library, which is widely used in quantitative finance studies. In addition, 252 well-established indicators are taken from Achelis's Technical Analysis from A to Z (2001). For analytical clarity, the full set of features is grouped into four primary categories, namely Trend, Momentum, Volatility, and Volume.

Table 4.1. Feature distribution by category and source library.

Source	Trend	Momentum	Volatility	Volume	Total
Alpha158 (Qlib)	45	65	34	35	**179**
Achelis (2001)	71	59	81	41	**252**
COMBINED	**116**	**124**	**115**	**76**	**431**

The feature set is largely balanced across the Trend, Momentum, and Volatility indicators, whereas Volume-related features are underrepresented. This distribution reflects the long-standing focus of technical analysis on price behavior and momentum-based signals.

4.1.2 Asset universe and data configuration

The asset universe comprised 20 S&P 100 index stocks, selected based on their average daily trading volume over the one-year period ending January 30, 2026. High-volume stocks were picked because they naturally demonstrate greater intraday price dispersion and are less prone to data quality issues, such as missing or irregular observations, which can compromise the reliability of the training process. The final asset universe consists of NFLX, TSLA, NVDA, INTC, F, PFE, AVGO, AMD, AAPL, T, BAC, AMZN, META, GOOGL, ORCL, CMCSA, MSFT, VZ, GOOG, and FCX. Historical OHLCV data for all 20 stocks were sourced from Interactive Brokers, with data coverage beginning in January 2021.

All assets can be consolidated into one analytical dataset allowing evaluation of cross-asset features. While each asset is monitored individually for its performance during the training phase, correlation analysis and importance scoring are made on the entire dataset together. This method enables a bootstrapped detection of features that are consistent sources of information across multiple equity instruments, rather than limiting the analysis to asset-specific indicators.

4.1.3 Data preprocessing pipeline

The preprocessing pipeline consists of four sequential steps designed to ensure data quality, prevent information leakage, and establish a robust binary classification framework..

Step 1 – Feature Calculation: 431 technical indicators are calculated from raw OHLCV data by using the lookback periods as specified in their original definition. Lookback periods on these variables range from 1 to 60 trading days, including very short-term (lookbacks of 1–5 days), medium-term (10–30-day lookbacks), and longer intervals (60-day lookbacks).

Step 2 – Look-Ahead Bias Prevention: Since the prediction target variable is the daily open-to-close direction, all calculated indicators are shifted by one trading day. This temporal displacement ensures that predictions at market open rely solely on information from previous trading sessions, thereby preventing look-ahead bias. Formally:

$$Feature(t) = Indicator(t - 1) \tag{4.1}$$

Step 3 – Output Label Creation: The binary classification target is defined based on the intraday price movement direction. Specifically:

$$\begin{aligned} y = 1 \ if \ (Close - Open) > 0, \\ y = 0 \ if \ (Close - Open) < 0, \\ otherwise \ NAN \end{aligned} \tag{4.2}$$

For observations where Close = Open, we set the return as NaN, so that it is possible to be ambiguous in naming. If there are NaN labeled instances during this process, they will be dropped just before start training: so only the unambiguous directional samples will train the model, but we kept the total dataset to compute and select upstream features. The prediction output is subsequently converted into a directional

signal denoting whether the market price is expected to go up or down: +1 (Long), −1 (Short).

Step 4 – Anomaly detection and data cleaning: Any features with NaN values, arising from their initialisation due to the lookback, for example entries that have not acquired any information yet in their first entry, infinite values (as a result of division-by-zero operations), and constant features, i.e. variables for which all the observations took on the same value and thus exhibit zero variance, are removed from the dataset in a systematic manner. This process protects against numerical instability during model training and removes degenerate features that could inform the selection process.

4.1.4 Correlation-based feature elimination

The first step in feature selection is correlation analysis, which aims at eliminating redundant features that carry similar information. Analysis of inter-feature correlation is performed using the recent 12 months of training data, and a threshold of 0.80 is applied with respect to features in order to assess the risk level. The elimination process is not a single-pass pairwise comparison, nor does it operate in so-called iterations on small batches. The algorithm proceeds as follows:

(1) Rank features in an output-relevance fashion by computing the absolute Pearson correlation coefficient with a (potentially transformed and encoded) output variable.

(2) Sort all features by their output correlation in descending order and split them into an active analysis batch {top 120 features} and a waiting queue {remaining features}.

(3) Calculate the full pairwise inter-feature correlation matrix over all feature pairs within the active batch.

(4) For each feature pair with inter-feature correlation greater than or equal to the threshold ($\rho \geq 0.80$), keep the feature with higher output correlation and discard the other.

(5) Replace removed features in the active batch with as many features from the waiting queue, plus a buffer of extra features.

(6) Repeat Steps 3–5 until the depletion of the waiting queue; three more consolidation rounds are then performed on the remaining features.

This iterative batched parametric style is more principled than a single-pass elimination, as it permits features that pass early rounds to be reassessed against newly introduced candidates from the waiting queue. In this case, it usually compresses the feature space from 431 to about 280 features, removing more than 35% of the feature space as redundant.

4.1.5 XGBoost feature ımportance scoring

Following correlation-based elimination, the surviving features undergo gain-based importance scoring using XGBoost. The importance score quantifies the average improvement in classification accuracy (information gain) contributed by each feature across all tree branches where it appears. The iterative backward elimination process operates as follows:

An XGBoost classifier is trained on the pooled cross-asset dataset using a 12-month training window. The model configuration employs conservative hyperparameters to prevent overfitting: max_depth=3, n_estimators=100, and learning_rate=0.12. After training, gain-based importance scores are computed for all features. The feature with the lowest importance score is removed, and the process is repeated iteratively until the target count of 40 features is reached.

The importance score is calculated on a per-asset basis and then averaged across all 20 assets in the universe. This cross-asset averaging mitigates the risk of selecting features that are informative only for a narrow subset of equities, thereby promoting the identification of universally predictive indicators.

Table 4.2. Feature selection pipeline stages and dimensional reduction.

Stage	Input	Method	Output	Reduction
1. Calculation	Raw OHLCV	Alpha158+ Achelis	431	N/A
2. Preprocessing	431 features	NaN/Inf/Zero-var removal	~420	~2.5%
3. Correlation	~420 features	Pearson $\rho \geq 0.80$	~280	~33%
4. Importance	~280 features	XGBoost gain elimination	40	~86%

5. Training	40 features	12-month rolling validation	40	Evaluation

4.1.6 Rolling window training and validation

Each of the 40 surviving features is evaluated independently using a rolling-window training framework to assess its predictive capability across multiple temporal periods and assets. The training configuration employs the following parameters.

Table 4.3. XGBoost classifier hyperparameter configuration.

Parameter	**Value**
Training Window	6 months (rolling)
Validation Horizon	12 consecutive months
Asset Universe	20 S&P 100 constituents
Classifier	XGBClassifier
max_depth	3
n_estimators	100
learning_rate	0.12
min_child_weight	6
gamma	0.3
subsample	0.80
colsample_bytree	0.80
reg_alpha (L1)	0.1
reg_lambda (L2)	2.0
scale_pos_weight	Auto-calculated (min. 1.0)
Random Seed	42

Every feature generates 12 months × 20 assets = 240 unique train-predict cycles. For every cycle, a 6-month rolling window of historical data is utilized to train the model while out-of-sample validation is carried out for the following month. The class imbalance ratio (scale_pos_weight) is calculated automatically for each training window to neutralize any potential directional bias across the training labels.

4.1.7 Performance metrics

The evaluation framework relies on ten complementary metrics spanning three different levels of classification accuracy, directional precision and recall, as well as

return-based economic performance to provide a comprehensive assessment of model predictive quality [22]. All metrics are computed at the daily observation level and subsequently aggregated for the entire asset universe on a monthly basis in order to account for temporal differences in performance.

Directional Precision and Recall. Four directional metrics are computed to gauge the asymmetric performance of the model on long and short signals. Long Precision reflects the ratio of long signals that correctly predict upward movements while Long Recall describes the fraction of all actual upward movements which were identified by long signals. Likewise, Short Precision and Short Recall look at the respective rates for down moves. The four metrics are then averaged with their harmonic means - Long F1 Score and Short F1 Score. This provides strong metrics that help neutralize the impact of extremely high or low precision or recall:

Return-Based Metrics. Three metrics translate predictive accuracy to monetary terms. Model Return is the cumulative return on portfolio-weighted positions (absolute open-to-close returns), where the sign depends on prediction accuracy (+|r| for correct predictions and −|r| for incorrect ones). This gives a clear economic interpretation of the predictive power of the model. Long Return and Short Return decompose this aggregate measure into cumulative returns produced only from Long or Short position signals, so directionality of the model-level return can be examined.

Table 4.4. Performance metric definitions.

Metric	Definition
Model Accuracy	Proportion of correct directional predictions (excluding zero-return days)
Model Return	Cumulative return: +|r| for correct predictions, −|r| for incorrect predictions
Long Precision	Proportion of long signals that correctly identified upward movements
Long Recall	Proportion of actual upward movements captured by long signals
Short Precision	Proportion of short signals that correctly identified downward movements
Short Recall	Proportion of actual downward movements captured by short signals
Long Return	Cumulative return generated exclusively from long position signals
Short Return	Cumulative return generated exclusively from short position signals

Long F1 Score	Harmonic mean of Long Precision and Long Recall
Short F1 Score	Harmonic mean of Short Precision and Short Recall

4.2 Case 0: Experimental Results and Analysis

This section offers a complete breakdown of the 40 features selected via the Case 0 baseline pipeline. Feature selection reduced the dimensionality by 90.7%, selecting from the original 431 features down to only 40 features. A variety of angles on the findings are explored, including distribution across categories, accuracy and return performance, directional asymmetry analysis, and lookback period attributes.

4.2.1 Post-selection category distribution

Table 4.5. Post-selection category distribution compared to original feature space.

Category	Count	Selected %	Original %	Survival Rate
Volatility	17	42.5%	26.7%	14.8% (17/115)
Momentum	12	30.0%	28.8%	9.7% (12/124)
Trend	10	25.0%	26.9%	8.6% (10/116)
Volume	1	2.5%	17.6%	1.3% (1/76)
Total	**40**	**100%**	**100%**	**9.3% (40/431)**

After the selection pipeline, a significant change in category proportions occurs. Volatility features have an asymmetric retention pattern, retained in 42.5% of the final feature set from only 26.7% in the original set, yielding a total survival rate of 14.8%, the highest of all categories. This pattern suggests that volatility-derived indicators have unique predictive information, which is relatively uncorrelated with other feature groups, allowing them to survive the correlation filtering and importance scoring stages more consistently. By contrast, features from Volume experience the highest rate of attrition (17.6% to 2.5%), coalescing around one feature, OBV_D_1_Volume. This steep drop-off hints at a high level of redundancy among volume-based indicators, where one representative feature captures the predictive information across an entire category.

4.2.2 Overall performance summary

Table 4.6. Descriptive statistics for Case 0 selected features (12-month average across 20 assets.

Metric	Mean	Std Dev	Median	Min	Max
Model Accuracy	50.66%	0.57%	50.45%	49.91%	51.97%
Model Return	+0.357	0.428	+0.355	−0.622	+1.707
Long Precision	52.05%	0.65%	52.05%	50.53%	53.50%
Short Precision	49.15%	0.96%	48.99%	47.54%	51.14%
Long Recall	58.35%	2.85%	58.20%	53.19%	70.45%
Short Recall	42.59%	2.71%	43.10%	32.10%	47.87%
Long F1 Score	0.537	0.014	0.535	0.516	0.579
Short F1 Score	0.448	0.020	0.449	0.412	0.522
Long Return	+0.627	0.214	+0.626	+0.137	+1.302
Short Return	−0.270	0.214	−0.271	−0.759	+0.405

The average model accuracy of 50.66% represents a significant achievement compared to the random baseline (50%). While this increment may seem small on a standalone basis, it becomes more interesting in the context of these single-feature models aiming to predict daily direction only, possibly the hardest problem known in the literature where predicting daily equity returns is said to have a very high noise-to-signal ratio. Most importantly, this feature selection method selected a set of features each having an accuracy greater than 49.9%, meaning that the non-informative variables were excluded. Aside from that, the overall mean model return of +0.357 implies these features generate financially relevant predictions on average when used to make directional trading strategies.

4.2.3 Model accuracy distribution analysis

The distribution of accuracy shows that 92.5% (37 out of 40) of selected features achieve accuracy above or on par with the random baseline of 50%. There are only 3 features (7.5%) that drop just below the 50% line and still show over 49.9% accuracy. Importantly, 42.5% of features are above the accuracy threshold of 50.5%, and three features exceed 51.5%. The concentration of features in the 50.0–50.5% range (50.0%) reflects the fundamental difficulty of predicting daily direction with a single feature, where small outperformances over random chance can translate into large cumulative returns over long enough trading time horizons.

Table 4.7. Distribution of model accuracy across 40 selected features.

Accuracy Range	Count	Proportion	Cumulative	Interpretation
< 50.0%	3	7.5%	7.5%	Below random
50.0% – 50.5%	20	50.0%	57.5%	Marginal edge
50.5% – 51.0%	7	17.5%	75.0%	Moderate edge
51.0% – 51.5%	7	17.5%	92.5%	Notable edge
51.5% – 52.0%	3	7.5%	100%	Strong edge

4.2.4 Return distribution analysis

Most of them (85.0%; 34 out of 40) result in positive cumulative returns, showing that the feature selection pipeline is able to select features with real predictive value. Returns are right-skewed, with 40% of features giving returns above +0.5 and a significant outlier at +1.707 (VROC_D_60_Volume). Of the six features with negative returns, BB_WIDTH_D_30_Volatility is by far the largest (−0.622). This high fraction of positive returns (85%) indicates a significant improvement over the random selection of features and confirms the robustness of the multi-stage selection pipeline.

Table 4.8. Distribution of model returns across 40 selected features.

Return Range	Count	Proportion	Observation
Negative (≤ 0)	6	15.0%	Net loss-generating features
0 – 0.5	18	45.0%	Modest positive returns
0.5 – 1.0	15	37.5%	Meaningful positive returns
> 1.0	1	2.5%	VROC_D_60_Volume (+1.71)

4.2.5 Top-performing individual features

The Volume Rate of Change (VROC) indicator family dominates the top-performing features, occupying three of the top five positions with lookback periods of 10, 20, and 60 days. This finding is particularly noteworthy as it suggests that changes in trading volume carry significant predictive information for subsequent price direction—a result consistent with the volume-price relationship hypothesis in market microstructure theory. The single surviving Volume category feature, OBV (On-Balance Volume), achieves the sixth-highest accuracy and exhibits the most balanced Long-Short performance among all features, with a notably high Short F1 Score of 0.522.

All the top features list are from the Volume Rate of Change (VROC) indicator family, taking three out of five places with 10-, 20- and 60-day lookback periods. The implication is important: it means that differences in trading volume give you significant predictive power of future price movement, which is consistent with the hypothesis of a relationship between volume and price changes from microstructure theory. On-Balance Volume (OBV), the only remaining feature in the Volume category, has 6th highest accuracy and most balanced Long-Short performance of all features with stronger Short F1 Score than any other at 0.522.

Table 4.9. Top 10 features ranked by model accuracy (Long P. = Long Precision, Short P. = Short Precision).

Feature Name	Category	Accuracy	Return	Long P.	L-F1	S-F1
VROC_D_10_Volume	Momentum	51.97%	+0.798	53.50%	0.543	0.464
VSUMP_D_10_Volume	Volatility	51.86%	+0.858	53.27%	0.543	0.459
ATR_DIFF_D_20-60	Volatility	51.79%	+0.719	52.34%	0.549	0.449
VROC_D_20_Volume	Momentum	51.48%	+0.212	52.44%	0.545	0.448
VROC_D_60_Volume	Momentum	51.48%	+1.707	52.90%	0.548	0.455
OBV_D_1_Volume	Volume	51.40%	+0.717	52.88%	0.579	0.522
ADXR_D_60_Trend	Trend	51.22%	+0.492	52.90%	0.524	0.481
MOM_D_5_Momentum	Momentum	51.18%	+0.629	52.19%	0.543	0.441
MINUS_DI_D_5_Trend	Momentum	51.13%	+0.084	52.50%	0.549	0.440
IMIN_D_5_Momentum	Momentum	51.07%	+0.198	52.95%	0.574	0.412

4.2.6 Category-level performance comparison

The individual Volume feature (OBV_D_1_Volume) ranks as number one in performance for all three of the different metrics calculated: accuracy (51.40%), return (+0.717), Short Precision (50.72%), and Short F1 Score (0.522). However, this result is subject to prudent interpretation because it depends on a single feature.

For categories with more than one representation, Momentum features achieve the highest average returns (+0.511) and accuracy (50.78%), boosted slightly from continued strength in the VROC family. Volatility features, the most dense category with 17 features, yield medium returns (+0.312) for an accuracy of 50.60%. For multiple-feature categories, trend features offer the lowest average return of +0.211.

All categories maintain a Long-Short asymmetry in that Long Precision is approximately three percentage points higher than Short Precision.

Table 4.10. Category-level average performance metrics (Long P. = Long Precision, Short P. = Short Precision).

Category	N	Accuracy	Return	Long P.	Short P.	Long F1	Short F1
Momentum	12	50.78%	+0.511	52.17%	49.31%	0.540	0.440
Trend	10	50.52%	+0.211	51.92%	49.02%	0.533	0.455
Volatility	17	50.60%	+0.312	52.00%	49.01%	0.534	0.444
Volume	1	51.40%	+0.717	52.88%	50.72%	0.579	0.522

4.2.7 Long-short directional asymmetry

In the Case 0 results, the most significant finding is a very strong Long-Short asymmetry across all performance metrics. The directional imbalance is particularly apparent in Recall, where the Long Recalls (58.35%) exceed Short Recalls (42.59%) with a gap of 15.76 percentage points. This behavior indicates the models are able to identify most upward movements but miss a significant number of downward movements.

Table 4.11. Long versus Short directional performance comparison (pp = percentage points).

Metric	Long	Short	Difference (Δ)
Precision	52.05%	49.15%	+2.91 pp (Long advantage)
Recall	58.35%	42.59%	+15.76 pp (Long advantage)
F1 Score	0.537	0.448	+0.089 (Long advantage)
Return	+0.627	−0.270	+0.897 (Long advantage)

This asymmetry can be observed in the economic consequences, where long positions yield an average return of +0.627 against a net loss of −0.270 on short positions. This gap shows that the Short side is a net eroder of profitability in the model and suggests a direction for future improvement to achieve greater accuracy of Short predictions.

The Long bias is explained by the general bullish nature of the market during much of the 2023–2025 evaluation window. While scale_pos_weight automatically addresses class imbalance, the fundamental properties of market behaviour create an asymmetric prediction context where long trades are inherently more likely to lead to a positive

outcome. This has major implications for how Case 1 should be assessed, because the ability of the agentic feature engineering system to counterbalance this trend will be a key differentiator.

4.2.8 Summary of key findings

The Case 0 baseline is essential in adequately contrasting its performance to the agentic feature engineering approach as implemented in Case 1. This baseline illustrates that traditional technical indicators, after going through a careful and multi-stage feature selection from 431 features down to 40, are capable of providing small, yet consistent positive predictive performance in daily directional forecasting. An average accuracy of 50.66%, combined with an 85% positive return rate, shows evidence that the selected features tend to capture real predictive content as opposed to random noise.

Table 4.12. Summary of key findings from Case 0 baseline analysis.

Dimension	Finding
Feature Reduction	431 → 40 features (90.7% reduction) through correlation, importance, and iterative elimination
Best Individual Accuracy	51.97% (VROC_D_10_Volume) — consistent edge over random baseline
Best Individual Return	+1.707 (VROC_D_60_Volume) — highest economic value generation
Most Balanced Feature	OBV_D_1_Volume — highest Short F1 (0.522) and most balanced Long-Short
Dominant Indicator Family	VROC (Volume Rate of Change) — 3 of the top 5 features by accuracy
Surviving Categories	Volatility overrepresented (42.5%); Volume nearly eliminated (2.5%)
Return Positivity Rate	85.0% of features (34/40) generate positive cumulative returns
Mean Accuracy	50.66% — +0.66 pp above random baseline across all 40 features
Long-Short Asymmetry	Systematic Long bias: +2.9 pp precision, +15.8 pp recall advantage

Short Side Weakness	Net negative Short Return (−0.270) — primary area for improvement
Lookback Dominance	Short-term (0–5 days) features comprise 55% of the selected set.

On the other hand, the strong Long minus Short imbalance (15.76 pp recall difference) and a negative mean Short Return (−0.270) signal that there is still plenty of room for improvement in predicting downtrending prices specifically. VROC as the leading predictive indicator family, the remarkable persistence of Volatility features, and the overall robustness across different lookback periods represent a rather interesting pattern found globally outside of the originally constructed domain set. These insights feed directly into how the creativity enhancement system is designed in the multi-agent feature generation framework implemented for Case 1. In particular, the results suggest that the agentic system should focus on volume-volatility interactions and cross-horizon differential features as feature generation targets, along with new short-side predictors.

5. CASE 1: AGENT-BASED GENERATIVE FEATURE ENGINEERING

As a corollary result of this research, Case 1 makes the main contribution of this thesis: an autonomous, multi-agent framework that replaces the manual process of selecting and calculating technical indicators with a generative process triggered by a large language model (LLM). As opposed to Case 0, which relies on 431 pre-defined indicators extracted from known libraries, Case 1 takes a completely different approach by treating feature engineering as a creative generation task. Instead of selecting from a predefined set, the approach produces new mathematical transformations of OHLCV data, checks for uniqueness through a four-layer deduplication process, and iteratively modifies these transformations according to feedback from predictive performance metrics.

In other words, the assumption is that LLMs trained on a large corpus of quantitative finance literature, scientific papers, and programming code encode long-dormant knowledge about legitimate mathematical relationships that no single technical analysis book has managed to fully capture. This is achieved by providing these models structured prompts, creativity seeds, and an iterative feedback loop that systematically explores a feature space well outside of standard technical analysis.

The system works as a pipeline where in each iteration the Feature Generator agent generates 50 candidate features, which are then deduplicated, run, trained, and evaluated. Results from every cycle inform insights for the next, creating a self-optimizing feedback loop. Five different LLM configurations were evaluated per feature generator, repeating 30 iterations with 50 features per iteration, yielding as many as 1,500 candidate features for each LLM. Final comparative analysis was conducted on the top 20 features outputted by each LLM.

Figure 5.1 presents a structural comparison between Case 0 and Case 1 across seven key dimensions, highlighting the paradigm shift from static, library-dependent feature engineering to dynamic, LLM-driven generative feature engineering.

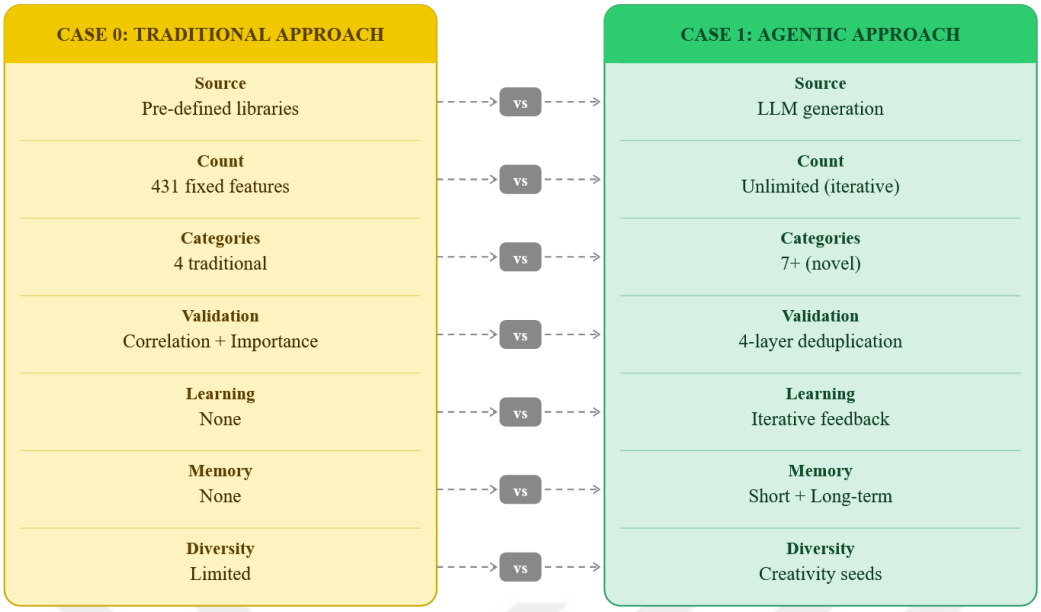

Figure 5.1: Structural comparison between Case 0 (Traditional) and Case 1 (Agentic) approaches

5.1 System Architecture

The multi-agent framework is implemented using LangGraph's StateGraph, a directed acyclic graph (DAG) execution engine that manages state transitions between specialized nodes. The architecture comprises eight distinct nodes, each responsible for a specific aspect of the feature engineering pipeline. Figure 5.2 illustrates the complete system architecture and data flow between nodes.

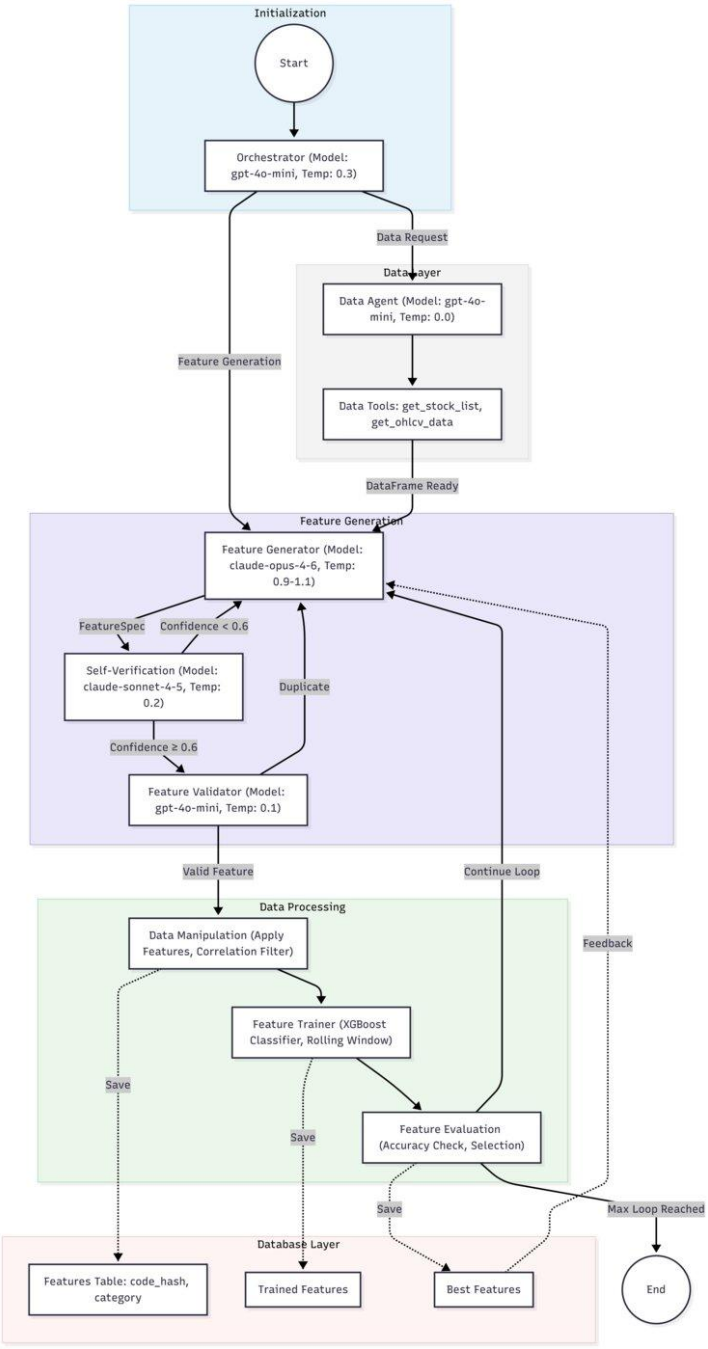

Figure 5.2: Complete system architecture showing eight specialized nodes, data flow, and database interactions

5.1.1 Graph-based agent orchestration

The input state dictionary in a particular step is used as the base, and each node reads from and writes to a shared state dictionary. Using conditional edges that evaluate the current status to determine what node will be next, LangGraph controls the execution order. It handles more complicated control flows, including retry loops (when deduplication is unsuccessful), skip paths (when no features pass validation), and

iterative cycles (returning to the Feature Generator after an evaluation). As shown in Figure 5.2, the Feature Validator can redirect flow back to the Feature Generator when too many duplicates are observed, while the Feature Evaluator loops to keep generating until its maximum loop count.

This state dictionary, implemented as a Python TypedDict, has 25 fields that cover the working DataFrame, generated features, loop counters, token usage tracking and feedback strings. This shared state serves to the short-term memory of the system preserving relevant information both inside a single loop execution and across multiple iterations.

5.1.2 Node configuration and LLM assignment

Each node is assigned an LLM model, temperature setting, and maximum token budget according to its functional requirements. Table 4.1 presents the complete node configuration for the Claude Opus 4.6 preset, which was one of the five configurations evaluated.

Table 5.1. Multi-agent node configuration (Claude Opus 4.6 preset).

Node	Role	LLM Model	Temp.	Description
Orchestrator	Route requests	gpt-4o-mini	0.3	Parses user intent, determines workflow path
Data Agent	Fetch OHLCV data	gpt-4o-mini	0.0	Retrieves stock data from SQL Server database
Feature Generator	Generate features	claude-opus-4-6	0.9–1.1	Produces novel feature code via structured output
Self-Verification	Pre-check uniqueness	claude-sonnet-4-5	0.2	Confidence-scored uniqueness assessment
Feature Validator	4-layer dedup	gpt-4o-mini	0.1	Name, hash, self-verify, semantic validation
Data Manipulation	Apply & process	N/A (code)	N/A	Executes feature code, shifts, cleans data
Feature Trainer	XGBoost training	N/A (XGBoost)	N/A	Rolling window training with per-asset evaluation
Feature Evaluator	Select best	N/A (code)	N/A	Ranks features, generates feedback for next loop

These temperature settings follow an intentional strategy: Feature Generator (0.9–1.1 to encourage creative diversity), Self-Verification node (0.2 for critical performance)

and Orchestrator node (0.3 in order to keep routing stable). The Data Agent, on the other hand, operates at T=0 to provide deterministic interactions with the database.

5.2 Multi-LLM Provider System

One of the main design principles behind the framework is provider-agnostic operation so that features generated by LLMs can be fairly compared under consistent conditions. An automatic provider-detection mechanism, depending on model-name prefix, allows the system to accommodate OpenAI and Anthropic models alike. Provider discovery, role assignment, default configurations, and token tracking architecture are shown in Figure 5.3.

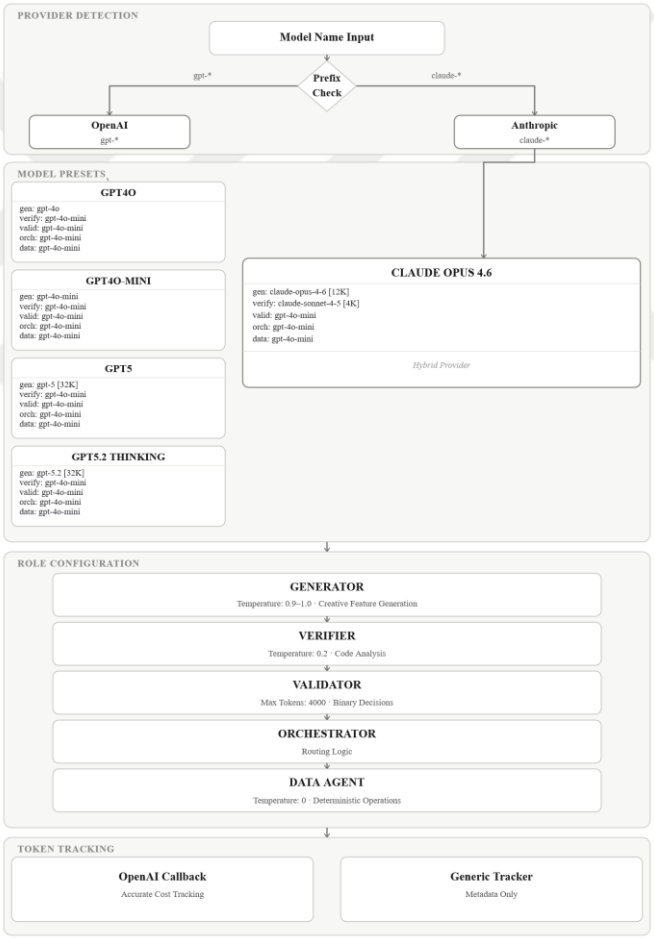

Figure 5.3: Multi-LLM provider system with automatic detection, role-based assignment, and preset configurations

5.2.1 Provider detection and configuration

The ModelConfig class implements routing based on the model name prefix, it starts sending requests to the OpenAI API if any of model names starting with "gpt-" or "o1-", and to Anthropic API if model names start with "claude-". Hybrid configurations are possible since each role Generator, Verifier, Validator, Orchestrator and Data Agent can work independently with any provider. An example is the Claude Opus preset: generation and verification by Anthropic models, orchestration and validation using OpenAI's gpt-4o-mini, balancing creativity quality vs cost.

5.2.2 Evaluated LLM configurations

Five distinct LLM configurations were evaluated as feature generators, while the supporting infrastructure (orchestration, validation, data handling) remained constant across all experiments. Table 5.2 presents the evaluated configurations.

Table 5.2. Evaluated LLM configurations for feature generation.

Configuration	Generator Model	Provider	Max Tokens	Base Temp.
ChatGPT 4o	gpt-4o	OpenAI	12,000	0.9
ChatGPT 4o mini	gpt-4o-mini	OpenAI	12,000	0.9
ChatGPT 5	gpt-5	OpenAI	32,000	0.9
ChatGPT 5.2	gpt-5.2	OpenAI	32,000	0.9
Claude Opus 4.6	claude-opus-4-6	Anthropic	12,000	1.0

5.2.3 Token tracking and cost management

The framework implements provider-agnostic token tracking through a unified interface. For OpenAI models, the system leverages the native callback mechanism for precise cost attribution at the node level. For Anthropic and other providers, a generic TokenUsageTracker extracts usage metadata directly from response objects. All token consumption is systematically logged per-node and per-loop, persisted to structured tabular records for post-hoc cost analysis, and accumulated within the state dictionary for real-time runtime monitoring.

5.3 Feature Generation Pipeline

The Feature Generator node acts as the brain of the system. A structured prompt is compiled and transformed into executable Python source code which calculates new technical indicators. These features are generated in batches of 10; for each loop there

are five of these, so the total number of candidate features per iteration is 50. The complete flow of feature generation, from the state of the input to preparation, creativity injection, generation with LLMs, and structured output is shown in Figure 5.4.

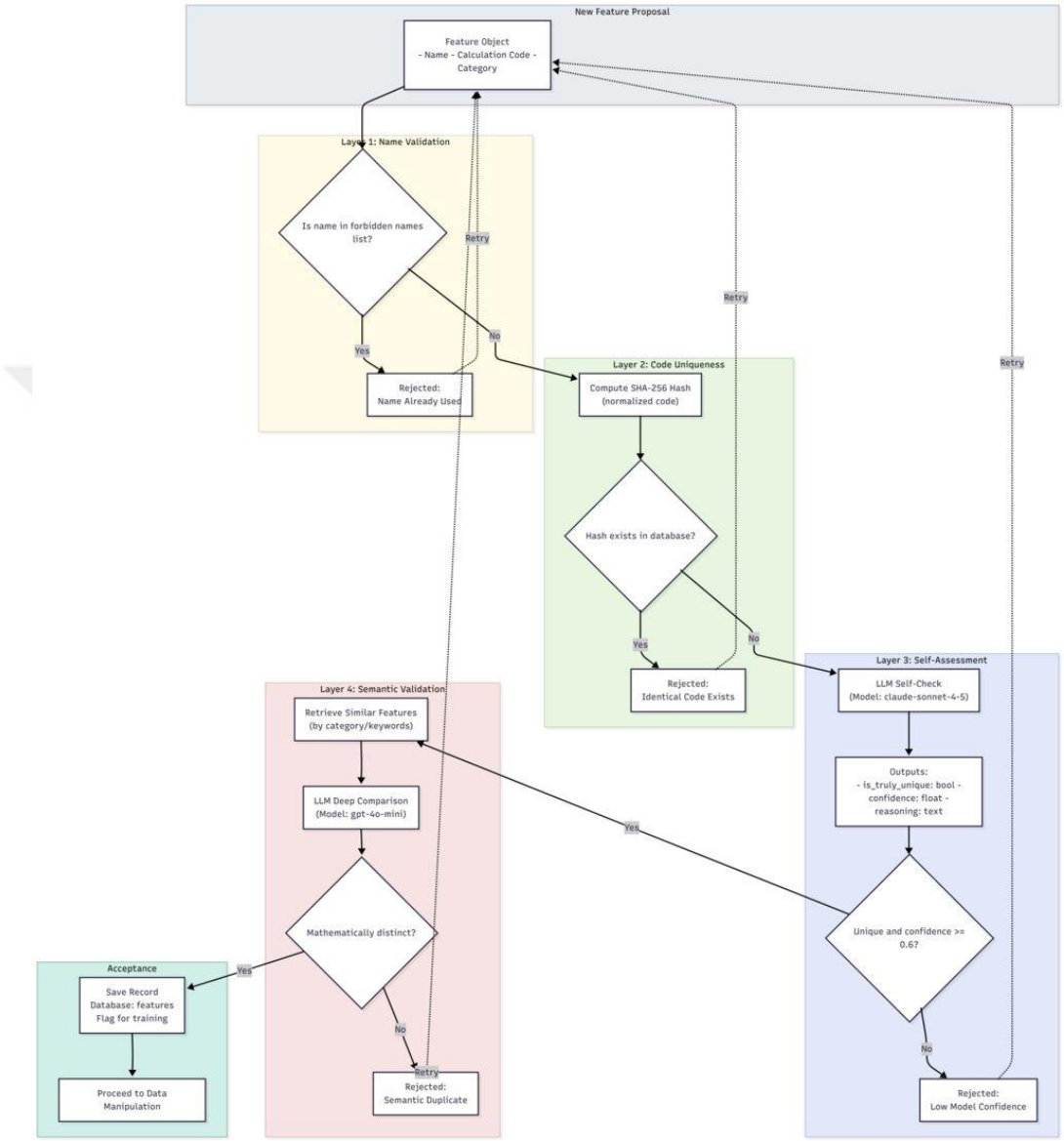

Figure 5.4: Feature generation pipeline showing preparation, creativity engine, LLM call, and structured output stages

5.3.1 Structured output schema

The LLM is instructed to produce output conforming to a Pydantic *FeatureSpec* schema, which enforces structured generation with the following fields:

- **feature_code:** Unique identifier (e.g., F_001, F_002)
- **feature_name:** The exact DataFrame column name produced by the code

- **feature_description:** Natural language explanation of what the feature measures and its predictive rationale
- **feature_calc_code:** Executable Python code using df, pd, np, math, statistics, and scipy.stats
- **uniqueness_reasoning:** Mathematical justification for why this feature differs from existing ones
- **category:** Feature classification (existing or newly invented category)

The structured output mechanism ensures that every generated feature is immediately parseable, executable, and traceable. The system performs automatic reconciliation between the declared feature_name and the actual column name generated by the code, correcting mismatches via regular expression matching.

5.3.2 Prompt engineering strategy

Each batch prompt is dynamically constructed from multiple components that evolve across loops:

- **Role Definition:** The LLM is cast as a "brilliant quantitative researcher" inventing features that predict next-day stock returns, with explicit instructions to think beyond standard technical indicators.
- **Difficulty Level:** Assigned via round-robin from five levels (see Section 4.2.5), shaping the expected novelty.
- **Creativity Seeds:** Three randomly sampled mathematical concepts from a pool of 22 (see Section 4.2.5).
- **Forbidden Names List:** Complete list of previously generated feature names to prevent recreation.
- **Best Features Feedback:** Performance summaries of top features from previous loops for in-context learning.
- **Category Guidance:** Distribution analysis of existing categories, identifying underexplored areas.
- **Data Schema:** Available columns, data types, date range, and asset count.
- **Safety Rules:** Ten strict rules, including look-ahead bias prevention, mandatory use of .transform() for groupby operations, prohibition of helper function definitions, and column reference validation.

5.3.3 Code safety and execution constraints

Generated feature code is subject to multiple safety mechanisms before execution:

Look-Ahead Bias Detection: A regex-based scanner checks for negative shift operations (e.g., shift(-1), shift(-2)) that would introduce future data into the training set. Any feature containing such patterns is immediately rejected.

Recursion Protection: The Python recursion limit is temporarily reduced to 300 during feature execution, preventing infinite loops from crashing the system.

Timeout Enforcement: Each feature is executed in a daemon thread with a 60-second timeout. Features exceeding this limit are terminated and rejected.

Column Validation: Before execution, all DataFrame column references in the code are validated against the available schema. Features referencing non-existent columns are rejected pre-execution.

Function Definition Prohibition: Features containing "def" statements are rejected, forcing all computation to be inline and preventing complex, hard-to-debug code structures.

5.4 Creativity Enhancement System

To prevent the LLM from converging on a narrow set of feature types across many loops, the framework implements a two-dimensional creativity enhancement mechanism combining mathematical concept seeds with graduated difficulty levels. Figure 5.5 presents the complete creativity system architecture, showing how seeds and difficulty levels are combined through batch assignment and injected into the prompt construction pipeline.

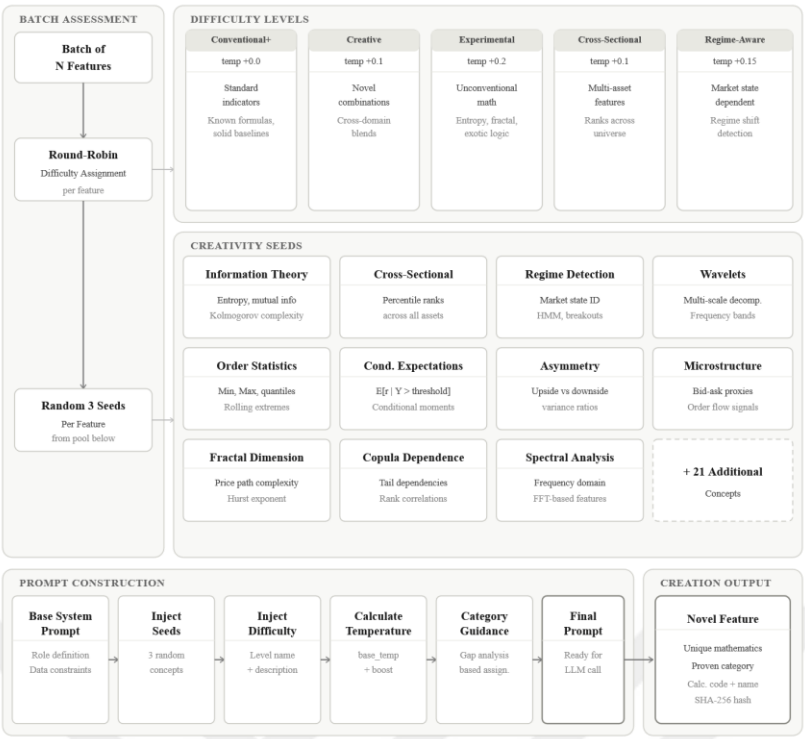

Figure 5.5: Feature generation pipeline showing preparation, creativity engine, LLM call, and structured output stages

5.4.1 Mathematical concept seed

The system draws on a pool of 22 creativity seeds, each tied to a particular mathematical domain or area for analysis. Before each batch, three seeds are randomly chosen to inject into the prompt as creative inspiration. Every seed contains a concept name, a hint that describes its financial utility, and an example code that is meant to create the first reference. Table 5.3 shows some of these creativity seeds.

Table 5.3. Selected creativity seeds from the pool of 22 mathematical concepts.

Concept	Description / Application
Information Theory	Shannon entropy of binned returns measures uncertainty and predictability of the distribution
Fractal / Self-Similarity	Hurst exponent, rescaled range analysis; measures trending (H>0.5) vs mean-reverting (H<0.5) behavior
Cross-Sectional Ranking	Rank each stock against all others on each date; percentile rank and z-score vs peers
Regime Detection	Identify market states via rolling statistics: high/low volatility, trending/mean-reverting
Wavelets / Frequency	Decompose price into frequency components; differences of EMAs as wavelet-like decomposition
Conditional Expectations	Statistics conditioned on events: mean return on high-volume days, volatility after gaps

Asymmetry / Tail Behavior	Upside vs downside analysis: skewness, tail ratios, gain-to-pain ratio
Microstructure Patterns	Candle body ratios, shadow analysis, gap patterns, open-vs-close dynamics
Volume-Price Divergence	Divergence between price trend and volume trend signals potential reversals
Symbolic Regression	Unusual combinations: $(high-low)^2/volume$, $\log(close/open) \times \sqrt{volume}$
Multi-Scale Analysis	Compare same metric at different time scales; ratio of 5-day to 60-day metrics
... (11 more)	Including: Nonlinear Transforms, Order Statistics, Distance Metrics, Time-Decay Weighting, Rolling Correlation, Compression Detection, Graph Features, Persistence Measures, Relative Value

5.4.2 Difficulty levels and temperature modulation

Five difficulty levels are assigned to batches in round-robin order, each adding a temperature boost to the base temperature and providing specific instructions that shape the expected novelty level. Table 5.4 presents the difficulty level configuration.

Table 5.4. Difficulty levels with temperature boost and instruction summary.

Level	Temp. Boost	Instruction Summary
Conventional+	+0.0	Known financial concepts with original twists; modified indicators, unusual parameters
Creative	+0.1	Novel features combining concepts from different domains; nonlinear transforms, conditional logic
Experimental	+0.2	Exotic math: entropy, fractal measures, wavelet decompositions, symbolic regression
Cross-Sectional	+0.1	Features comparing stock to all others on same date via groupby('date') operations
Regime-Aware	+0.15	Market regime detection, volatility shifts, conditional statistics by market state

5.4.3 Dynamic category system

In contrast to Case 0, which uses a static set of four categories (Trend, Momentum, Volatility, Volume), Case 1 uses dynamic categorization. The framework provides 12 seed categories to start with, but intentionally encourages the LLM to come up with new categories when the canonized features do not seem to completely capture a feature's concept. The initial seed categories are Trend, Momentum, Volatility, Volume, CrossSectional, Regime, Statistical, Microstructure, Interaction, Temporal, Fractal, and InformationTheory. Similar categories between iterative loops which are less exploited can serve as feedback to guide the diversification process.

5.5 Four-Layer Deduplication System

One primary challenge is to guarantee uniqueness of features across generation loops. An LLM may output features that are semantically identical but with different names, or generate syntactically different code implementing the same mathematical transformation. To avoid this problem, a cascading four-layer validation framework is designed on the basis of different types of duplicates. The full deduplication process consists of Name Check, Crypto Hash, LLM Self-Check, and Semantic Validity (as shown in Figure 5.6).

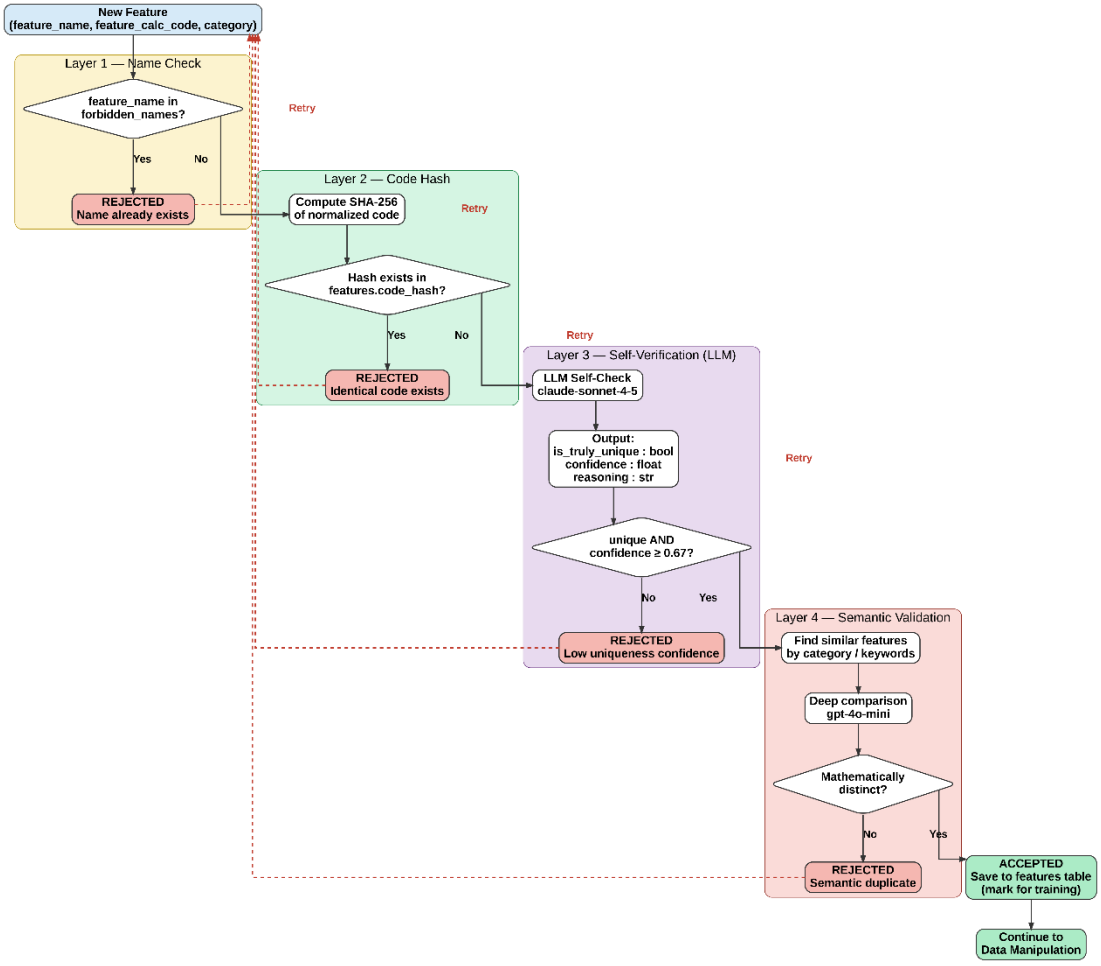

Figure 5.6: Four-layer deduplication system with cascading validation and retry mechanism

5.5.1 Layer 1: name check

The first and fastest layer performs literal string matching with a list of names that should not be allowed. This includes all features from the SQLite database where applied_successfully=1, and also all entries in the best_features table. In addition, at each generation session new features which were accepted are added immediately to

the forbidden set in order not to use them again inside the same batch. This layer works in O(1) time complexity per feature and eliminates the most obvious duplicates.

5.5.2 Layer 2: code hash

The second layer addresses the "same formula, different name" problem through code normalization and cryptographic hashing. Before hashing, the feature code undergoes four normalization steps.

Comment Removal: All Python comments (lines beginning with #) are stripped.

Target Column Placeholder: The left-hand side assignment (e.g., df['vol_20'] =) is replaced with a generic placeholder df['__TARGET__'] =, ensuring that renaming a feature does not change its hash.

Whitespace Normalization: All consecutive whitespace is collapsed to single spaces, and leading/trailing whitespace is removed.

Case Conversion: The entire normalized string is converted to lowercase.

The normalized code is then hashed using SHA-256, with the first 16 hexadecimal characters retained as the code hash. This hash is stored in the features table and indexed for fast lookup.

5.5.3 Layer 3: LLM self-verification

The third layer employs a separate LLM (Claude Sonnet 4.5, temperature 0.2) to perform analytical uniqueness assessment. Each candidate feature is presented alongside code previews of up to 80 existing features, and the verifier produces a structured VerificationResult containing:

- **is_truly_unique:** Boolean uniqueness determination
- **confidence:** Float score between 0.0 and 1.0
- **reasoning:** Natural language justification
- **most_similar_existing:** Name of the closest existing feature

Features are accepted when is_truly_unique = True and confidence $\geq$ 0.6. The verification prompt is very explicit with these requirements. If a formula yet structurally identical, bears the same criteria but different name, it counts as a duplicate; Similarly, if the same concept is applied within a like window, it also

qualifies as duplicate. On the other hand, separate mathematical ideas are treated as different entities, as is any new combination of old ones.

5.5.4 Layer 4: LLM semantic validation

The 4th and the last layer uses gpt-4o-mini (temperature = 0.1) to perform a deep semantic comparison. This layer works on the features that have already passed through Layers 1–3 in conjunction with the full corpus of existing feature descriptions and code snippets. It takes a quantitative approach compare to find semantic equivalence even through syntactic differences. If of the features in a batch, more than 30% fail semantic validation, all features trigger a retry cycle This cycle points back to the Feature Generator with explicit feedback about which features were rejected and why (iteration arrows again in Figure 5.6).

5.6 Memory System Architecture

The framework implements a three-tier memory architecture that enables learning across loops and resilience to interruptions. Figure 5.7 illustrates the complete memory system showing the relationships between short-term state, long-term SQLite storage, episodic feedback memory, and the checkpoint system.

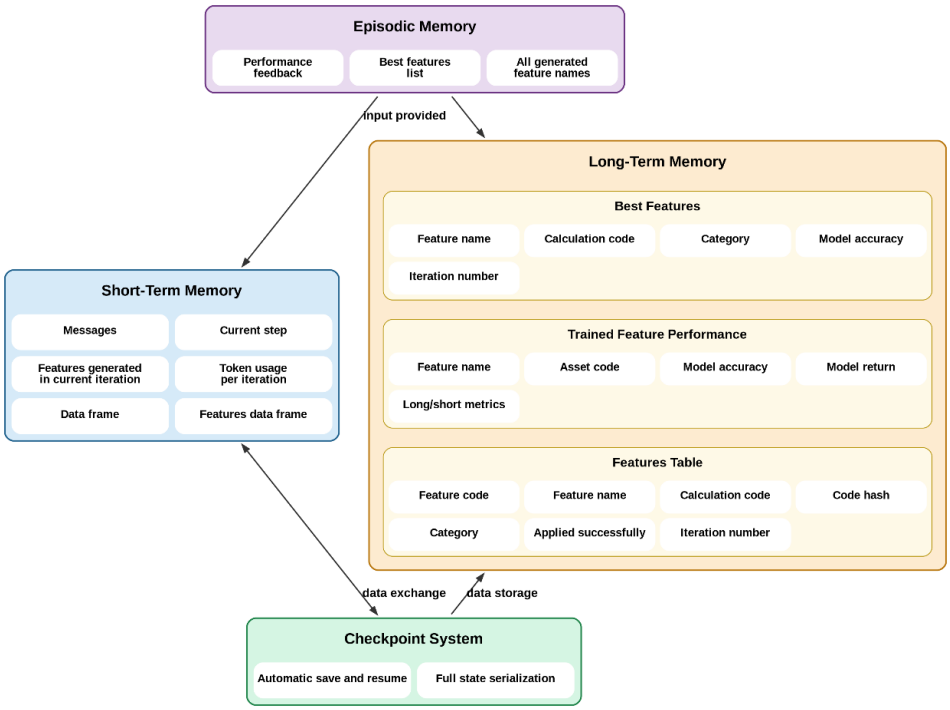

Figure 5.7: Three-tier memory architecture: short-term state, long-term SQLite database, and episodic feedback memory

5.6.1 Short-term memory (state dictionary)

LangGraph State Dictionary: This acts similar to a working memory and contains about 25 fields which track the current execution context. It has the active DataFrame, a list of all features generated in the current loop iteration, loop counters, token usage, and step identifiers. This memory is volatile and resets at the beginning of every workflow execution, but some key fields persist between loops within a single run.

5.6.2 Long-term memory (sqlite database)

Persistent storage is implemented through a SQLite database containing three tables, as shown in Figure 5.7:

features: The main table is where all the features are stored and contains feature_code (primary key), feature_name, feature_description, featured_calc_code, category, code_hash (this would be indexed to help with very fast deduping), loop_number, thread_id and applied_successfully. This table is monotonically expanded through loops and serves as a single source of truth for deduplication.

trained_features_performance: Training results with per-asset, per-month granularity including model_accuracy, model_return, Long/Short precision, recall, F1 scores, and return metrics. Indexed on feature_code, feature_name, and model_accuracy for efficient querying.

best_features: A curated table of top-performing features with a UNIQUE constraint on feature_name, updated at the end of each loop by selecting features above the 75th percentile of accuracy. This table is cleared and rebuilt each loop to reflect cumulative performance.

5.6.3 Episodic memory (performance feedback)

Reused from memory over longer time horizons, episodic memory appears as structured strings of feedback along with the Feature Generator's prompt. This feedback includes a distribution of categories (the presence of feature types in the top k) and discusses underexplored categories (those present in the features table but missing from best_features), top 10 best features with descriptions for context learning, and all the eliminated names. It effectively closes the feedback loop, enabling the system to learn continuously from its successes and errors.

5.6.4 State restoration and checkpointing

It features a state restoration system that allows resuming after interruptions. In the initialization stage, the restore_previous_state function queries the SQLite database for all previously applied features, reconstructs both the best_features_list and the all_unique_features_list, rebuilds the feedback string, and loads the basic OHLCV data. Next, the loop counter is assigned to the maximum value of loop_number recorded in the database, making it possible to continue from where execution left off.

5.7 Data Manipulation and Preprocessing

Once features pass all validation layers, they enter the Data Manipulation node for execution, preprocessing, and preparation for training.

5.7.1 Feature execution

Features are applied sequentially to a copy of the raw OHLCV DataFrame. The code for each feature is executed in a sandboxed environment, with access to df, pd, np, math, statistics and scipy.stats. Execution happens in a daemon thread that is killed after 60 seconds at most; this provides timeout protection. Success is verified by checking that at least one column has been added. Features that do not execute successfully are logged and ignored in future processing.

5.7.2 Look-ahead bias prevention

After all features are applied, the entire set of non-base columns is shifted by one day using groupby('code').shift(1). This is identical to the approach used in Case 0 and ensures that at any given date, only information from previous trading days is available for prediction.

$$Feature_t \; = \; ComputedValue_\{t - 1\} \tag{5.1}$$

5.7.3 Data cleaning

The preprocessing pipeline carries out three cleaning steps: removing columns that are entirely NaN, removing infinite values in any column, and dropping rows with NaN values using dropna(). Non-numeric columns are transformed to data type float64, and boolean columns are type-cast into integers. Then the data is filtered by date, as training involves only data after January 2023.

5.8 Training and Evaluation Pipeline

The training and evaluation methodology from Case 0 is retained except for the source of features, which differs in this case. The overall training pipeline is shown in Figure 5.8, which includes input preprocessing, correlation analysis, feature importance scoring, rolling-window training and evaluation with database updates.

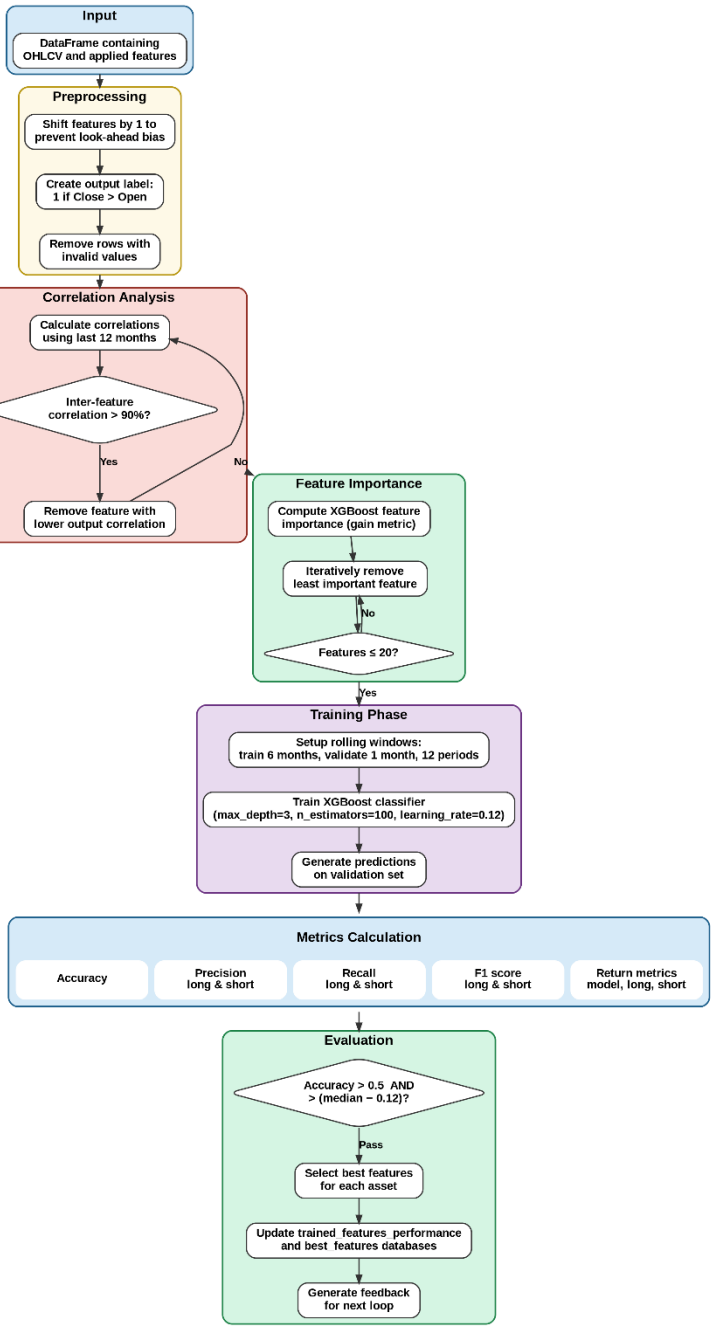

Figure 5.8: Training and evaluation pipeline with rolling window XGBoost and performance-based feature selection

5.8.1 XGBoost training configuration

The same XGBoost classifier configuration is used for every generated feature as in Case 0: max_depth=3, n_estimators=100, learning_rate=0.12, min_child_weight=6, gamma=0.3, subsample=0.80, colsample_bytree=0.80, reg_alpha=0.1, reg_lambda=2; with automatic scale_pos_weight to balance classes. The training process adopts a rolling-window technique, wherein each 6-month period is treated as a training set while the next month is considered as validation. This procedure covers consecutive 12 months to test various periods of both types, thus keeping every data point intact for analysis.

5.8.2 Per-asset evaluation

Training is conducted separately for 20 S&P 100 assets with the following ticker evaluation set: NFLX, TSLA, NVDA, INTC, F, PFE, AVGO, AMD, AAPL, T, BAC, AMZN, META, GOOGL, ORCL, MSFT, VZ, GOOG, FCX. The system records accuracy, return, Long/Short precision, recall, F1 score and directional returns for 12 months of validation for every feature-asset pair.

5.8.3 Feature selection and best features update

The Feature Evaluator aggregates training results by feature_name (averaging across all assets and months) and selects features above the 75th percentile of mean accuracy for inclusion in the best_features table. The evaluator implements a three-layer deduplication strategy during insertion: feature_details are deduplicated before merging, the merged result is deduplicated after merging, and INSERT OR REPLACE is used for atomic upserts with UNIQUE constraint enforcement.

5.9 Feedback Loop Mechanism

The iterative nature of the framework is sustained by a structured feedback mechanism that connects the Feature Evaluator's output to the Feature Generator's input.

The generate_feature_feedback function generates a long string of feedback after every evaluation cycle that contains: (1) bar plots indicating the distribution and category of feature type amongst the best-performing features, making it clear which types give the highest performance; (2) information regarding categories underrepresented in the features table but not well represented in the best performers,

along with specific recommendations for generating more features around those categories; (3) descriptions and accuracies for the top 10 best-performing features providing concrete examples for in-context generation by the LLM; and (4) a complete list of forbidden names to disallow duplication on future runs.

This information is included directly in the system prompt for the Feature Generator in future iterations. The best_features get better over time; hence the in-context examples get better, thus creating a positive feedback loop. Simultaneously, the ever-growing list of prohibited names heightens the pressure to seek actual novelty, so that the LLM is forced deeper and deeper into increasingly creative realms of mathematics.

5.10 Experimental Protocol

To ensure systematic comparison across LLMs, the following standardized experimental protocol was followed:

5.10.1 Execution parameters

- **Loops per LLM:** 30 complete generation-validation-training-evaluation cycles
- **Features per loop:** 50 (5 batches × 10 features)
- **Maximum candidates per LLM:** 1,500 (30 × 50)
- **Final selection per LLM:** Top 20 features by mean accuracy
- **Target assets:** 20 S&P 100 stocks
- **Data period:** 2008-01-01 to 2026-01-30
- **Prediction target:** Binary classification of daily Open-to-Close direction

5.10.2 Controlled variables

Across all five LLM configurations, the following variables remained constant: the OHLCV data source, the 20-asset evaluation set, the preprocessing pipeline (shift, clean, filter), the XGBoost training configuration, the rolling window schedule, the evaluation metrics, the deduplication system, and the feedback mechanism. The only variable was the LLM used for feature generation (the Generator node), with its associated provider-specific token limits and temperature constraints.

5.10.3 Database isolation

A different SQLite database is used for each LLM configuration (for example, db_claude_opus_4_6_1500.db, db_gpt_4o_1500.db) to ensure total isolation between experiments. This prevents sharing a common forbidden names list, code hashes, or performance feedback across LLM generation histories, and avoids cross-contamination of deduplication states.

5.11 Summary

This is the main contribution of the thesis as seen in Case 1, which substitutes the 431 predefined technical indicators found in Case 0 with a large language model (LLM)-driven generative process. It changes how feature engineering is viewed (as a creative task where the system generates unique mathematical transformations of OHLCV data and makes incremental improvements to those transforms based on performance). This architecture is implemented as a directed state machine of eight different node types sharing a 25-field dictionary for the current state in LangGraph. It is provider-agnostic as far as routing, working with both OpenAI and Anthropic models, and supports hybrid setups and per-node temperatures. The structured Pydantic outputs from the Feature Generator produce 10 batches of features, which iterate through dynamically created prompts. These include role definitions, round-robin difficulty adjustments, three randomly sampled creativity seeds from a pool of 22 mathematical concepts, forbidden-name lists and best-feature feedback along with ten safety rules. Safety constraints include look-ahead bias detection, recursion limits, restricting execution time to 60 seconds, and column validation.

A cascading four-layer deduplication system is implemented with exact name-matching, SHA-256 hash of normalized code, LLM self-verification upon threshold-confidence, and batch-level retry for deep semantic validation to ensure feature uniqueness. The framework supports a three-tier memory architecture composed of (1) LangGraph state in memory (volatile), (2) write-once SQLite database per configuration with trained_features_performance and best_features tables, and (3) episodic feedback strings injected into future prompts. It supports learning across iterations and can be resumed after interruption. Accepted features are run in a sandbox where the data is shifted one day ahead to prevent look-ahead bias. The data is then cleaned and each feature is trained separately using the same XGBoost configuration

as Case 0, with a rolling window of 6-month training and 1-month validation on 20 S&P 100 assets. Features with performance metrics above the 75th accuracy percentile are promoted into a best_features table, and their respective performance metrics are fed back into the next generation cycle.

A standardized protocol of 30 loops × 50 features (1,500 candidates per run; top 20 retained) was utilized to evaluate five LLM configurations. Each experiment was conducted in its own SQLite database to prevent contamination and hold every variable constant except the Generator LLM.

6. COMPARATIVE ANALYSIS, KEY FINDINGS, AND CONCLUSION

In this chapter, a cross-evaluation between the two experimental cases defined in this thesis is carried out and the main conclusions are synthesized. Case 0 is a classic feature engineering baseline (431 technical indicators) from Alpha158 and Achelis libraries. This baseline is compared directly against Case 1, which leverages an agent-based generative feature engineering framework operated across five unique configurations of a large language model. The comparison uses ten different performance metrics that include accuracy, directional returns, precision, recall, and F1 score across several supporting discussions on aggregate performance, distributional characteristics, directional asymmetry of the results, composition of the categories used for training and testing in relation to each metric, correlation between all metrics (for instance: does higher precision indicate higher F1 or not?), and performance of individual features. The chapter concludes with a summary of the major findings, followed by a discussion on contribution, limitations, and future research directions.

6.1 Analysis Overview and Data Preparation

This section presents a comprehensive comparison of the features created by the conventional task execution strategy (Case 0) alongside five LLM-based agentic configurations (Case 1). This evaluation contains predictive performance across a selection of metrics, studies directional trading behaviour, details different distributions in feature categories, and finally provides a comprehensive ranking amongst all six methodologies.

A multi-step filtering process was utilized to build the analysis dataset, in order to provide a level playing field comparison of all methods. The initial top 30 features were selected from the raw experimental outputs of each LLM source based on a composite measure of model performance. Realizing that mathematically equivalent features could be the product of different LLM configurations independently, a cross-source deduplication step was created to discard features that had been used in more

than one group. After deduplication, up to 20 features from each source were selected for final analysis (ranked by mean model accuracy), leading to a total of 120 features across six approaches (Case 0, traditional baseline; ChatGPT 4o; ChatGPT 4o mini; ChatGPT 5; ChatGPT 5.2; and Claude Opus 4.6).

All 120 features were evaluated using the same uniform XGBoost classification models with rolling-window validation on 20 S&P 100 assets using a total of 12 sequential monthly data points. Evaluation metrics included model accuracy, model return, Long/Short precision, recall, and F1 scores; directional returns were also calculated. The executive dashboard in Figure 6.1 synthesizes the top value drivers across each of the elements of the analysis.

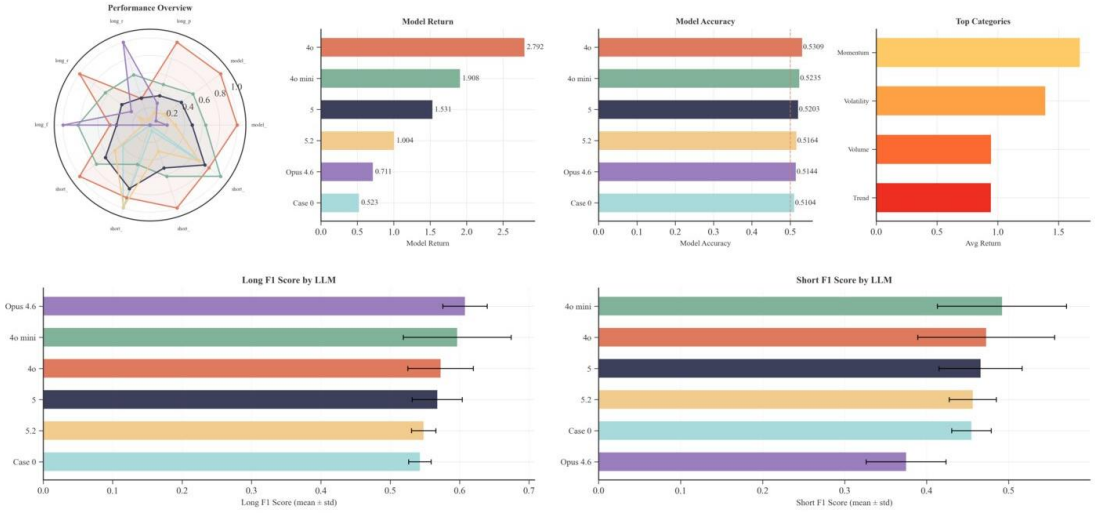

Figure 6.1: Training and evaluation pipeline with rolling window XGBoost and performance-based feature selection

6.2 Overall LLM Performance Comparison

The first dimension of analysis examines the aggregate performance of each LLM source across all evaluation metrics. Table 6.1 presents the mean, standard deviation, and maximum values for each metric, providing a complete statistical profile of the six approaches.

Table 6.1. Mean performance metrics across top 20 features per LLM source.

Source	Accuracy	Return	Long Return	Short Return	Long F1	Short F1
Case 0	0.5104	0.523	0.710	−0.187	0.543	0.455
ChatGPT 4o	0.5309	2.792	1.764	1.028	0.573	0.473
ChatGPT 4o mini	0.5235	1.908	1.378	0.530	0.597	0.492
ChatGPT 5	0.5203	1.531	1.133	0.397	0.568	0.466
ChatGPT 5.2	0.5164	1.004	0.870	0.134	0.548	0.456
Claude Opus 4.6	0.5144	0.711	0.991	−0.280	0.608	0.375

6.2.1 Model accuracy

The best mean model accuracy achieved was 53.09% and is statistically significantly better than the Case 0 baseline of 51.04%, measured by a two-tailed Wilcoxon matched-pairs signed-rank test with a significance level defined at $p < 0.05$ for ChatGPT 4o (+2.05 percentage points). But while this improvement may seem small in absolute terms, it is an important achievement in the world of financial prediction, where even marginal increases in accuracy can translate into large economic gains. Trailing behind was ChatGPT 4o mini with a score of 52.35%, ChatGPT 5 with a score of 52.03%, ChatGPT 5.2 at 51.64%, and finally Claude Opus 4.6 at 51.61%. Significantly, all five LLM-based methods outperformed the classical baseline, indicating that agentic feature engineering creates features with better predictive power than simple technical indicators.

The standard deviation of the accuracy reveals a very distinct dispersion pattern: ChatGPT 4o scored highest (53.09%) with high variability ($\sigma = 0.028$) in comparison to ChatGPT 5.2, which resulted in the highest consistency with much more consistent clustering ($\sigma = 0.0019$). This means that the creative generation method of ChatGPT 4o allows for a greater variation in feature quality, with both great outliers and some ineffective candidates, while the newer GPT models create features more safely but less adventurously.

6.2.2 Model return

The model return metric, which measures the total return of a trading strategy based on the directional prediction of the feature, shows significant differences across the methods. The results are in favor of ChatGPT 4o, which achieved a mean return of 2.792, over five times higher than the baseline set by Case 0 (0.523). ChatGPT 4o mini

was then reported with a 1.908 return, while ChatGPT 5 followed at 1.531, ChatGPT 5.2 at 1.004, and finally Claude Opus came in at 0.711. These differences in scale far outweigh the gaps observed for accuracy, indicating that features generated by LLMs increase prediction power and confidence regarding significant price movements.

Maximum return values stand out in particular: ChatGPT 4o derived one feature (accelerated_trend_dissipator) with a return of 11.74. This shows that the LLM found a very accurate mathematical transformation that can be even more powerful than traditional indicators based on market dynamics.

6.2.3 Multi-metric radar analysis

Figure 6.2 presents a normalized radar chart comparing all six approaches across ten performance dimensions simultaneously. The visualization reveals distinct performance profiles for each approach rather than uniform superiority or inferiority.

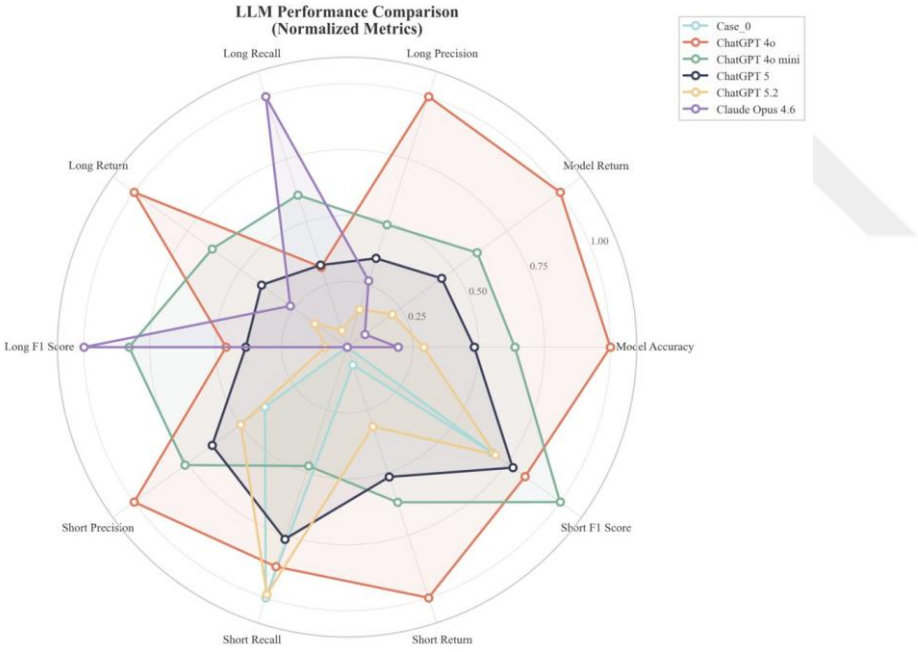

Figure 6.2: Normalized radar chart comparing six approaches across ten performance metrics

Several key insights are revealed in the radar plot. ChatGPT 4o (red) has the most balanced performance profile and performs best for all metrics with respect to accuracy, model return, Long precision, Long Return, Short Precision, and Short Return. However, it is particularly poor at Short recall. ChatGPT 4o mini (green) is the most well-rounded, scoring very evenly across almost all metrics with no significant weaknesses. Claude Opus 4.6 (purple) is an exception with Long recall and

Long F1 being the highest among all methods, but Short-side performance being the lowest; this means it has a stronger directional bias toward Long in its generated features. Case 0 (light blue) is the most centered on all axes, reinforcing that the traditional baseline generally performs lower, yet has a relatively large spike in Short recall, surpassing all LLM method evaluations other than ChatGPT 4o mini.

6.2.4 Detailed metric comparison

Figure 6.3 decomposes the radar analysis into individual bar charts for each of the ten metrics, with the best performer highlighted in full color and other approaches shown in faded bars. This visualization enables precise comparison of magnitudes that the radar chart's normalization obscures.

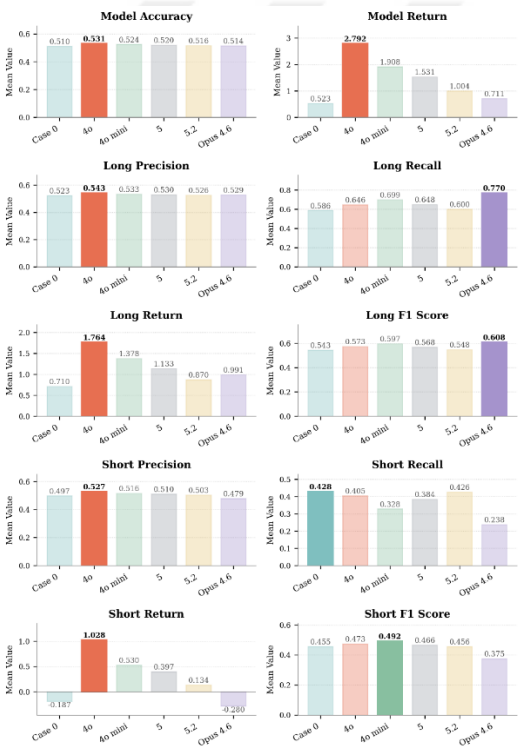

Figure 6.3: LLM performance across all metrics with best performer highlighted per metric

The metric decomposition reveals that no single model is the best in all dimensions evaluated. ChatGPT 4o scores the highest in six out of ten metrics: accuracy, model return, Long precision, Long return, Short precision, and Short return. Meanwhile, Claude Opus 4.6 achieves the highest Long recall (0.770) and Long F1 score (0.608), while ChatGPT 4o mini gets the best Short F1 score (0.492). Case 0 achieves the highest Short recall (0.428) because it predicts Short significantly more often than the

LLM-based models. This behavior suggests that while features from the LLMs yield higher overall accuracy, they systematically favour Long predictions.

6.3 Performance Heatmap and Ranking Analysis

In order to rank definitively on all metrics, Figure 6.4 shows two more heatmaps: the left panel is a heatmap for absolute performance with color intensity indicating magnitude; and the right (Rank) panel converts those absolute values into ordinal ranks (1 = best; 6 = worst) by row (i.e., by metric).

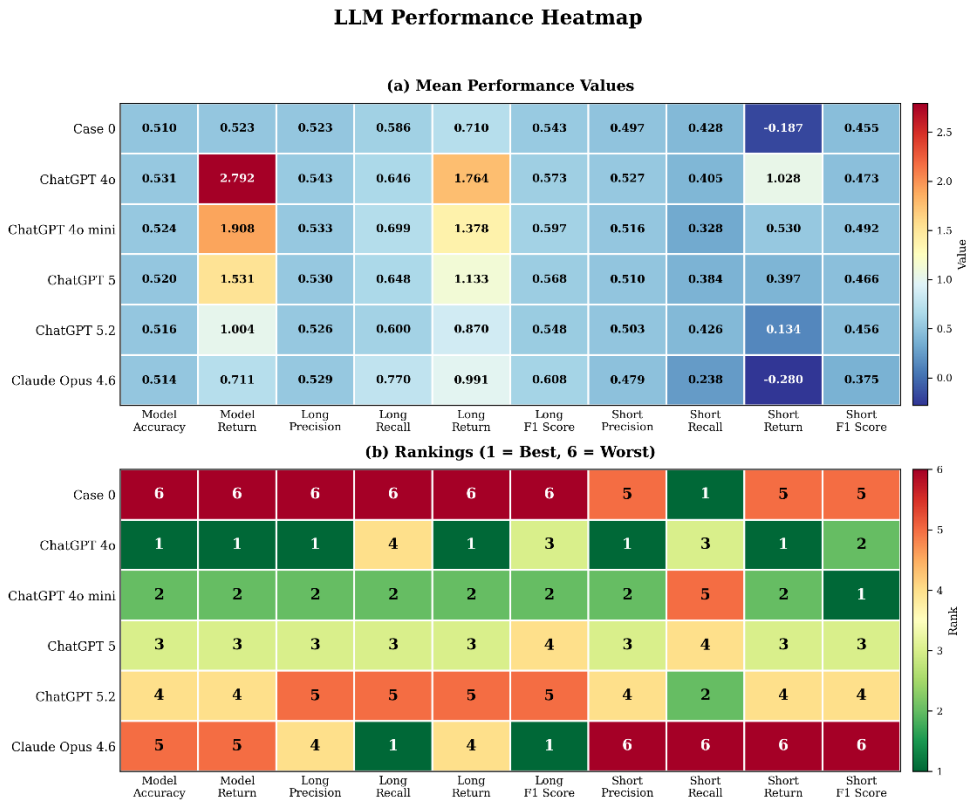

Figure 6.4: Performance heatmap showing absolute values (left) and metric-wise rankings (right) across six approaches

6.4 Feature Performance Distributions

Mean values are great at summarizing the central tendency of each approach, but for executive overviews or investment decisions, understanding the distribution of individual feature performance tends to give some critical insights into consistency and tail behavior, as well as risk characteristics. In Figure 6.5, six key metrics (as in Figure 6.4) are decomposed in box plots for all approaches; diamonds indicate the

means per approach and dashed lines indicate the overall population mean across all groups.

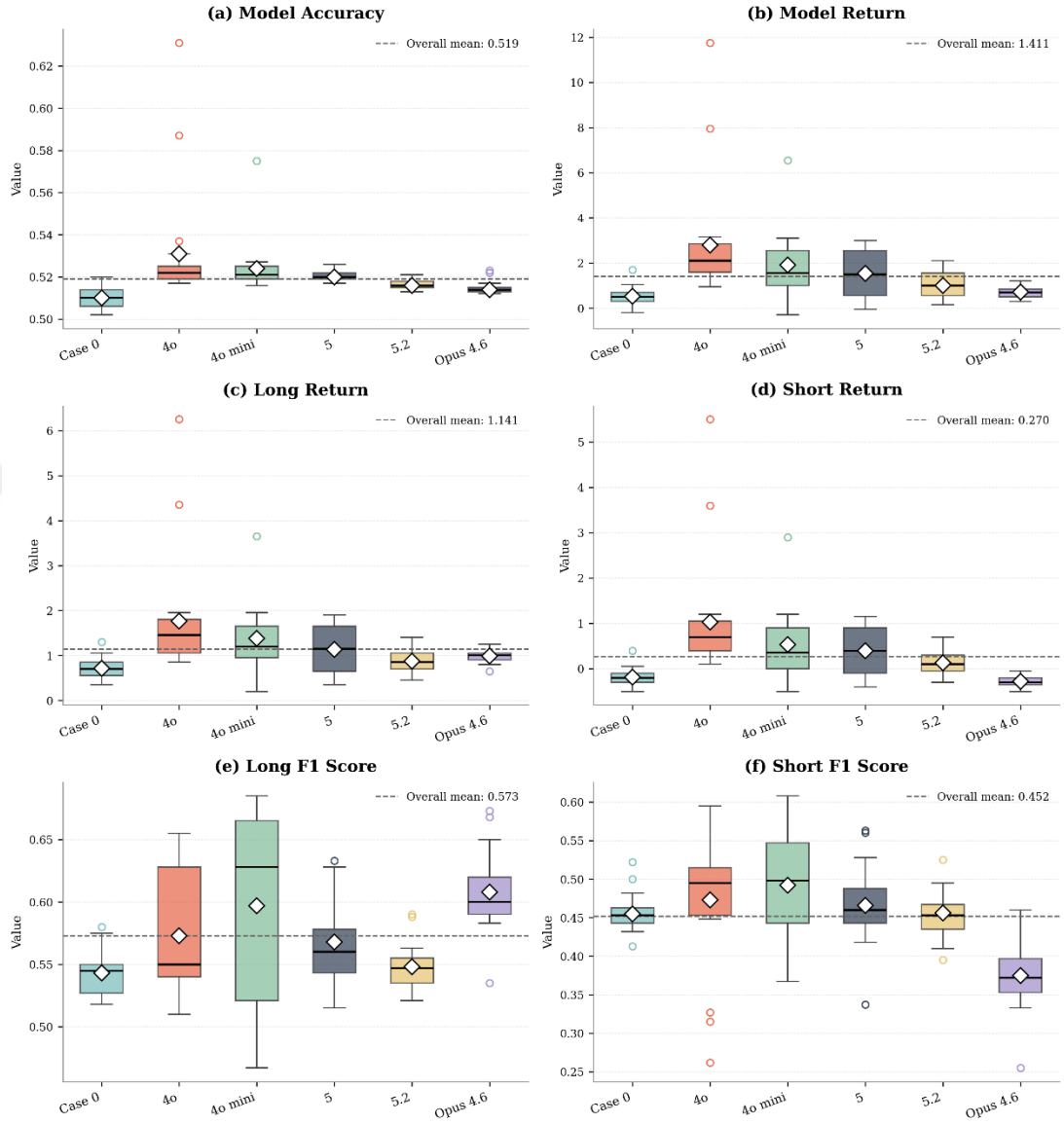

Figure 6.5: Feature performance distributions by LLM source across six key metrics

6.4.1 Accuracy distribution

Accuracy box plots (panel a): ChatGPT 4o clearly outperforms all other methods. ChatGPT 4o shows the largest IQR, with values ranging from about 50.5% to 54.0%, and outliers at over 63.1%. The wide spread is indicative that ChatGPT 4o produces features on a vast spectrum of quality from mediocre to excellent. In comparison, ChatGPT 5 and 5.2 have extremely narrow distributions tightly packed around approximately 52%, suggesting that these newer models produce more uniform but less diverse features. The distribution of Case 0 is the most compressed overall, with

values showing almost no variation around 51%, which corresponds to the intrinsic bounds when choosing only from a fixed set of traditional indicators.

6.4.2 Return distribution

Model return distributions (panel b) underscore the accuracy patterns shown. While the median stays at nearly 2.0, ChatGPT 4o shows significant positive skew, with outliers above 11.0. The importance of this heavy right tail lies in economic significance: a few features selected by ChatGPT 4o produced highly profitable trading signals. ChatGPT 4o mini exhibits a similar but less pronounced trend. On the other hand, the Case 0 distribution is tightly clustered around 0 with very low variance, while Claude Opus 4.6 has a slightly better median accuracy because its return curve is so close to the baseline that the difference between the two curves can barely be seen.

6.4.3 F1 Score distributions

The Long F1 distributions (panel e) present an unexpected finding: Claude Opus 4.6, which ranked fifth in accuracy and return, achieves the highest median Long F1 score with a notably tight distribution centered around 0.63. ChatGPT 4o mini also shows strong Long F1 performance with the widest IQR extending above 0.65. This decoupling between accuracy/return and F1 score suggests that different LLMs optimize for different aspects of prediction quality—some produce features that predict direction more profitably (higher returns) while others produce features that are more balanced in their long prediction quality (higher F1).

The Short F1 distributions (panel f) tell a complementary story. ChatGPT 4o mini dominates with the highest median and widest distribution, while Claude Opus 4.6 collapses to the bottom with a median near 0.38. This confirms the Long bias previously identified in Claude Opus 4.6's generated features—its features are excellent at identifying Long opportunities but poor at predicting Short movements.

6.5 Feature Category Analysis

The 120 features in the analysis span four categories: Momentum, Volatility, Trend, and Volume. This section examines how category membership relates to predictive

performance and how different LLMs distribute their top features across categories. Table 6.2 presents the category-level performance statistics.

Table 6.2. Performance metrics by feature category.

Category	Count	Accuracy	Return	Long F1	Short F1
Momentum	58 (48.3%)	0.5215	1.674	0.576	0.453
Volatility	31 (25.8%)	0.5193	1.390	0.570	0.453
Trend	17 (14.2%)	0.5140	0.942	0.575	0.447
Volume	14 (11.7%)	0.5170	0.943	0.561	0.454

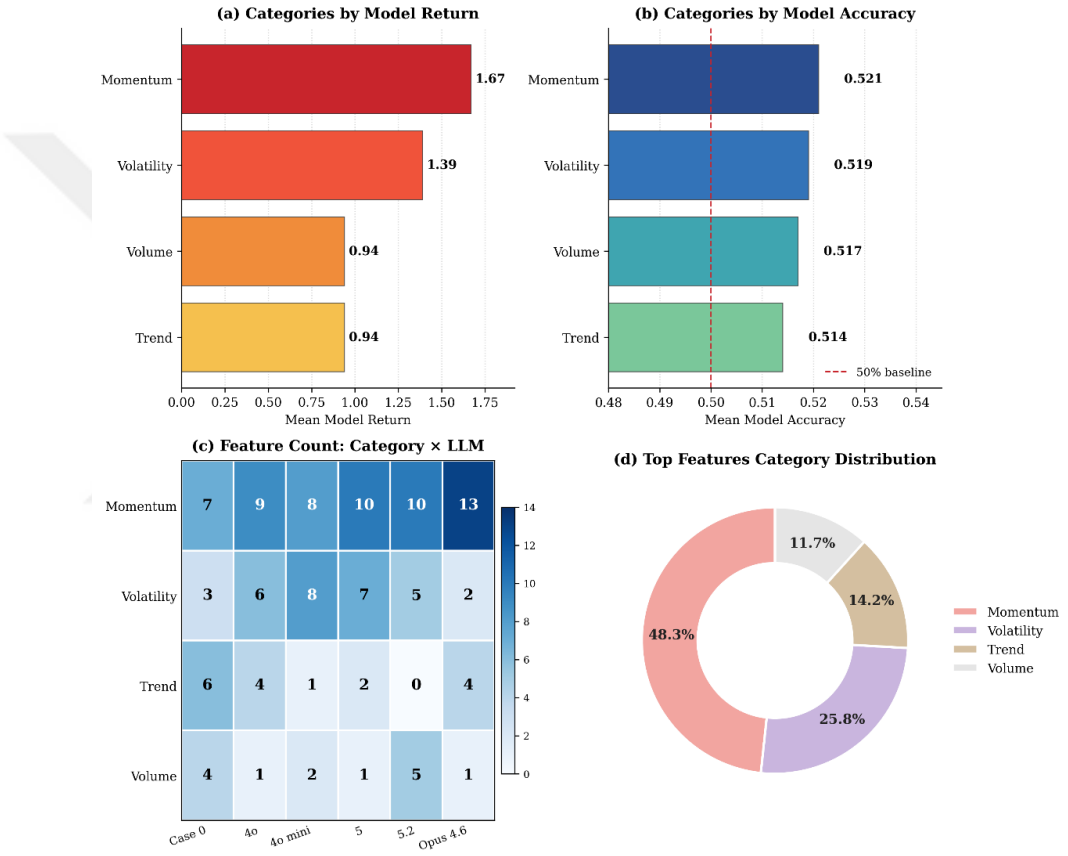

Figure 6.6: Feature category analysis: (a) return by category, (b) accuracy by category, (c) category count by LLM, (d) overall distribution

6.5.1 Category-level performance

Momentum features dominate both in quantity and quality, comprising 48.3% of all top-20 features and achieving the highest mean accuracy (0.5215) and mean return (1.674). Volatility features rank second in both count (25.8%) and return (1.390). Trend and Volume features show similar accuracy levels (0.514 and 0.517 respectively) and nearly identical returns (0.942 and 0.943), suggesting that in the

context of LLM-generated features, these two categories offer comparable but lower predictive value compared to Momentum and Volatility.

An important result is that the relevance of Momentum-based features in both Case 0 and Case 1 suggests that whether features are hand-picked from classical libraries or generated by LLMs, momentum-type indicators provide the strongest prediction of market signals during the daily forecasting horizon. This observation corroborates the momentum anomaly widely reported in the literature and suggests that LLMs have discovered and generalized these stock market structures independently by deriving new mathematical formulations. BURDAYIM

6.5.2 Category distribution by LLM

By Category and LLM, panel (c) in Figure 6.6 illustrates a stacked heatmap of feature counts. There are clear patterns here as Claude Opus 4.6 produces the most overall Momentum features (13/20), indicating a strong categorical focus. On the other hand, Case 0 has a most balanced distribution; that is: 7 Momentum, 6 Trend, 4 Volume and 3 Volatility features. ChatGPT 4o mini generates the most Volatility features (8) with ChatGPT 5.2 showing a strong preference for Volume features (5), and zero Trend features.

Such distributional differences imply that every LLM is establishing its own creative preference in which features are produced. ChatGPT 4o mini generally emphasizes Volatility-based expression, Claude Opus 4.6 mainly skews toward Momentum construction, and lastly, ChatGPT 5.2 strongly gravitates to volume-based trajectories. By contrast, the traditional Case 0 approach simply yields the most uniform category distribution possible relative to its library composition as feature selection is entirely driven by empirical performance and not generative bias.

6.6 Metric Correlation Analysis

Understanding the relationships between evaluation metrics is essential for interpreting the results and designing composite performance measures. Figure 6.7 presents the Pearson correlation matrix for all 10 performance metrics computed across the 120 features.

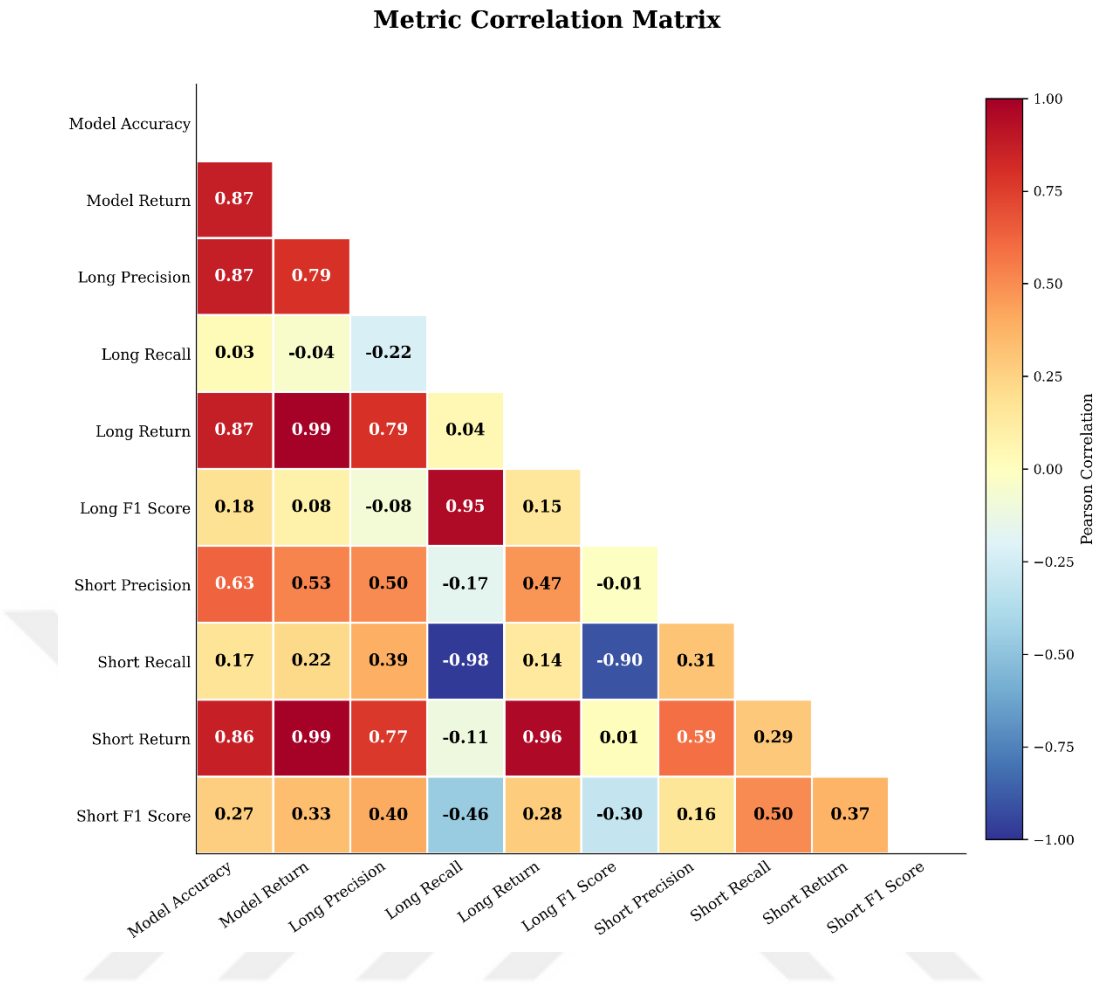

Figure 6.7: Pearson correlation matrix across ten performance metrics computed over 120 features

This, then brings up a number of structural relationships that are much better illustrated in the context of the correlation matrix in this evaluation framework. There exists a very strong positive linear relationship between model accuracy and the model return ($r = 0.87$), the Long return ($r = 0.87$), and the Long precision ($r = 0.87$). This close association lays to rest the claim that features with higher accuracy are also associated with greater trading returns, thus lending further support to the use of accuracy as a reliable proxy for economic performance in this context.

Perhaps the most surprising result is a near-perfect negative correlation between Long recall and Short recall ($r = -0.98$). This is a core trade-off: features that successfully bring up more Long opportunities (high Long recall) always fail to deliver Short opportunities (low Short recall), and vice versa. This relationship is mathematically expected in a binary classification setting, as raising the threshold for one class means sacrificing recall from the other, but the size of -0.98 suggests something about LLM-

generated features makes them preferentially likely to encounter directional calibration problems.

Long F1 score shows a strong positive correlation with Long recall (r = 0.95) but a strong negative correlation with Short recall (r = −0.90), indicating that F1 improvements on the Long side come at the direct expense of Short-side classification quality. Conversely, Short return shows high correlation with model return (r = 0.99) and Long return (r = 0.96), suggesting that features generating high returns do so through both Long and Short positions simultaneously—a desirable characteristic for bidirectional trading strategies.

6.7 Winner Analysis and Composite Ranking

To establish a definitive hierarchy among the six approaches, this section synthesizes the multi-metric analysis into composite scores and head-to-head comparisons. Figure 6.8 presents four complementary views of the competitive landscape.

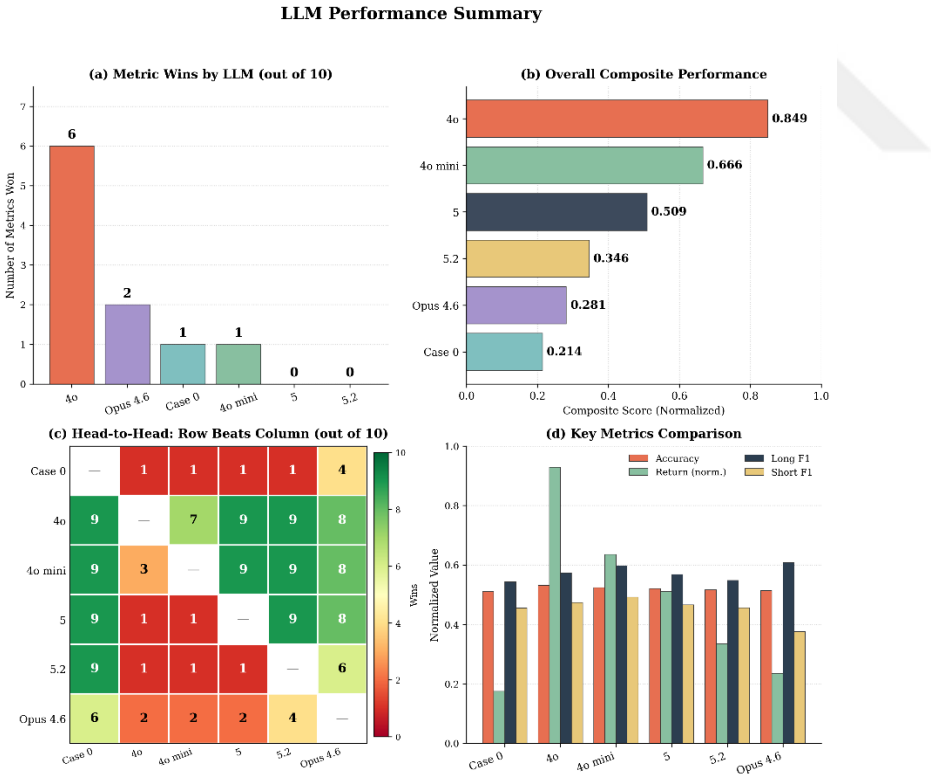

Figure 6.8: Winner analysis: (a) metric wins by LLM, (b) composite score, (c) head-to-head matrix, (d) key metrics comparison

6.7.1 Metric wins

Panel (a) counts how many metrics out of 10 each method achieves the highest mean. In 6 metrics, including precision, model return (Long and Short), ChatGPT 4o takes the lead, demonstrating clear superiority. Claude Opus 4.6 achieves the highest in 2 metrics, Long recall and Long F1 score, demonstrating its specialization in Long-side prediction. On the other hand, ChatGPT 4o mini and Case 0 take one each, Short F1 and Short recall respectively. ChatGPT 5 and ChatGPT 5.2 do not lead in any individual metric; their advantage is in maintaining consistent mid-range performance rather than excelling on any single metric.

6.7.2 Composite performance score

Panel (b) shows a normalized composite score capturing performance across all measures in one scalar value. The composite ranking is ChatGPT 4o (0.849), ChatGPT 4o mini (0.666), ChatGPT 5 (0.509), ChatGPT 5.2 (0.346), Claude Opus 4.6 (0.281), and Case 0 (0.214). This ranking breaks down into three distinct tiers: a top tier consisting of ChatGPT 4o only, an intermediate tier of ChatGPT 4o mini and ChatGPT 5, followed by a lower tier made up of ChatGPT 5.2, Claude Opus 4.6, and Case 0. The difference between ChatGPT 4o (0.849) and the second-place ChatGPT 4o mini (0.666) is 0.183 points, less than half the gap of the second to fifth position separation of 0.385 points, further emphasizing its dominant performance over all other models.

6.7.3 Head-to-head comparison

Panel (c) presents a pairwise dominance matrix where cell (i, j) shows the number of metrics (out of 10) where row i outperforms column j. This granular view reveals the strength of dominance relationships. ChatGPT 4o beats Case 0 on 9 of 10 metrics and beats ChatGPT 5.2 on 9 of 10 metrics, confirming its broad superiority. ChatGPT 4o mini shows identical dominance over Case 0 (9/10) and matches ChatGPT 4o against the weaker approaches, but loses to ChatGPT 4o on 7 of 10 metrics in direct comparison.

An interesting finding is that Claude Opus 4.6 beats Case 0 on only 6 of 10 metrics—the weakest Case 1 advantage over the traditional baseline. In head-to-head comparisons with other LLM approaches, Claude Opus 4.6 consistently wins only 2–4 metrics, primarily the Long-side measures. This confirms that, while Claude Opus

4.6 generates features with unique Long-prediction strengths, its overall performance profile is less competitive than that of the ChatGPT family.

6.8 Top-Performing Individual Features

Beyond aggregate comparisons, examining the highest-performing individual features provides insight into what types of mathematical transformations the LLMs discovered. Table 6.3 presents the 10 features that achieve the highest model accuracy across the entire 120-feature dataset.

Table 6.3. Top 10 features by model accuracy across all 120 features.

Rank	Feature Name	LLM Source	Category	Accuracy	Return
1	accelerated_trend_dissipator	ChatGPT 4o	Momentum	0.6309	11.736
2	convoluted_time_flux	ChatGPT 4o	Momentum	0.5870	7.922
3	momentum_gradient	ChatGPT 4o mini	Momentum	0.5750	6.544
4	differential_price_intensity	ChatGPT 4o	Momentum	0.5369	2.809
5	deviant_high_low_rally	ChatGPT 4o	Volatility	0.5316	2.709
6	extreme_return_regime_conflict	ChatGPT 4o mini	Volatility	0.5241	3.093
7	average_drawdown	ChatGPT 4o mini	Momentum	0.5242	3.006
8	deviation_amplitude_signal	ChatGPT 4o	Momentum	0.5225	3.171
9	_cs4d_run	ChatGPT 5	Volatility	0.5216	2.758
10	cross_sectional_volume_kurtosis	ChatGPT 4o	Trend	0.5193	3.171

ChatGPT 4o has six of the top 10 features, ChatGPT 4o mini has three, and ChatGPT 5 contributed one. None of the top 10 features belong to Case 0, ChatGPT 5.2, or Claude Opus 4.6. Momentum features appear in seven of these top positions, showing how predictive this category is.

The best homogeneous feature is accelerated_trend_dissipator, with the highest accuracy of 63.09%, 4.4% point greater than that of the second-best feature and a return of 11.74. This is an interesting outlier performance: the feature was generated at a high temperature level using ChatGPT 4o in an Experimental difficulty batch meaning that the creativity enhancement system's exploration of unconventional mathematical formulations can sometimes yield very useful transformations. Its name

suggests it measures the rate of momentum decay and combines trend-following aspects with mean-reversion signal detection – a hybrid notion usually not heuristically modeled in classical indicator libraries.

The presence of features with new names such as convoluted_time_flux, extreme_return_regime_conflict_index, and cross_sectional_volume_kurtosis tells us that the LLMs were not merely copying existing indicators under different guises; they were generating genuinely novel mathematical abstractions. These names exhibit the creativity cues and complexity parameters contained in the generation prompts, confirming that the creativity enhancement system specified in Section 5.4 is indeed able to push the LLMs beyond conventional feature design limits.

6.9 Key Findings

The comparative analysis yields several definitive findings that directly address the thesis research questions:

Finding 1: LLM-generated features consistently perform far better than traditional technical indicators. For all five LLM configurations, the features produced through this process had both a substantially greater mean accuracy and a significantly larger median return than Case 0. Such a fairly large improvement is notable: ChatGPT 4o attains a 5.3x model return over Case 0 (2.792 vs. 0.523), which clearly demonstrates that agentic feature engineering has economic value as a tool to make predictive performance economically viable.

Finding 2: It is, however, clear that there is a relative performance ranking with large language models (LLMs). In accuracy, recall and composite rankings, ChatGPT 4o leads, followed by ChatGPT 4o mini, ChatGPT 5, and then ChatGPT 5.2 and Claude Opus 4.6. Interestingly, the seemingly older and smaller ChatGPT 4o outperforms both ChatGPT 5 and 5.2 (which are more recent models), suggesting that model scale or recency is not necessarily correlated with feature engineering proficiency. ChatGPT 4o exhibits a wider range of creative diversity in its generative behaviour, while the latest models produce narrower but more consistent outputs.

Finding 3: This systematic Long bias can be seen in the results for all LLM-generated features. As demonstrated by the near-perfect negative correlation between Long recall and Short recall ($r = -0.98$) along with consistently higher Long metrics across each

method evaluated, there seems to be an inherent structural bias whereby predictions are biased toward Longs. The bias is most obvious in Claude Opus 4.6, which has a Long recall of 0.770 and Short recall of only 0.238; ChatGPT 4o mini is the least biased, with a Long recall of 0.699 and Short recall of 0.328. These findings have important consequences for the design of actionable trading strategies: while the vast majority of features generated by large language models should be complementary to Long-only strategies, directional balancing with extreme caution will need to play an essential role in Long-Short models if this positive bias is not desired.

Finding 4: Among the features best able to predict performance, momentum features consistently dominate. Momentum-type features achieve the best accuracy (0.5215) and return ratio (1.674) among all data sources, constituting 48.3% of the features ranked at the top 10%. This ranking is consistent with both traditional and LLM-generated approaches, further solidifying the momentum anomaly as the primary driver of short-term predictive signals in equities.

Finding 5: Note that the variance in feature quality is highly dependent on the LLM itself. The widest distribution is for ChatGPT 4o and the narrowest one is for ChatGPT 5.2 (accuracy $\sigma = 0.028$, max = 0.631; accuracy $\sigma = 0.0019$, max = 0.520). This points to a fundamental trade-off between creative exploration and consistent output in LLM-based feature engineering. Specifically, the most effective unique features emerge from generation processes that have high variance.

Finding 6: It is obvious that Case 0 (the classical approach) performs the worst. The traditional baseline is ranked in last place (9 out of 10) across most of the metrics and receives the lowest composite score (0.214), as well as losing head-to-head comparisons against all LLM approaches on 6 to 9 out of the 10 metrics. The somewhat more even (better balanced) directional predictions of Case 0 lead to stronger average Short recall, but this does not translate into profit, reflected in the negative mean Short return (−0.187).

6.10 Conclusion

This thesis set out to investigate if an agent-based framework, theoretically fueled by large language models, could systematically generate unique and economically meaningful features for forecasting financial market price behavior better than those

derived from the traditional ad-hoc feature engineering processes typically employed in the quantitative finance ecosystem. In response to this question, two fully functioning experimental cases were constructed and tested under the same setting: a traditional baseline (Case 0) based on 431 technical indicators inferred from the Alpha158 and Achelis libraries, followed by correlation elimination and XGBoost importance scoring; and a generative agentic framework (Case 1), configured as an eight-node LangGraph StateGraph through five separate LLM configurations. The two cases were run on daily open-to-close directional prediction of 20 S&P 100 equities with scaling and drift, using a rolling-window validation over 12 months in-sample for predictions. The outputs were then evaluated with a common analytical pipeline over the top 120 features and ten complementary performance metrics.

Results provide strong and consistent support for the hypothesized agentic framework. All five LLM configurations produced a top-feature set that outperformed the Case 0 baseline in both mean accuracy and mean return, with ChatGPT 4o producing about 5.3 times the cumulative model return of the standard approach. The comparative results showed that the advantage of the framework was not just in terms of pure predictive accuracy, but also reflected in more complex mathematical forms and a wider range of categories summarised by LLM-generated features (e.g., especially those related to momentum) and unique characteristics where none would generally appear even though models were trained on very similar datasets. At the same time, the analysis uncovered widespread Long-side bias in LLM-generated features, a definitive performance hierarchy amongst five configurations (with their variances), and evidence of a trade-off between creative diversity and final output stability. More specifically, ChatGPT 4o, an older and less constrained model, showed the widest performance distributions and highest individual feature performances, whereas the newest models yield narrower but more consistent outputs.

The main novel contribution of this thesis is a reproducible agent-based framework for systematic generative feature engineering in financial markets, its implementation and empirical validation. The framework operationalizes a concrete architecture that breaks down LLM-driven hypothesis generation, deterministic evaluation, deduplication and ranking in order to mitigate the purported failures of unguided LLM outputs in quantitative procedures. It also includes a referential methodological framework for fair cross-comparison of approaches, built on an ad cultural spec metric

assessment using a taxonomy of ten metrics in tandem with an explainable four-layer deduplication pipeline that yields genuine predictive signals, isolated from variation due to style or lexicon. Lastly, the thesis serves as an empirical lighthouse of five state-of-the-art LLM architectures on a common feature engineering task, documenting their relative performance but also different creativity and direction biases, a finding with immediate applied relevance for researchers and practitioners wanting to incorporate generative models into quantitative research workflows.

The current study results should, however, be interpreted with caution and several limitations should be acknowledged. To avoid confounding factors, the experimental universe was limited to 20 large-capitalization U.S. equities, which offered a liquid, highly studied testbed but excludes smaller-capitalization and non-U.S. assets. The benchmark only looked at daily open-to-close directional prediction, and thus was not directly applicable to longer time frames or intraday microstructure phenomena, or portfolio-level risk. Each output from the five LLM configurations was accessed through their own respective interfaces, which means each model's outputs were conditioned on proprietary training data and alignment processes that cannot be controlled or examined fully. While the return calculations leave out transaction costs, slippage, short-selling constraints, and other market frictions, the economic magnitudes reported should be regarded as estimates of profitability before costs. Finally, the framework is evaluated in deterministic mode but uses stochastic generative phases; thus two independent runs of an identical LLM configuration will not yield the same feature set. As such, the results are a single instance drawn from a generative distribution rather than a predetermined, permanent record.

These limitations directly indicate a number of explicit directions for future research. Natural extensions include considering different assets (beyond large capitalization equities) and treatment of the prediction horizon (e.g. including mid-cap, small capitalisation, international markets, fixed income and/or crypto assets). This would allow us to test the extent to which features generated by LLMs generalize across regimes with different microstructural properties. A more ambitious route involves introducing transaction costs, capacity constraints and portfolio-level risk management to build on the previous feature-level analyses and assess a strategy as a whole. Third, agentic architecture merits study consideration: future work could investigate reflective and self-criticism reasoning patterns, tool-aided generation with direct

access to streaming market data or research databases, and multi-agent debate protocols where competing feature proposals are refined through adversarial interaction. Fourth, the systematic investigation of the Long-side bias discovered in this thesis is an emerging area for research. This may entail best-in-class classification-level tuning, balanced generation goals across categories, and post-hoc calibration to make libraries directionally neutral in their features. Lastly, it is recommended that the benchmarking piece be expanded to include LLM providers and model sizes which have publicly available weights in order to more fully characterize the current state-of-the-art in terms of creative feature-generation tools available for quantitative researchers.

Overall, building on the comparative analysis and previous research in this thesis leads to a clear conclusion: agent-based feature engineering using generative features from large language models is not just another approach to traditional feature engineering but fulfills a much more advanced role. In a design with an ever-present and rigid evaluation pipeline, features generated by LLMs outperformed a strong classical baseline on nearly all dimensions measured while injecting an analytically rich layer of creative diversity into the feature construction process. The presented framework is deliberately structured, repeatable and universal so that it can provide a useful basis for research going forward regarding both relevant domains of agentic artificial intelligence and quantitative finance.

6.11 Future Works

While the experimental results presented in this thesis demonstrate the viability of agent-based generative feature engineering, the framework itself remains an early-stage architecture, and several concrete extensions can be identified that would improve its efficiency, autonomy, and domain specificity. The following directions outline the most immediate and, in the author's view, most promising paths for subsequent development.

One main concern is the efficiency and speed of the framework. The current implementation requires previously generated features to be added back into the prompt at every step in order to avoid duplication and push the generator in the direction of generating new values. This methodology yields a roughly linear growth in both token utilization, and therefore operating expenses, as the feature library grows.

A scalable approach would be to implement a dedicated database agent as an additional node in the LangGraph StateGraph. The generator would then ask this database agent for structured but concise feedback requests to see if either a candidate formulation (or its closest semantic equivalent) exists in the persistent store, rather than embedding the entire feature history with every prompt. Such a design would disentangle long-term memory from the generation prompt and reduce tokens by orders of magnitude, allowing for feature libraries an order of magnitude larger without raising costs, while enabling more sophisticated retrieval methods (such as embedding-based similarity searches or category-aware lookups) to be layered under top-level generative agents.

One direction that readily flows out of the former. Each LLM configuration was restricted to thirty generation loops, generating fifty features per loop (i.e., approximately 1,500 candidates per configuration). The scale was largely dictated by budgetary concerns rather than methodological considerations. After optimization of the token economy using the aforementioned database-agent mechanism, it is possible to redeploy from a batch experiment setting to a continuously running, server-hosted and self-improving framework. In this setup, feature generation would be an ongoing process with downstream models being retrained at regular intervals; poorly performing features would be automatically retired while families of features that consistently improve predictive performance would continue to get built. Transitioning from what is essentially a limited-life exploratory run to a 24/7 generative pipeline would allow the framework to continuously adjust based on evolving macroeconomic situations and develop an order of magnitude larger and more diverse feature library than feasible in any single experiment.

The third extension regards the domain specialization of agents. For the moment, each LLM-driven node is acting just like a general-purpose reasoning engine; the only thing that guides it is its system prompt and whatever knowledge resides in the shared state. While this setting has been shown to outperform a naive baseline, it does not take advantage of the rich domain knowledge in quantitative finance, market microstructure and empirical asset pricing. One obvious enhancement is to add a RAG layer on each node containing an LLM, specialized for what the node does in the workflow. As an example, the feature-generation agent may have access to a hand-selected database of empirical asset pricing research papers, technical analysis sites, and factor investing literature; the validation agent may be trained on financial statistics textbooks and

backtesting best-practice guides; and the evaluation agent might draw from a repository focused on performance evaluation studies and risk management resources. With this type of role-specific retrieval, the existing generic reasoning agents would become specialist agents that are well-versed in background knowledge similar to domain experts and generate outputs that are even more grounded, defensible and theoretically consistent.

Outside the three main avenues lie a handful of other possible extensions worth noting. The universal Long-side bias pinpointed in Chapter 6 gives rise to a requirement of adding a directional balance system into the generation loop. This could be addressed via category-balanced generation targets, post-hoc re-weighting, or through the creation of an auxiliary agent whose only job is to audit and correct for directional skew in the growing feature library. Likewise, the finding that different LLMs exhibit unique creative signatures motivates investigating multi-LLM ensemble or debate protocols. Such frameworks could allow features proposed by one model to be vetted and enhanced by another, leveraging complementary biases instead of a singular generator. A second exciting direction is exploring supplementary and multi-modal data sources in the context of generation, e.g., news flow, earnings transcripts, macro releases and order-book statistics. In this manner, the framework could create fused features between price-based structures and text- or event-based signals. Finally, the framework might connect with a reinforcement-type feedback loop in which realised trading performance extends outside of sample and feeds back to the generative process. This would enable the system to learn not just which features are statistically predictive, but which ones show real economic strength during live execution.

Overall, these extensions paint a clear path for the framework: from an experimental pipeline limited in scope and prompt length to a cost-effective, runtime platform on which general-purpose generative research can be accelerated with appropriate consideration of economic factors. The architecture proposed in this thesis allows for each of the outlined directions to be followed step-by-step, without needing a complete re-design of the underlying StateGraph. Future work of the author or other researchers will likely expand on this groundwork, narrowing the generative AI gap in quantitative finance.

REFERENCES

Acharya, D. B., Kuppan, K., & Divya, B. (2025). Agentic AI: Autonomous intelligence for complex goals—A comprehensive survey. IEEE Access. https://doi.org/10.1109/ACCESS.2025.1000000

Achelis, S. B. (2001). Technical analysis from A to Z (2nd ed.). McGraw-Hill.

Achiam, J., Adler, S., Agarwal, S., Ahmad, L., Akkaya, I., Aleman, F. L., et al. (2023). GPT-4 technical report. arXiv preprint arXiv:2303.08774

Biswas, A., & Talukdar, W. (2025). Building agentic AI systems. Packt Publishing.

Bornet, P., Wirtz, J., Davenport, T. H., De Cremer, D., Evergreen, B., Fersht, P., et al. (2025). Agentic artificial intelligence: Harnessing AI agents to reinvent business, work and life. Irreplaceable Publishing

Brown, T., Mann, B., Ryder, N., Subbiah, M., Kaplan, J. D., Dhariwal, P., et al. (2020). Language models are few-shot learners. Advances in Neural Information Processing Systems, 33, 1877–190

Cao, L., Xi, Z., Liao, L., Yang, Z., & Cao, Z. (2025). Chain-of-Alpha: Unleashing the power of large language models for alpha mining in quantitative trading. arXiv preprint arXiv:2508.06312

Chen, Z., Chen, W., Smiley, C., Shah, S., Borova, I., Langdon, D., Moussa, R., Beane, M., Huang, T.-H., Routledge, B., & Wang, W. Y. (2021). FinQA: A dataset of numerical reasoning over financial data. In Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing (pp. 3697–3711). Association for Computational Linguistics

Craik, F. I. M., & Lockhart, R. S. (1972). Levels of processing: A framework for memory research. Journal of Verbal Learning and Verbal Behavior, 11(6), 671–684. https://doi.org/10.1016/S0022-5371(72)80001

Cheng, S., Liang, J., Tang, G., Cheng, Z., Chen, C., Peng, Y., Li, J., Li, Z., Huang, K., Ji, W., Xu, G., Tan, X., & Lin, F. (2025). HAWK: Hierarchical multi-agent workflow for multi-agent AI system. arXiv preprint arXiv:2502.02631

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

Guo, J., Wang, S., Ni, L. M., & Shum, H.-Y. (2022). Quant 4.0: Engineering quantitative investment with automated, explainable and knowledge-driven artificial intelligence. arXiv preprint arXiv:2301.04020

Hong, S., Zhuge, M., Chen, J., Zheng, X., Cheng, Y., Zhang, C., Wang, J., Wang, Z., Yau, S. K., Lin, Z., Zhou, L., Ran, C., Xiao, L., Wu, C., & Schmidhuber, J. (2024). MetaGPT: Meta programming for a multi-

agent collaborative framework. In Proceedings of the International Conference on Learning Representations (ICLR 2024

Hu, E., Shen, Y., Wallis, P., Allen-Zhu, Z., Li, Y., Wang, S., Wang, L., & Chen, W. (2021). LoRA: Low-rank adaptation of large language models. arXiv preprint arXiv:2106.09685

Jay, R. (2024). Generative AI apps with LangChain and Python. Apress.

Kakushadze, Z. (2016). 101 formulaic alphas. Wilmott, 2016(84), 72–81

Kanter, J. M., & Veeramachaneni, K. (2015). Deep feature synthesis: Towards automating data science endeavors. In 2015 IEEE International Conference on Data Science and Advanced Analytics (DSAA) (pp. 1–10). IEEE

Khattab, O., Singhvi, A., Maheshwari, P., Zhang, Z., Santhanam, K., Vardhamanan, S., Haq, S., Sharma, A., Joshi, T. T., Moazam, H., Miller, H., Zaharia, M., & Potts, C. (2023). DSPy: Compiling declarative language model calls into self-improving pipelines. arXiv preprint arXiv:2310.03714

Lee, J., Stevens, N., Han, S. C., & Song, M. (2024). A survey of large language models in finance (FinLLMs). arXiv preprint arXiv:2402.02315

Li, G., Hammoud, H. A. A. K., Itani, H., Khizbullin, D., & Ghanem, B. (2023). CAMEL: Communicative agents for mind exploration of large language model society. In Advances in Neural Information Processing Systems (NeurIPS)

Li, Y., Zhang, Y., Sun, Q., Wang, W., Liu, J., Ma, C., Han, J., & Sun, Y. (2025). Co-STEER: Collaborative scheduling and task execution engine for research. arXiv preprint arXiv:2502.04250

Liu, X.-Y., Yang, H., Gao, J., & Wang, C. D. (2021). FinRL: Deep reinforcement learning framework to automate trading in quantitative finance. In Proceedings of the 2nd ACM International Conference on AI in Finance (pp. 1–9). ACM

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

Louck, T., Chen, X., & Wang, Y. (2025). Security analysis of agent-to-agent protocols in multi-agent AI systems. arXiv preprint

Mumuni, A., & Mumuni, F. (2024). Automated data processing and feature engineering for deep learning and big data applications: A survey. Preprint

Murphy, J. J. (1999). Technical analysis of the financial markets: A comprehensive guide to trading methods and applications. New York Institute of Finance

Pandya, K., & Holia, M. (2023). Automating customer service using LangChain: Building custom open-source GPT chatbot for organizations. arXiv preprint arXiv:2310.05421

Park, J. S., O'Brien, J. C., Cai, C. J., Morris, M. R., Liang, P., & Bernstein, M. S. (2023). Generative agents: Interactive simulacra of human behavior. In Proceedings of the 36th Annual ACM Symposium on User Interface Software and Technology (UIST '23) (pp. 1–22). ACM

Qian, C., Liu, W., Liu, H., Chen, N., Dang, Y., Li, J., et al. (2023). ChatDev: Communicative agents for software development. arXiv preprint arXiv:2307.07924

Schick, T., Dwivedi-Yu, J., Dessì, R., Raileanu, R., Lomeli, M., Hambro, E., et al. (2023). Toolformer: Language models can teach themselves to use tools. Advances in Neural Information Processing Systems, 36, 68539–68551

Shi, H., Song, W., Zhang, X., Shi, J., Luo, C., Ao, X., Arian, H., & Seco, L. (2025). AlphaForge: A framework to mine and dynamically combine formulaic alpha factors. In Proceedings of the AAAI Conference on Artificial Intelligence

Shinn, N., Cassano, F., Gopinath, A., Narasimhan, K., & Yao, S. (2023). Reflexion: Language agents with verbal reinforcement learning. Advances in Neural Information Processing Systems, 36, 8634–8652

Sun, S., Xue, Z., Zhang, J., Yang, Z., Li, Z., Han, L., & Zhang, C. (2023). TradeMaster: A holistic quantitative trading platform empowered by reinforcement learning. In Advances in Neural Information Processing Systems (NeurIPS) Datasets and Benchmarks Track

Tran, K.-T., Dao, D., Nguyen, M.-D., Pham, Q.-V., O'Sullivan, B., & Nguyen, H. D. (2025). Multi-agent collaboration mechanisms: A survey of LLMs. arXiv preprint arXiv:2501.06322

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., & Polosukhin, I. (2017). Attention is all you need. Advances in Neural Information Processing Systems, 30, 5998–6008

Wang, G., Xie, Y., Jiang, Y., Mandlekar, A., Xiao, C., Zhu, Y., et al. (2023). Voyager: An open-ended embodied agent with large language models. arXiv preprint arXiv:2305.16291

Wang, L., Ma, C., Feng, X., Zhang, Z., Yang, H., Zhang, J., Chen, Z., Tang, J., Chen, X., Lin, Y., Zhao, W. X., Wei, Z., & Wen, J.-R. (2025). A survey on large language model based autonomous agents. Frontiers of Computer Science. https://doi.org/10.1007/s11704-024-40231-1

Wang, S., Yuan, H., Ni, L. M., & Guo, J. (2024a). QuantAgent: Seeking holy grail in trading by self-improving large language model. arXiv preprint arXiv:2402.03755

Wang, S., Yuan, H., Zhou, L., Ni, L. M., Shum, H.-Y., & Guo, J. (2023). Alpha-GPT: Human-AI interactive alpha mining for quantitative investment. arXiv preprint arXiv:2308.00016

Wei, J., Wang, X., Schuurmans, D., Bosma, M., Xia, F., Chi, E., Le, Q. V., & Zhou, D. (2022). Chain-of-thought prompting elicits reasoning in large language models. Advances in Neural Information Processing Systems, 35, 24824–24837

Wooldridge, M., & Jennings, N. R. (1995). Intelligent agents: Theory and practice. The Knowledge Engineering Review, 10(2), 115–152. https://doi.org/10.1017/S0269888900008122

Wu, Q., Bansal, G., Zhang, J., Wu, Y., Li, B., Zhu, E., Jiang, L., Zhang, X., Zhang, S., Liu, J., Awadallah, A., White, R. W., Burger, D., & Wang, C. (2023). AutoGen: Enabling next-gen LLM applications via multi-agent conversation. arXiv preprint arXiv:2308.08155

Yang, X., Liu, W., Zhou, D., Bian, J., & Liu, T.-Y. (2020). Qlib: An AI-oriented quantitative investment platform. arXiv preprint arXiv:2009.11189

Yao, S., Yu, D., Zhao, J., Shafran, I., Griffiths, T., Cao, Y., & Narasimhan, K. (2023b). Tree of thoughts: Deliberate problem solving with large language models. Advances in Neural Information Processing Systems, 36, 11809–11822

Yao, S., Zhao, J., Yu, D., Du, N., Shafran, I., Narasimhan, K., & Cao, Y. (2023a). ReAct: Synergizing reasoning and acting in language models. In Proceedings of the 11th International Conference on Learning Representations (ICLR 2023)

Yu, L., Chen, W., Qin, Y., & Liu, Y. (2023b). FINMEM: A performance-enhanced LLM trading agent with layered memory and character design. arXiv preprint

Yu, S., Xue, H., Ao, X., Pan, F., He, J., Tu, D., & He, Q. (2023a). Generating synergistic formulaic alpha collections via reinforcement learning. In Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (pp. 5240–5250

Zhang, A., Sheng, L., Chen, Y., Li, H., Deng, Y., Wang, X., & Chua, T. S. (2024). FinAgent: A multimodal foundation agent for financial trading. arXiv preprint

Zhang, B., Fang, Y., Gao, Q., Zhang, L., Wang, L., & Hu, Y. (2020). AutoAlpha: An efficient hierarchical evolutionary algorithm for mining alpha factors in quantitative investment. arXiv preprint arXiv:2002.08245

CURRICULUM VITAE

Name Surname : Mustafa Enes KARAMAN

EDUCATION :

- **M.Sc. :** 2019, University of Birmingham, School of Engineering (Department of Civil Engineering), Geotechnical Engineering and Management Program
- **B.Sc. :** 2014, Yildiz Technical University, Faculty of Civil Engineering, Civil Engineering Department

PROFESSIONAL EXPERIENCE AND REWARDS:

- 2024–Present, Data Scientist & Financial Data Analytics – Navimod Financial Solutions
- 2020–2024, Procurement Specialist – General Directorate of State Hydraulic Works
- 2016–2018, Structural Design Engineer - Yapı Teknik Engineering and Consultancy
- 2016, Technical Office Engineer - Detas Konut Imar A.S.
- 2014–2015, Technical Office Engineer - Polimeks