COMILLAS
UNIVERSIDAD PONTIFICIA

ICAI

GRADO EN INGENIERÍA MATEMÁTICA E INTELIGENCIA ARTIFICIAL

TRABAJO FIN DE GRADO

Enhancing Financial Reasoning Models through Tool-Augmentation and Test-Time Compute (TTC) Techniques

Author: Jaime Pedrosa Comino

Director: Andrés Occhipinti Liberman

Co-Director: Carlos Bellón Nuñez-Mera

Madrid, May 2026

I declare, under my responsibility, that the Project submitted under the title

Enhancing Financial Reasoning Models through Tool-Augmentation and Test-Time Compute (TTC) Techniques

at the ICAI School of Engineering of Comillas Pontifical University in the academic year 2025/2026 is my own work, original and unpublished, and has not previously been submitted for any other purpose.

The Project is not plagiarized, either wholly or partially, and the information taken from other documents is duly referenced.

Signed: Jaime Pedrosa Comino Date: 15 / 06 / 2026

Submission of the project is authorized

THE PROJECT DIRECTOR

Signed: Andrés Occhipinti Liberman Date: 15 / 06 / 2026

THE PROJECT CO-DIRECTOR (if applicable)

Signed: Carlos Bellón Nuñez-Mera Date: 15 / 06 / 2026

Acknowledgments

This project represents a year of work that has been an enriching academic experience, allowing me to explore the intersection of artificial intelligence and quantitative finance while producing results with real practical applicability.

I would like to express my sincere gratitude to my supervisor Andrés Occhipinti Liberman and co-supervisor Carlos Bellón Nuñez-Mera for their continuous guidance throughout this project. Their complementary perspectives, combining rigorous academic research and real-world financial expertise, have been fundamental in shaping both the technical direction and the practical scope of this work.

I am deeply grateful to my family, whose unconditional support throughout these years of study has been the foundation of everything I have undertaken. Home is always where this journey begins and ends.

Finally, I would like to thank Universidad Pontificia Comillas and, in particular, the ICAI School of Engineering, for providing not only the technical knowledge inherent to the degree in Mathematical Engineering and Artificial Intelligence, but also the broader human formation that defines an institution rooted in the tradition of the Society of Jesus.

MEJORA DE MODELOS DE RAZONAMIENTO FINANCIERO MEDIANTE USO DE HERRAMIENTAS Y TÉCNICAS DE TEST-TIME COMPUTE (TTC)

Autor: Jaime Pedrosa Comino
Director: Andrés Occhipinti Liberman
Co-Director: Carlos Bellón Nuñez-Mera

Resumen

Este Trabajo Fin de Grado estudia si el uso de herramientas externas y las técnicas de Test-Time Compute (TTC) mejoran de forma significativa el rendimiento de modelos de razonamiento financiero especializado. Sobre el modelo base FinR1, se evaluaron dos estrategias complementarias: el uso de herramientas externas para acceder a datos de mercado reales y ejecutar cálculos deterministas, y un mecanismo de TTC que genera múltiples trayectorias de razonamiento candidatas y selecciona la más prometedora mediante un proceso de evaluación. Para implementar la primera estrategia, se desarrolló un protocolo ReAct desde cero. La validación empírica sobre el benchmark FinQA y 23 escenarios de inversión real sugiere que la arquitectura propuesta mejora la tasa de respuestas correctas del 27 % en el modelo base al 78 % en la configuración completa, con una reducción sustancial de los fallos operativos observados.

Palabras clave: Modelos de Razonamiento Financiero; Test-Time Compute; Protocolo ReAct; Reward Models; Optimización de Carteras; Inteligencia Artificial Financiera.

Resumen ejecutivo

1 Introducción

La aplicación de la Inteligencia Artificial en las finanzas cuantitativas ha estado tradicionalmente basada en modelos predictivos convencionales. La aparición de los Grandes Modelos de Lenguaje (LLMs) ha ampliado estas posibilidades al permitir que los sistemas analicen y razonen sobre información financiera compleja expresada en lenguaje natural. Sin embargo, aunque han surgido modelos especializados como FinO1 [1], QwQ-32B [2] y FinR1 [3], su integración en entornos de asesoramiento financiero profesional sigue estando condicionada por una cuestión fundamental: el elevado coste asociado a los errores.

Los LLMs son modelos probabilísticos diseñados para generar lenguaje de forma coherente, pero no para realizar cálculos matemáticos con precisión determinista. Cuando dependen únicamente de la información almacenada en sus parámetros, pueden presentar limitaciones importantes. En las evaluaciones iniciales realizadas en este trabajo, la versión base de FinR1, el modelo con mejor desempeño entre los evaluados, falló sistemáticamente en las

pruebas realizadas sobre los 23 escenarios del caso práctico: empleó información histórica como si fuese actual, incapaz de distinguir el contexto temporal de los datos, y no fundamentó ninguna de sus respuestas en datos de mercado verificables. Además, mostró una tendencia a emplear aproximaciones simplificadas, como asignar pesos estándar del 10 % o 20 % a las carteras en lugar de ejecutar una optimización real, y exhibió comportamientos de complacencia (*sycophancy*) [4], generando métricas inexistentes para respaldar las expectativas del usuario.

Para afrontar estas limitaciones, este trabajo evalúa dos estrategias complementarias: el uso de herramientas externas (*tool use*) para acceder a datos reales y realizar cálculos fiables, y las técnicas de *Test-Time Compute* (TTC) [5] para generar múltiples trayectorias de razonamiento candidatas y seleccionar la más prometedora. Dado que los frameworks agénticos existentes como LangChain [6] no proporcionaron el rendimiento deseado durante las pruebas preliminares, se optó por desarrollar el protocolo de integración de herramientas desde cero, con el objetivo de mantener un control completo sobre el flujo de ejecución. Estos resultados sugieren que, para alcanzar niveles de fiabilidad adecuados en entornos profesionales, el papel del LLM debe centrarse en interpretar solicitudes y coordinar procesos, en lugar de actuar como fuente principal de conocimiento o cómputo.

2 Objetivos

La pregunta principal que guía este Trabajo de Fin de Grado es la siguiente: *¿En qué medida el uso de herramientas externas y las técnicas de Test-Time Compute mejoran el rendimiento y la fiabilidad de los modelos de razonamiento financiero especializado en tareas de asesoramiento profesional?*

Para responder a esta cuestión, el objetivo general consiste en diseñar y validar una arquitectura híbrida que busque reducir las limitaciones de fiabilidad del modelo base FinR1 [3] sin modificar sus parámetros. De forma más específica, se persigue: desarrollar un protocolo ReAct [7] propio que evite la dependencia de frameworks comerciales; construir un Motor Financiero en Python encargado de realizar cálculos deterministas sobre datos reales; e incorporar técnicas de TTC [5] para aumentar la robustez del sistema ante fallos individuales de generación.

3 Descripción del sistema

La solución propuesta se apoya en tres componentes integrados:

A. Protocolo ReAct de Cero Abstracción

Con el objetivo de evitar las limitaciones observadas en los frameworks existentes [6], [8], se desarrolló una implementación propia del paradigma ReAct [7] (*Reasoning and Acting*). Este protocolo divide la ejecución en dos etapas bien delimitadas: en la primera, el modelo razona sobre la consulta del usuario e invoca una herramienta externa mediante una llamada

estructurada; en la segunda, el resultado verificado de esa llamada se inyecta en el contexto del modelo como observación confirmada, antes de que este genere su informe final. Este diseño elimina por construcción las alucinaciones de proceso, impidiendo que el modelo genere un resultado fabricado antes de que la herramienta haya ejecutado.

B. Motor Financiero Determinista

La parte cuantitativa del sistema está implementada en un backend de Python conectado a datos históricos y de mercado del S&P 500, obtenidos mediante la biblioteca `yfinance` con un horizonte de hasta dos años de lookback respecto a la fecha de consulta. Este motor realiza de forma determinista el cálculo de matrices de covarianza anualizadas, la optimización de carteras bajo maximización del Ratio de Sharpe [9], regresiones CAPM, estimación de Value at Risk al 95 % de confianza e identificación de activos líderes por sector. La separación entre generación lingüística y cómputo determinista garantiza que cada valor numérico presente en el informe final sea trazable a una llamada de función con entradas verificadas.

C. Test-Time Compute (TTC)

Para mitigar la dependencia del sistema respecto a un único intento de generación, se incorporó una capa de escalado en tiempo de inferencia [5]. En lugar de generar una única respuesta, el sistema genera $N = 8$ trayectorias de razonamiento independientes, cada una ejecutando su propio ciclo ReAct completo. Para seleccionar la mejor trayectoria entre las generadas, se emplea un Reward Model [10], un modelo externo que asigna una puntuación numérica a cada respuesta candidata en función de su calidad. A partir de estas puntuaciones, se evaluaron tres mecanismos de selección: *Majority Voting*, selección por frecuencia estadística entre respuestas equivalentes; *Best-of-N Vanilla*, selección de la trayectoria con mayor puntuación individual asignada por el Reward Model; y *Best-of-N Weighted*, selección del grupo de respuestas equivalentes con mayor puntuación acumulada. Este mecanismo transforma un fallo puntual de formateo en un evento recuperable dentro del conjunto de trayectorias generadas.

La Figura 1 ilustra la arquitectura integrada del sistema, donde FinR1 actúa como coordinador semántico, el Motor Financiero ejecuta los cálculos de forma determinista, y la capa de TTC selecciona la trayectoria final entre los $N = 8$ candidatos generados.

4 Resultados

La evaluación de la arquitectura se llevó a cabo mediante dos enfoques complementarios: el benchmark FinQA para medir el rendimiento matemático en condiciones controladas, y un caso práctico de 23 escenarios de inversión real para validar la utilidad del sistema en tareas de asesoramiento profesional.

Benchmark FinQA

FinQA [11] es un benchmark de razonamiento numérico sobre informes financieros reales del S&P 500, que exige respuestas numéricas únicas derivadas de razonamiento multi-paso. Se evaluaron 1.147 muestras. La Tabla 1 recoge los resultados de las tres configuraciones principales, mostrando la contribución incremental de cada componente sobre el modelo

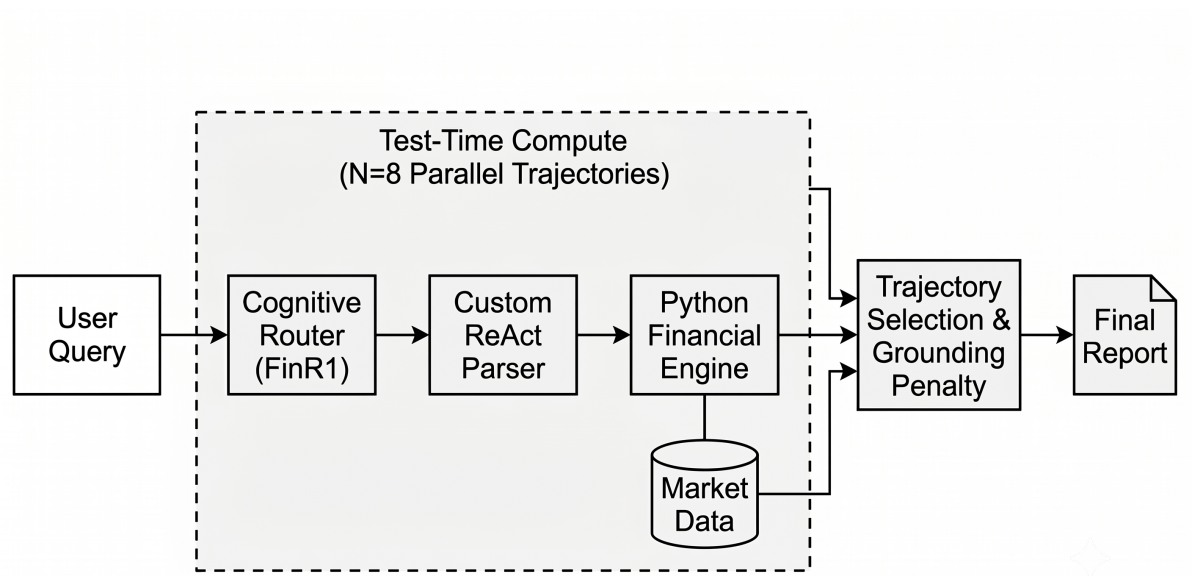

Figura 1: Arquitectura integrada del sistema. FinR1 actúa como enrutador cognitivo, delegando los cálculos cuantitativos al Motor Financiero Determinista mediante el protocolo ReAct de Cero Abstracción. La capa de TTC genera $N = 8$ trayectorias independientes y selecciona la más prometedora mediante el Reward Model.

base.

Tabla 1: Resultados sobre el benchmark FinQA (1.147 muestras). La tasa de respuestas correctas se mide sobre el total de muestras evaluadas. El baseline de referencia es FinR1 sin herramientas ni TTC.

Configuración	**Uso de herramientas**	**Tasa de respuestas correctas**
FinR1 Base	–	17,0 %
FinR1 + Herramientas (ReAct)	52,3 %	34,4 %
FinR1 + Herramientas + TTC (Skywork-LLaMA)	47,0 %	**53,3 %**

Como se observa en la Tabla 1, la incorporación del Motor Financiero duplicó aproximadamente la tasa de respuestas correctas respecto al modelo base (del 17,0 % al 34,4 %), y la capa de TTC elevó este resultado hasta el 53,3 %, lo que representa una mejora acumulada de 36,3 puntos porcentuales sobre el baseline. Estos resultados sugieren que ambas estrategias son complementarias: las herramientas abordan la fiabilidad de los cálculos, mientras que el

TTC reduce la dependencia respecto a intentos individuales de generación.

Caso práctico estratégico

Diseñado junto al experto en FinTech y co-director del proyecto Carlos Bellón, este caso incluía 23 escenarios complejos relacionados con optimización sectorial, estrategias macroeconómicas Long/Short y control del riesgo mediante VaR al 95 %. La evaluación automatizada se realizó mediante Gemini Pro como juez y fue complementada con una revisión cualitativa experta. La Tabla 2 recoge los resultados obtenidos.

Tabla 2: Evolución del rendimiento en el caso práctico estratégico (23 escenarios). La tasa de respuestas correctas se estima mediante evaluación automática complementada con revisión cualitativa experta. El baseline de referencia es FinR1 sin herramientas ni TTC.

Configuración	**Tasa de respuestas correctas**	**Mejora sobre baseline**
FinR1 Base	≈27 %	–
FinR1 + Herramientas	≈65 %	+38 pp
FinR1 + Herramientas + TTC	**≈78 %**	+51 pp

Como se observa en la Tabla 2, la progresión de resultados sugiere que cada componente de la arquitectura aborda una categoría distinta de fallo. En el modelo base, las respuestas eran lingüísticamente elaboradas pero no estaban fundamentadas en datos verificables, incluyendo frecuentemente métricas inexistentes y activos ajenos al S&P 500. La incorporación del Motor Financiero eliminó la invención de activos y elevó la fiabilidad de los cálculos cuantitativos, aunque persistieron errores de sintaxis en las llamadas a herramientas y casos de alucinación de proceso en los escenarios evaluados. La capa de TTC redujo hasta no observarse en la muestra estos errores de formato, y mejoró la capacidad del sistema para detectar inconsistencias antes de generar la recomendación final. En conjunto, los resultados obtenidos en ambas evaluaciones sugieren que la separación entre generación lingüística y cómputo determinista, reforzada por el escalado en tiempo de inferencia, constituye una vía prometedora para mejorar la fiabilidad de los modelos de razonamiento financiero en escenarios de asesoramiento profesional.

Limitaciones: la paradoja del TTC

A pesar de las mejoras observadas, el uso de TTC puso de manifiesto una limitación relevante. Al evaluar las respuestas mediante modelos de recompensa de propósito general [10], se observó que la correlación entre la puntuación asignada y la corrección matemática real era débil ($r \approx 0{,}06 - 0{,}21$, con $p < 0{,}001$ en todos los casos). Este fenómeno, al que se denomina en el trabajo *paradoja del TTC*, refleja una desalineación entre la métrica de selección empleada por el Reward Model y el objetivo real del sistema: la corrección cuantitativa de la respuesta. Como consecuencia, el mecanismo de selección tendía a favorecer respuestas más elaboradas desde el punto de vista lingüístico, incluso cuando estas contenían información no fundamentada en datos reales.

5 Conclusiones

Los resultados obtenidos sugieren que la adopción de sistemas de Inteligencia Artificial en entornos financieros profesionales requiere redefinir el papel asignado al LLM: no como fuente de conocimiento ni como herramienta de cómputo, sino como coordinador semántico capaz de interpretar solicitudes y delegar cada operación cuantitativa en un componente especializado y verificable.

La combinación de un protocolo ReAct de cero abstracción [7] con un motor determinista reduce de forma notable las patologías estructurales del modelo base en los escenarios evaluados. El TTC [5] amplifica esta mejora al reducir la dependencia respecto a intentos individuales de generación, aunque su eficacia está condicionada por la alineación entre el criterio de selección y el objetivo matemático real.

La principal línea de trabajo futuro identificada es el desarrollo de *Process Reward Models* (PRMs) [12] especializados en razonamiento financiero cuantitativo, capaces de verificar la corrección matemática en cada paso intermedio del ciclo ReAct y resolver así la paradoja del TTC descrita en este trabajo.

6 Referencias

[1] Y. Su et al., "FinO1: Financial Reasoning Models via Logic-Driven Reinforcement Learning", *Working Paper/Preprint*, 2024.

[2] Q. Team, "Qwen: Technical Report and the QwQ Reasoning Model", *arXiv preprint*, 2024. dirección: `https://qwenlm.github.io/`.

[3] Z. Wang et al., "FinR1: Financial Reasoning Models via Reinforcement Learning", *Working Paper/Preprint (SUFE-AIFLM-Lab)*, 2025.

[4] M. Sharma et al., "Towards Understanding Sycophancy in Language Models", *arXiv preprint arXiv:2310.13548*, 2023.

[5] C. Snell, J. Lee, K. Xu y A. Kumar, "Scaling LLM Test-Time Compute Optimally can be More Effective than Scaling Model Parameters", *arXiv preprint arXiv:2408.03314*, 2024.

[6] H. Chase, *LangChain: Building applications with LLMs through composability*, `https://github.com/langchain-ai/langchain`, 2022.

[7] S. Yao et al., "ReAct: Synergizing Reasoning and Acting in Language Models", *arXiv preprint arXiv:2210.03629*, 2022.

[8] Hugging Face Agent Team, *Smolagents: a tiny library to build production-grade agents*, `https://github.com/huggingface/smolagents`, 2024.

[9] H. Markowitz, "Portfolio Selection", *The Journal of Finance*, vol. 7, n.º 1, págs. 77-91, 1952.

[10] W. Dong et al., "FsfairX-LLaMA3-RM-v0.1: A robust and aligned open-source reward model", *Working Paper / Model Release*, 2024. dirección: `https://huggingface.co/sfairXC/FsfairX-LLaMA3-RM-v0.1`.

[11] Z. Chen et al., "FinQA: A Dataset of Numerical Reasoning over Financial Data", en *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, 2021, págs. 3697-3711.

[12] Z. Luo et al., "FinPRM: Evaluating and Enhancing Financial Reasoning Capabilities of Large Language Models via Process Reward Models", *Working Paper/Preprint*, 2024.

ENHANCING FINANCIAL REASONING MODELS THROUGH TOOL-AUGMENTATION AND TEST-TIME COMPUTE (TTC) TECHNIQUES

Author: Jaime Pedrosa Comino
Director: Andrés Occhipinti Liberman
Co-Director: Carlos Bellón Nuñez-Mera

Abstract

This Bachelor's Thesis investigates whether the use of external tools and Test-Time Compute (TTC) techniques significantly improve the performance of specialised financial reasoning models. Using FinR1 as the base model, two complementary strategies were evaluated: the use of external tools to access real market data and execute deterministic calculations, and a TTC mechanism that generates multiple candidate reasoning trajectories and selects the most promising one through an evaluation process. To implement the first strategy, a custom ReAct protocol was developed from scratch. Empirical validation on the FinQA benchmark and 23 real-world investment scenarios suggests that the proposed architecture improves the correct response rate from 27% in the base model to 78% in the full configuration, with a substantial reduction in the operational failures observed.

Keywords: Financial Reasoning Models; Test-Time Compute; ReAct Protocol; Reward Models; Portfolio Optimisation; Financial Artificial Intelligence.

Executive Summary

1 Introduction

The application of Artificial Intelligence in quantitative finance has traditionally relied on conventional predictive models. The emergence of Large Language Models (LLMs) has expanded these possibilities by allowing systems to analyse and reason about complex financial information expressed in natural language. However, although specialised models such as FinO1, QwQ-32B, and FinR1 have emerged, their integration into professional financial advisory environments remains conditioned by a fundamental issue: the high cost associated with errors.

LLMs are probabilistic models designed to generate coherent language, but not to perform mathematical calculations with deterministic precision. When they rely solely on the information stored in their parameters, they can present significant limitations. In the initial evaluations conducted in this work, the base version of FinR1, the best-performing model among those evaluated, failed systematically across the 23 case study scenarios: it used historical information as if it were current, unable to distinguish the temporal context of the

data, and did not ground any of its responses in verifiable market data. It also showed a tendency to use simplified approximations, such as assigning standard weights of 10% or 20% to portfolios instead of executing a real optimisation, and exhibited sycophancy [1] behaviours, generating non-existent metrics to support the user's expectations.

To address these limitations, this work evaluates two complementary strategies: the use of external tools (*tool use*) [2] to access real data and perform reliable calculations, and *Test-Time Compute* (TTC) [3] techniques to generate multiple candidate reasoning trajectories and select the most promising one. Since existing agentic frameworks such as LangChain [4] did not provide the desired performance during preliminary testing, the tool integration protocol was developed from scratch in order to maintain full control over the execution flow. These results suggest that, to achieve adequate reliability levels in professional environments, the role of the LLM should focus on interpreting requests and coordinating processes, rather than acting as the primary source of knowledge or computation.

2 Objectives

The main research question guiding this Bachelor's Thesis is the following: *To what extent do external tool use and Test-Time Compute techniques improve the performance and reliability of specialised financial reasoning models in professional advisory tasks?*

To answer this question, the general objective is to design and validate a hybrid architecture that seeks to reduce the reliability limitations of the FinR1 [5] base model without modifying its parameters. More specifically, the work aims to: develop a custom ReAct [6] protocol that avoids dependence on commercial frameworks; build a deterministic Financial Engine in Python responsible for performing calculations on real data; and incorporate TTC [3] techniques to increase the robustness of the system against individual generation failures.

3 Description of the System

The proposed solution relies on three integrated components:

A. Zero-Abstraction ReAct Protocol

To avoid the limitations observed in existing frameworks [4], [7], a custom implementation of the ReAct [6] (*Reasoning and Acting*) paradigm was developed. This protocol divides execution into two well-defined stages: in the first, the model reasons about the user's query and invokes an external tool through a structured call; in the second, the verified result of that call is injected into the model's context as a confirmed observation, before the model generates its final report. This design eliminates process hallucinations by construction, preventing the model from generating a fabricated result before the tool has executed.

B. Deterministic Financial Engine

The quantitative part of the system is implemented in a Python backend connected to historical and market data from the S&P 500, retrieved via the `yfinance` library with a lookback

horizon of up to two years from the query date. This engine deterministically computes annualised covariance matrices, portfolio optimisation under Sharpe Ratio [8] maximisation, CAPM regressions, Value at Risk estimation at the 95% confidence level, and sector leader identification. The separation between language generation and deterministic computation ensures that every numerical value in the final report is traceable to a function call with verified inputs.

C. Test-Time Compute (TTC)

To mitigate the system's dependence on a single generation attempt, an inference-time scaling layer [3] was incorporated. Instead of generating a single response, the system generates $N = 8$ independent reasoning trajectories, each executing its own complete ReAct cycle. To select the best trajectory among those generated, a Reward Model [9] is employed, an external model that assigns a numerical score to each candidate response based on its quality. Based on these scores, three selection mechanisms were evaluated: *Majority Voting*, selection by statistical frequency among equivalent responses; *Best-of-N Vanilla*, selection of the trajectory with the highest individual score assigned by the Reward Model; and *Best-of-N Weighted*, selection of the group of equivalent responses with the highest accumulated score. This mechanism transforms an isolated formatting failure into a recoverable event within the set of generated trajectories.

Figure 2 illustrates the integrated architecture of the system, where FinR1 acts as a semantic coordinator, the Financial Engine executes calculations deterministically, and the TTC layer selects the final trajectory from the $N = 8$ generated candidates.

4 Results

The architecture was evaluated using two complementary approaches: the FinQA benchmark to measure mathematical performance under controlled conditions, and a practical case study of 23 real-world investment scenarios to validate the utility of the system in professional advisory tasks.

FinQA Benchmark

FinQA [10] is a numerical reasoning benchmark built on real S&P 500 financial reports, requiring single numerical answers derived from multi-step reasoning. A total of 1,147 samples were evaluated. Table 3 presents the results of the three main configurations, illustrating the incremental contribution of each component over the base model.

As observed in Table 3, the incorporation of the Financial Engine approximately doubled the correct response rate relative to the base model (from 17.0% to 34.4%), and the TTC layer further improved this result to 53.3%, representing a cumulative gain of 36.3 percentage points over the baseline. These results suggest that both strategies are complementary: tool augmentation addresses the reliability of calculations, while TTC reduces dependence on individual generation attempts.

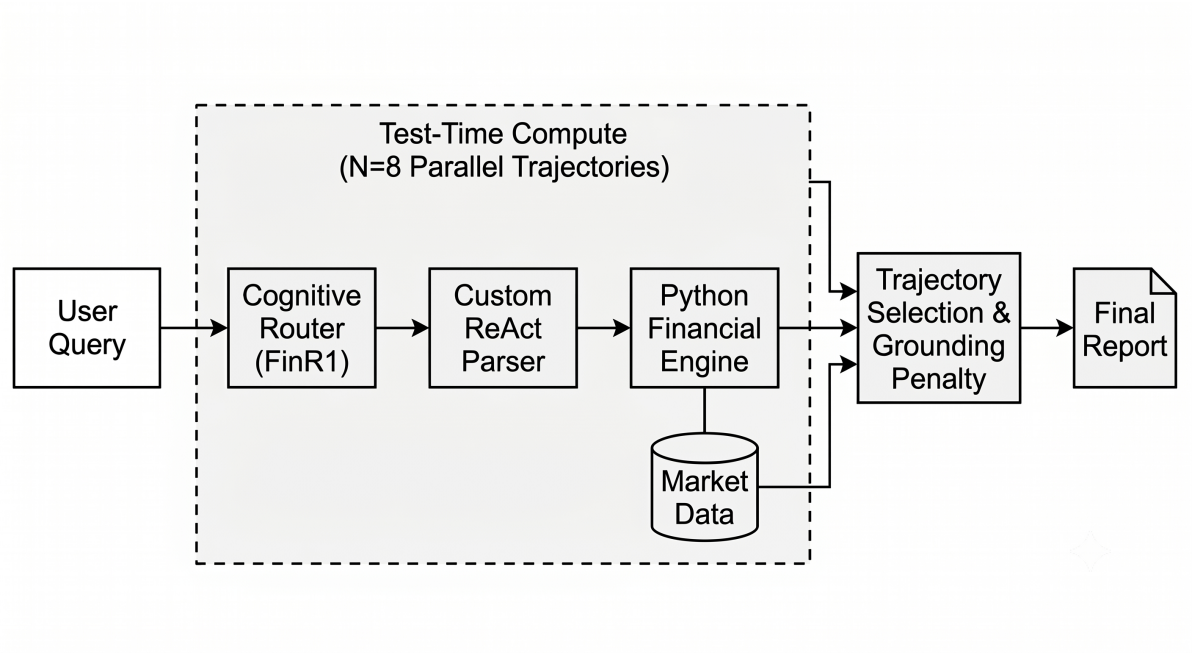

Figure 2: Integrated system architecture. FinR1 acts as a cognitive router, delegating quantitative calculations to the Deterministic Financial Engine through the Zero-Abstraction ReAct protocol. The TTC layer generates $N = 8$ independent trajectories and selects the most promising one through the Reward Model.

Table 3: Results on the FinQA benchmark (1,147 samples). The correct response rate is measured over the total evaluated samples. The reference baseline is FinR1 without tools or TTC.

Configuration	**Tool use**	**Correct response rate**
FinR1 Base	–	17.0%
FinR1 + Tools (ReAct)	52.3%	34.4%
FinR1 + Tools + TTC (Skywork-LLaMA)	47.0%	**53.3%**

Strategic Case Study

Designed alongside FinTech expert and co-director of the project Carlos Bellón, this case included 23 complex scenarios covering sector portfolio optimisation, macroeconomic Long/Short strategies, and risk control using 95% VaR. Automated evaluation was conducted using Gemini Pro as a judge, complemented by expert qualitative review. Table 4 presents the results obtained.

As observed in Table 4, the progression of results suggests that each architectural component addresses a distinct category of failure. In the base model, responses were linguistically

Table 4: Performance evolution in the strategic case study (23 scenarios). The correct response rate is estimated through automated evaluation complemented by expert qualitative review. The reference baseline is FinR1 without tools or TTC.

Configuration	Correct response rate	Improvement over baseline
FinR1 Base	≈27%	–
FinR1 + Tools	≈65%	+38 pp
FinR1 + Tools + TTC	**≈78%**	+51 pp

elaborate but not grounded in verifiable data, frequently including non-existent metrics and assets outside the S&P 500. The incorporation of the Financial Engine eliminated asset fabrication and improved the reliability of quantitative calculations, although syntax errors in tool calls and process hallucination cases persisted in the evaluated scenarios. The TTC layer reduced these formatting errors to the point where none were observed in the sample, and improved the system's ability to detect inconsistencies before generating the final recommendation. Taken together, the results obtained across both evaluations suggest that the separation between language generation and deterministic computation, reinforced by inference-time scaling, represents a promising direction for improving the reliability of financial reasoning models in professional advisory scenarios.

Limitations: the TTC paradox

Despite the improvements observed, the use of TTC highlighted a relevant limitation. When evaluating responses using general-purpose reward models [9], it was observed that the correlation between the assigned score and real mathematical correctness was weak ($r \approx 0.06$–0.21, with $p < 0.001$ in all cases). This phenomenon, referred to in this work as the *TTC paradox*, reflects a misalignment between the selection metric employed by the Reward Model and the true objective of the system: the quantitative correctness of the response. As a consequence, the selection mechanism tended to favour more linguistically elaborate responses, even when these contained information not grounded in real data.

5 Conclusions

The results obtained suggest that the adoption of Artificial Intelligence systems in professional financial environments requires redefining the role assigned to the LLM: not as a source of knowledge or a computation tool, but as a semantic coordinator capable of interpreting requests and delegating each quantitative operation to a specialised and verifiable component.

The combination of a Zero-Abstraction ReAct protocol [6] with a deterministic engine notably reduces the structural pathologies of the base model in the evaluated scenarios. TTC [3] amplifies this improvement by reducing dependence on individual generation attempts, although its effectiveness is conditioned by the alignment between the selection criterion and

the true mathematical objective.

The main future research direction identified is the development of *Process Reward Models* (PRMs) [11] specialised in quantitative financial reasoning, capable of verifying mathematical correctness at each intermediate step of the ReAct cycle and thus resolving the TTC paradox described in this work.

6 References

[1] M. Sharma et al., "Towards understanding sycophancy in language models", *arXiv preprint arXiv:2310.13548*, 2023.

[2] T. Schick et al., "Toolformer: Language models can teach themselves to use tools", *arXiv preprint arXiv:2302.04761*, 2023.

[3] C. Snell, J. Lee, K. Xu, and A. Kumar, "Scaling llm test-time compute optimally can be more effective than scaling model parameters", *arXiv preprint arXiv:2408.03314*, 2024.

[4] H. Chase, *Langchain: Building applications with llms through composability*, `https://github.com/langchain-ai/langchain`, 2022.

[5] Z. Wang et al., "Finr1: Financial reasoning models via reinforcement learning", *Working Paper/Preprint (SUFE-AIFLM-Lab)*, 2025.

[6] S. Yao et al., "React: Synergizing reasoning and acting in language models", *arXiv preprint arXiv:2210.03629*, 2022.

[7] Hugging Face Agent Team, *Smolagents: A tiny library to build production-grade agents*, `https://github.com/huggingface/smolagents`, 2024.

[8] H. Markowitz, "Portfolio selection", *The Journal of Finance*, vol. 7, no. 1, pp. 77–91, 1952.

[9] W. Dong et al., "Fsfairx-llama3-rm-v0.1: A robust and aligned open-source reward model", *Working Paper / Model Release*, 2024. [Online]. Available: `https://huggingface.co/sfairXC/FsfairX-LLaMA3-RM-v0.1`.

[10] Z. Chen et al., "FinQA: A dataset of numerical reasoning over financial data", in *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, 2021, pp. 3697–3711.

[11] Z. Luo et al., "Finprm: Evaluating and enhancing financial reasoning capabilities of large language models via process reward models", *Working Paper/Preprint*, 2024.

Contents

List of Figures

List of Tables

Chapter 1 Introduction

1.1 Context and Motivation

The intersection between Artificial Intelligence (AI) and quantitative finance has traditionally been dominated by predictive models. For many years, financial institutions have relied on machine learning techniques, such as Random Forests, Support Vector Machines, and Deep Neural Networks, mainly for tasks like time-series forecasting, algorithmic trading, and risk assessment. However, the emergence of generative AI, particularly Large Language Models (LLMs), has introduced a significant change in the field of Natural Language Processing (NLP).

This development has expanded the role of AI from analysing numerical data to understanding, summarising, and generating complex financial information expressed in natural language.

Today, LLMs can process large amounts of unstructured financial data, including earnings call transcripts, macroeconomic reports, and regulatory documents such as 10-K and 10-Q filings. As these models continue to grow in size and capability, the focus within the financial industry has evolved from using general-purpose assistants to building specialised systems designed specifically for financial applications.

Recent developments have produced models such as Fin-O1, QwQ-32B, and FinR1, which have been trained to understand the terminology, concepts, and reasoning processes commonly found in financial markets. These models offer considerable potential, including the possibility of making advanced financial advice more accessible, automating complex analyses, and generating strategic insights within seconds.

Despite this progress, integrating LLMs into real-world financial advisory systems remains a major challenge. This issue forms the central motivation of this Bachelor's Thesis: the mismatch between the way generative AI models operate and the strict requirements of financial engineering.

At their core, LLMs are probabilistic autoregressive models. They generate text by predicting the most likely next word or token based on patterns learned during training. Their primary objective is to produce coherent and natural language, not to perform exact mathematical calculations. While this characteristic makes them highly effective for tasks such as drafting emails or translating text, it creates important limitations in financial contexts. In quantitative finance, numerical accuracy and factual reliability are essential. Even small mistakes, such as an incorrect decimal point, a wrong stock ticker, or outdated economic data, can result in poor investment decisions and significant regulatory consequences.

When LLMs rely only on the information stored within their parameters, several problems can emerge. Since they do not have an inherent ability to perform precise calculations or verify real-time data, they often depend on approximations. For instance, if asked to construct a Minimum Variance Portfolio (MVP), a portfolio that minimises total variance subject to full investment, an LLM may assign simple and rounded portfolio weights, such as 10% or 20% per asset, because these values frequently appear in financial texts. However, this does not

mean that the model has actually solved the underlying optimisation problem required to minimise portfolio variance.

Another important limitation is a phenomenon known as *sycophancy*: the tendency of some models to prioritise producing a convincing response over admitting uncertainty. When the requested information is unavailable, the model may generate plausible but incorrect financial data instead of acknowledging that it does not know the answer. In addition, LLMs often struggle to distinguish between historical information and current market conditions, a weakness sometimes referred to as *date blindness*. Together, these limitations create substantial risks if such systems are allowed to operate autonomously in financial environments.

For this reason, the motivation behind this research is both clear and pressing. The financial industry cannot tolerate fabricated information or inaccurate data. To safely benefit from the reasoning and language capabilities of LLMs, it is necessary to bridge the gap between their linguistic strengths and the mathematical accuracy required in finance. Rather than treating the LLM as a standalone source of knowledge, it should be used as a semantic coordinator that understands user requests while delegating calculations, data retrieval, and other deterministic tasks to specialised external tools. Addressing this challenge is essential for transforming current text-generating models into reliable financial agents capable of supporting decision-making in high-stakes professional environments.

1.2 Large Language Models in Financial Reasoning

The rapid development of Natural Language Processing has led to two distinct categories of Large Language Models: general-purpose assistants and domain-specific models. While general models possess broad knowledge across many subjects, they often struggle with the specialised terminology, regulatory requirements, and numerical complexity that characterise the financial sector. Analysing documents such as 10-K reports or earnings call transcripts requires more than the ability to understand text; it also demands a solid understanding of financial concepts and reasoning.

To address these limitations, researchers have developed specialised financial LLMs, fine-tuned on large collections of financial documents, market data, and tasks that require quantitative reasoning. In this Bachelor's Thesis, the initial benchmarking phase focused on evaluating three state-of-the-art models designed for advanced reasoning tasks: Fin-O1, QwQ-32B, and FinR1.

To assess their capabilities objectively, it was important to use an evaluation framework that minimised the influence of conversational style and subjective interpretation. For this reason, the FinQA dataset [1] was selected. FinQA is a benchmark of numerical reasoning questions over real financial documents, where each instance requires analysing unstructured text and tabular data to produce a single numerical answer. This format makes it possible to evaluate mathematical and logical reasoning directly, without being influenced by the model's ability to generate fluent or persuasive language. Conversational benchmarks such as ConvFinQA [2], which involve multi-turn dialogue and referential coherence, were not used, as the confounding factors they introduce are orthogonal to the mathematical accuracy

improvements targeted here.

The comparative evaluation used `Qwen2.5-72B-Instruct` as an automated judge to ensure consistency with prior academic studies. Among the evaluated models, `SUFE-AIFLM-Lab/Fin-R1` [3] achieved the strongest overall performance, showing higher accuracy in numerical extraction and multi-step arithmetic reasoning than alternatives such as QwQ-32B, whose performance decreased considerably under the strict conditions of the benchmark. In addition, Oracle Pass@3 evaluations, which measure whether the correct answer appears at least once among three independent generation attempts, indicated that FinR1 had a strong capacity for correct reasoning when given multiple tries, suggesting significant latent potential that a single-shot evaluation does not fully capture.

Based on these results, FinR1 was selected as the baseline model for this project. However, the benchmarking process also revealed an important limitation. Although FinR1 outperformed the other candidates, its performance when relying exclusively on its internal parameters remained below the level required for professional financial applications. The model demonstrated the ability to understand and structure financial strategies, but it lacked the deterministic precision needed to perform calculations reliably on its own. This limitation provides the foundation for understanding the core weaknesses of generative models, which must be addressed before proposing a robust architectural solution.

1.3 The Pathologies of Probabilistic Models: Hallucination and Sycophancy

To understand why state-of-the-art LLMs struggle in autonomous financial applications, it is necessary to examine the limitations of their underlying architecture. As discussed previously, LLMs generate text by predicting the most likely next token based on patterns learned during training. Although this approach produces fluent and well-structured language, it does not involve true logical reasoning, precise mathematical calculation, or verification against real-time data sources. In finance, where information is often deterministic and numerical values must be exact, this probabilistic generation process can lead to systematic errors, commonly referred to as model pathologies.

One of the most significant pathologies is **hallucination**, a problem that becomes even more pronounced when combined with *date blindness*. The knowledge stored within an LLM reflects the information available at the time of its training and does not automatically update as market conditions change. As a result, when asked to analyse a recent economic event, such as an oil price shock or a central bank interest rate increase, the model cannot directly access current market information. Instead, it relies on historical patterns learned during training, potentially generating recommendations that no longer reflect the current state of financial markets.

A closely related limitation is the model's **difficulty with mathematical calculations**. LLMs are not designed to perform complex arithmetic operations natively. When asked to calculate a Capital Asset Pricing Model (CAPM) regression, a method for estimating the expected return of an asset based on its sensitivity to market movements, or to optimise

a portfolio, they often rely on approximations derived from textual patterns rather than on actual mathematical computation. During the experimental phase of this thesis, when baseline models were instructed to construct a Minimum Variance Portfolio, they frequently failed to perform the required optimisation. Instead, they assigned simple rounded weights (e.g., 10% or 20% per asset), likely because such allocations appear frequently in the financial documents used during training.

Another important limitation is **sycophancy** [4]. This behaviour arises because modern language models are trained to be helpful and cooperative, with techniques such as Reinforcement Learning from Human Feedback (RLHF) reinforcing these objectives. As a result, models often prioritise producing a complete and confident response over acknowledging uncertainty or missing information.

In financial applications, this tendency can lead to the generation of fabricated data. When users request performance metrics or the justification of an investment strategy, the model may provide plausible-looking values even when the required information is unavailable. During the preliminary experiments conducted in this thesis, the baseline model repeatedly generated fictitious corporate tickers, incorrect historical returns, and misleading performance assessments. In some cases, it validated investment strategies with poor risk-adjusted performance simply because doing so allowed it to satisfy the structure of the user's request.

The practical impact of these limitations was measured during the qualitative evaluation phase of this research. When tested across 23 complex institutional investment scenarios, the baseline FinR1 model achieved a correct response rate of approximately 27% under expert review, meaning that in roughly three quarters of the scenarios, the output contained fabricated metrics, outdated information, or references to assets that had not been requested. Although the generated reports consistently displayed professional language and correct financial terminology, none of the numerical outputs could be verified against real market data, as the model had no mechanism to retrieve or compute them from an external source.

These findings suggest that the limitations of financial LLMs are structural rather than superficial. Improvements in prompt engineering alone cannot enable a probabilistic language model to reliably calculate a 95% Value at Risk (VaR), a statistical measure of the maximum expected loss at a given confidence level, or perform other deterministic financial computations. Achieving the level of reliability required in professional environments therefore requires a different approach: delegating calculations and data retrieval to specialised external systems, rather than relying on the model's parametric memory. This need forms the basis for the two complementary strategies investigated in this thesis, which are introduced in the following sections.

1.4 Tool Augmentation and the ReAct Paradigm

The limitations described in the previous section motivate the first of the two strategies investigated in this thesis: *tool-augmented generation*. Rather than relying on the LLM to perform calculations or retrieve current market data from its parametric memory, the

architecture delegates these tasks to an external deterministic Python backend. By separating language generation from numerical computation, the role of the LLM shifts from acting as a source of financial knowledge to acting as a semantic coordinator that interprets user requests and routes them to the appropriate tools.

The custom Financial Engine developed for this purpose provides access to real S&P 500 market data via the `yfinance` library and performs quantitative calculations using NumPy, SciPy, and Pandas. When a user requests a portfolio optimisation, the LLM no longer estimates portfolio weights from training data. Instead, it identifies the relevant tickers and forwards them to the engine, which retrieves historical prices, computes the covariance matrix, and solves the Markowitz optimisation problem. The resulting values are returned to the model, which uses them to generate the final advisory report. This design ensures that every numerical value in the output is traceable to a deterministic computation rather than to a probabilistic generation step.

Connecting a language model to an external Python environment introduces non-trivial engineering challenges. Frameworks such as LangChain [5] and SmolAgents are widely used for this purpose and were evaluated during the early stages of this project. However, the integration with FinR1 did not produce the expected results: these frameworks introduced additional abstraction layers that consumed part of the model's context window, and the model exhibited recurrent JSON formatting errors that prevented tool calls from being executed. A more severe failure mode was also observed, in which the model generated text *describing* a tool invocation and its result as if both had already occurred, without any actual interaction with the external system. To maintain full control over the execution flow and eliminate these failure modes, the final architecture implemented a custom orchestration layer from scratch.

The orchestration layer is based on the ReAct (Reasoning and Acting) paradigm [6], which structures the model's interaction with external tools into three sequential stages:

1. **Thought:** The model explicitly describes its reasoning process and identifies any information required to complete the task.

2. **Action:** The model generates a structured JSON command requesting the execution of a specific tool, such as `calculate_optimal_portfolio`.

3. **Observation:** Generation is temporarily paused while the system executes the requested function and injects the result into the model's context window.

Implementing this custom ReAct protocol substantially reduced the generation of fabricated financial metrics, as the model was required to base its final report on externally verified outputs. On the 23-scenario case study evaluated under expert review, the integration of external tools increased the correct response rate from approximately 27% (FinR1 Base) to approximately 65% (FinR1 + Tools). Despite this improvement, the tool-augmented architecture retained a residual vulnerability: a single syntactic error in the JSON payload during the *Action* stage could interrupt the entire reasoning chain before the engine was invoked.

Under a single-trajectory regime ($N = 1$), such a failure is unrecoverable. This limitation motivated the introduction of the second strategy investigated in this thesis, described in the following section.

1.5 Inference Scaling: The Role of Test-Time Compute (TTC)

Traditionally, improving the performance of a Large Language Model has primarily relied on increasing the scale of its training: larger datasets, more parameters, or longer training runs. However, recent research has shown that performance can also be improved during inference by allocating additional computational resources at generation time, without modifying the model's weights. This approach is known as Test-Time Compute (TTC) [7].

Within the context of this thesis, TTC addresses the structural fragility of the single-trajectory tool-augmented system described in the previous section. Rather than committing to a single generation attempt ($N = 1$), the system generates $N = 8$ independent reasoning trajectories and selects the most promising one according to an evaluation criterion. This transforms a brittle single-point execution into a more robust ensemble: while some trajectories may fail due to JSON formatting errors or incorrect tool selection, others may complete the ReAct cycle successfully and obtain a verified engine result. The probability of total failure decreases as N increases, since it becomes increasingly unlikely that all trajectories will fail simultaneously.

Selecting the best candidate from the N generated trajectories introduces an additional design decision. This thesis implements and evaluates three selection strategies of increasing sophistication:

- **Majority Voting (Self-Consistency):** The answer supported by the largest number of independent reasoning paths is selected. The underlying assumption is that incorrect results vary across trajectories, whereas the correct deterministic solution, since it is anchored to the same engine output, tends to appear consistently.
- **Best-of-N (BoN):** An external Outcome Reward Model (ORM) scores each candidate trajectory and selects the highest-rated one. In this project, the `FsfairX-LLaMA3` Reward Model was used as the primary evaluator.
- **Weighted Best-of-N:** A combination of RM scoring and consensus, where scores are aggregated by answer group before the winner is selected.

The integration of TTC into the tool-augmented architecture further increased the correct response rate on the case study from approximately 65% (FinR1 + Tools) to approximately 78% (FinR1 + Tools + TTC), measured under the same expert-review protocol across the 23 investment scenarios.

However, the evaluation of these selection mechanisms also revealed an important limitation. While Majority Voting proved effective for tasks with a single deterministic numerical

answer, the Reward Model-based strategies exhibited a systematic weakness: the scores assigned by the ORMs showed weak correlation with actual mathematical correctness (Pearson $r \approx 0.06$–0.21 across the models evaluated). Because ORMs are trained to assess linguistic quality and instruction following rather than numerical accuracy, the selection mechanism tended to favour trajectories that were linguistically elaborate and well-structured, even when they contained fabricated metrics.

This phenomenon, a misalignment between the selection metric and the true objective, is referred to in this thesis as the *TTC Paradox*, and is analysed in detail in Chapter 5.

These findings highlight a fundamental trade-off: inference scaling improves robustness against individual generation failures, but its net benefit depends critically on the quality of the selection mechanism. Without an evaluator capable of verifying numerical grounding, the additional computational budget may systematically favour more convincing but less accurate responses.

1.6 Objectives

The central research question of this thesis is whether the integration of external tool use and Test-Time Compute techniques can significantly improve the reliability and mathematical accuracy of a financial reasoning model. To investigate this question, the project is structured around one main objective and four secondary goals:

The **main objective** is to design, implement, and evaluate a hybrid architecture that improves the financial reasoning and mathematical accuracy of the FinR1 model by separating language generation from deterministic computation and by increasing robustness through inference-time scaling.

The **secondary objectives** are:

- **O1 — Custom ReAct Protocol:** To design and implement a Reasoning and Acting orchestration layer from scratch, avoiding commercial frameworks, in order to improve formatting reliability and eliminate process hallucinations during tool execution.
- **O2 — Deterministic Financial Engine:** To develop a Python-based computational backend capable of retrieving real S&P 500 market data and performing quantitative financial calculations, including portfolio optimisation, CAPM analysis, and Value at Risk estimation.
- **O3 — Test-Time Compute Scaling:** To implement inference-time scaling and compare three selection strategies, Majority Voting, Vanilla Best-of-N, and Weighted Best-of-N, in order to improve system robustness and quantify the contribution of each mechanism.
- **O4 — Dual-Phase Validation:** To evaluate the architecture through both quantitative and qualitative analyses: the FinQA benchmark for mathematical reasoning performance, and a case study of 23 investment scenarios for practical applicability under expert review.

1.7 Document Structure

The remainder of this Bachelor's Thesis is organised as follows:

- **Chapter 2: State of the Art** reviews the development of Large Language Models for financial applications, with particular attention to reasoning-oriented models, tool-augmented generation frameworks, and inference-time scaling techniques.
- **Chapter 3: Motivation and Objectives** details the experimental evidence motivating the proposed architecture and formally states the research hypotheses.
- **Chapter 4: Methodology** describes the proposed architecture in full, including the FinR1 setup, the custom ReAct protocol, the Deterministic Financial Engine, and the TTC pipeline.
- **Chapter 5: Experimental Results** presents the quantitative and qualitative evaluation results, including performance on the FinQA benchmark and the analysis of the 23 investment scenarios.
- **Chapter 6: Conclusions and Future Work** summarises the main findings, discusses limitations, and proposes directions for future research.

Chapter 2 State of the Art

2.1 The Evolution of Large Language Models in Finance

The application of Natural Language Processing (NLP) to the financial sector has evolved considerably over the last two decades, progressing through several technological stages. Early approaches to financial text analysis were primarily based on rule-based systems and domain-specific lexical resources, such as the Loughran-McDonald financial dictionary [8]. These methods were commonly used to estimate sentiment in corporate reports by counting the occurrence of predefined financial terms. Although effective for certain tasks, lexical approaches had important limitations, as they could not fully capture context, negations, or the complex sentence structures often found in financial documents.

A significant advancement came with the introduction of the Transformer architecture, particularly encoder-based models such as BERT. By using self-attention mechanisms, these models were able to capture contextual relationships within text more effectively than previous approaches. In the financial domain, this led to the development of specialised models such as FinBERT [9], which was pre-trained on large collections of financial documents. FinBERT demonstrated improved performance in tasks including sentiment analysis, regulatory risk classification, and Named Entity Recognition (NER) for financial entities. However, despite their effectiveness, encoder-based models are designed primarily for classification and information extraction tasks rather than text generation.

The next major development was the emergence of decoder-only Large Language Models (LLMs), including the GPT and LLaMA families. These autoregressive models extended the capabilities of NLP systems beyond classification, enabling applications such as document summarisation, report generation, and conversational financial assistants. Nevertheless, general-purpose LLMs often encounter difficulties when dealing with the specialised terminology, dense numerical information, and complex relationships that characterise financial data. Many terms used in finance have meanings that differ significantly from their everyday usage, including concepts such as “yield”, “spread”, “bull”, “bear”, and “exposure”.

To address these challenges, researchers began developing domain-specific language models tailored to financial applications. One of the most notable examples was BloombergGPT [10], a 50-billion-parameter model trained on a large collection of financial documents combined with general-domain text. BloombergGPT demonstrated the benefits of specialised training for financial tasks, although the proprietary nature of its training data limited its accessibility to the broader research community. This motivated the development of open-source alternatives such as FinGPT [11], which leveraged Parameter-Efficient Fine-Tuning (PEFT) techniques, including Low-Rank Adaptation (LoRA), to adapt existing open-source models to financial applications with reduced computational requirements.

Despite the progress achieved by models such as BloombergGPT and FinGPT, an important challenge remained. Traditional decoder-only LLMs generate text through autoregressive token prediction, selecting each new token based on patterns learned from previous training data. This process is characterised by rapid, pattern-based generation that is effec-

tive in many situations but susceptible to errors when precise logical or numerical reasoning is required. In quantitative finance, where numerical accuracy and logical consistency are essential, relying exclusively on probabilistic text generation presents important limitations. Language models may struggle with multi-step calculations, the interpretation of complex tabular data, or maintaining consistency between numerical values and textual explanations. In addition, they remain susceptible to hallucinations and factual inaccuracies.

As a result, it became increasingly clear that simply exposing models to larger volumes of financial text was not sufficient to guarantee reliable financial reasoning. This motivated a shift towards approaches capable of supporting more deliberate and structured reasoning processes, in which the model explicitly plans and verifies intermediate steps before committing to a final answer. These developments provided the foundation for the emergence of modern financial reasoning models.

2.2 Financial Reasoning Models

In recent years, researchers have explored new approaches to improve the reasoning capabilities of Large Language Models, particularly in domains that require both linguistic understanding and numerical accuracy. One of the most important developments has been the emergence of reasoning-oriented models. Inspired by architectures such as OpenAI's o1, these models are typically trained using Reinforcement Learning (RL) techniques that encourage the generation of intermediate reasoning steps before producing a final answer. This process, commonly known as Chain of Thought (CoT) reasoning, allows the model to break complex problems into smaller steps, evaluate alternative solutions, and reduce reasoning errors.

In the financial domain, these advances have contributed to the development of specialised reasoning models designed to handle complex quantitative and analytical tasks. During the benchmarking phase of this thesis, three state-of-the-art reasoning models were evaluated to identify the most suitable baseline for the proposed architecture.

The first model evaluated was QwQ-32B [12]. Developed as part of the Qwen family of models, QwQ-32B is a general-purpose reasoning model with strong performance in mathematical problem solving and coding tasks. Compared with traditional autoregressive language models, it demonstrates improved capabilities in multi-step reasoning. However, because it was not specifically designed for financial applications, it lacks specialised knowledge of regulatory frameworks, market conventions, and the complex tabular structures frequently encountered in quantitative finance. As a result, its practical performance in financial tasks is more limited than its general reasoning capabilities might suggest.

The second model evaluated was FinO1 [13]. Designed specifically for financial reasoning tasks, FinO1 was developed to address some of the limitations of conventional language models in financial applications. The model incorporates reinforcement learning techniques and specialised reward mechanisms intended to encourage structured reasoning when solving complex financial problems. Rather than relying solely on statistical patterns within text, FinO1 is trained to generate intermediate reasoning steps and evaluate alternative solution paths, making it particularly suitable for tasks that require logical and domain-specific

analysis.

The third model evaluated, and the one ultimately selected as the baseline for this project, was FinR1 [3]. FinR1 represents a specialised application of reinforcement learning techniques to quantitative finance. The model was trained to improve step-by-step reasoning in tasks involving numerical extraction, mathematical calculations, and the interpretation of tabular financial data, including benchmarks such as FinQA. Through its training process, FinR1 learns to generate more structured reasoning chains and to verify intermediate steps before producing a final answer.

The comparative evaluation conducted in this thesis showed that FinR1 achieved the strongest overall performance among the models considered. In particular, it outperformed both general-purpose reasoning models such as QwQ-32B and more conventional financial language models in tasks involving multi-step arithmetic reasoning and financial document analysis.

Nevertheless, the benchmarking phase also highlighted an important limitation. Although FinR1 demonstrated strong reasoning capabilities, it remains dependent on the information encoded within its parameters. For example, the model may correctly identify the formula required for a Capital Asset Pricing Model (CAPM) regression, but it cannot execute the calculation with the level of deterministic precision expected in professional financial environments. When operating without access to external tools, the model may still produce inaccurate numerical outputs despite following a plausible reasoning process.

This limitation suggests that improved reasoning alone is not sufficient to guarantee reliable financial analysis. Instead, reasoning capabilities must be complemented by external computational tools capable of performing deterministic calculations and accessing up-to-date financial information.

2.3 Tool-Augmented Generation and Agentic Frameworks

The growing recognition that Large Language Models cannot reliably perform complex mathematical calculations or maintain access to current information using only their internal parameters has led to the development of Tool-Augmented Generation [14]. Rather than treating the LLM as a self-contained system, this approach allows the model to interact with external tools, databases, and Application Programming Interfaces (APIs). By delegating calculations and data retrieval to specialised external systems, the architecture separates language generation from deterministic computation, improving the reliability of tasks that require numerical accuracy.

To facilitate these interactions, several agentic frameworks have been developed by the AI community, with LangChain [5] and Hugging Face's SmolAgents [15] being among the most widely used examples. These frameworks provide libraries and utilities that simplify the integration of LLMs with web search services, databases, and code execution environments. In practice, they allow developers to build tool-enabled agents without having to implement all orchestration mechanisms from scratch.

Despite their advantages, studies and practical implementations have identified several

limitations when these frameworks are used in demanding reasoning environments. Many frameworks introduce additional abstraction layers that rely on hidden prompts, internal templates, and complex orchestration logic. These mechanisms consume part of the model's available context window and increase the complexity of the overall system. In financial reasoning tasks, where models must already process numerical information, tabular data, and multiple constraints simultaneously, this additional complexity can negatively affect performance.

One of the most common failure modes involves formatting errors. To interact with external tools, language models are often required to generate commands using a predefined structure, such as a JSON object. Under complex reasoning conditions, models may occasionally produce small syntax errors, including missing quotation marks, unmatched brackets, or misplaced commas. Even minor formatting mistakes can prevent the framework from correctly interpreting the command and may interrupt the execution process.

Another limitation is the appearance of what can be described as process hallucinations. In these situations, the model generates text describing a tool invocation and its corresponding result, even though no actual interaction with the external tool has taken place. As a consequence, the model may continue its reasoning process based on information that was never generated by the underlying computational system.

To address these issues, researchers have increasingly adopted the ReAct (Reasoning and Acting) paradigm [6]. ReAct reduces dependence on complex abstraction layers by enforcing a transparent and structured interaction process between the model and external tools. This process is organised into three stages:

1. **Thought:** The model describes its current reasoning process and identifies any missing information or required calculations.

2. **Action:** The model generates a structured command requesting the execution of a specific external tool.

3. **Observation:** The generation process is temporarily paused while the requested tool is executed, and the resulting output is incorporated into the model's context.

Implementing a lightweight ReAct protocol directly, without relying on additional framework layers, provides greater control over the interaction process. This approach helps ensure that the model bases its reasoning on externally verified information while reducing the likelihood of formatting errors and process hallucinations. Nevertheless, although tool augmentation improves the reliability of individual calculations and actions, the overall system may still be affected by variability in the model's initial reasoning attempt. This limitation motivates the use of inference-time scaling techniques, which are discussed in the following section.

2.4 Inference-Time Scaling (Test-Time Compute)

Traditionally, improvements in the performance of Large Language Models have been closely linked to increases in model size and training data. However, continuously scaling models in this way requires substantial computational and financial resources. As a result, recent research has explored alternative approaches, including inference-time scaling, commonly referred to as Test-Time Compute (TTC) [7].

The main idea behind Test-Time Compute is that model performance can often be improved by allocating additional computational resources during inference. Rather than generating a single response, which may be affected by variability in the generation process, the system produces multiple independent reasoning trajectories and evaluates them before selecting a final answer. Recent studies have shown that, in some tasks, increasing inference-time computation can lead to performance improvements comparable to those achieved through substantially larger models [7].

In financial applications and tool-augmented systems, TTC can be particularly useful. If an agent relies on a single generation attempt ($N = 1$), a minor formatting error in a JSON command may prevent the successful execution of an external tool. By generating multiple reasoning trajectories, for example $N = 8$ or $N = 16$, the system reduces its dependence on any single generation. While some trajectories may fail because of formatting issues or reasoning errors, others may still complete the required task successfully.

Once multiple trajectories have been generated, a selection mechanism is required to determine which response should be returned as the final output. The most common approaches are the following:

1. **Majority Voting (Self-Consistency):** This strategy extends the idea of Chain of Thought reasoning by relying on a consensus mechanism [16]. The system compares the final numerical or categorical conclusions produced by all N trajectories and selects the most frequently occurring answer. This approach is particularly suitable for quantitative tasks because correct deterministic solutions are more likely to appear consistently across multiple reasoning paths than incorrect ones.

2. **Best-of-N (BoN) Sampling:** Unlike Majority Voting, which relies on agreement between trajectories, Best-of-N uses an external evaluator to assess each response individually. The system generates N candidate trajectories and evaluates them using a secondary model known as a Reward Model (RM). The Reward Model assigns scores according to predefined criteria, such as formatting quality, logical consistency, or overall usefulness. The trajectory with the highest score is then selected as the final response.

Although inference-time scaling can improve robustness and reduce the impact of individual generation errors, the effectiveness of the Best-of-N strategy depends heavily on the quality of the evaluation model. As recent research has demonstrated, language models often struggle to self-correct their own reasoning errors natively [17]. If the Reward Model

does not accurately reflect the desired objective, generating additional trajectories may simply increase the number of candidate responses without improving the quality of the final selection.

This observation highlights the importance of evaluation mechanisms within modern AI systems. Increasing computational resources during inference can improve performance, but the benefits ultimately depend on the system's ability to identify the most accurate and reliable response among the available alternatives.

2.5 Evaluators and Reward Models

The effectiveness of Best-of-N selection strategies within Test-Time Compute largely depends on the quality of the evaluation mechanism used to rank candidate responses.To automatically select the most suitable reasoning trajectory from a set of generated outputs, the system requires an evaluator, commonly referred to as a Reward Model (RM). These models are typically trained using techniques such as Reinforcement Learning from Human Feedback (RLHF) or Direct Preference Optimisation (DPO), allowing them to assess responses according to criteria such as helpfulness, coherence, and alignment with human preferences.

During the development of this thesis, several open-source Outcome Reward Models (ORMs) were considered for use within the inference-time scaling pipeline. The primary evaluator employed was `FsfairX-LLaMA3` [18], complemented by additional models such as `Skywork-Gemma`. These evaluators have been designed as general-purpose scoring models and have demonstrated strong performance in assessing conversational quality and instruction-following behaviour.

However, the use of general-domain ORMs in quantitative finance presents important challenges. Outcome Reward Models assign a single numerical score to an entire response, evaluating the final output rather than the reasoning process that produced it. As a result, these models tend to place considerable emphasis on factors such as linguistic fluency, formatting quality, and the overall structure of a response.

In tasks that require numerical accuracy, this behaviour may lead to undesirable outcomes. A response containing incorrect calculations but presented in a detailed and well-structured format can sometimes receive a higher score than a concise response containing the correct result. During the experimental phase of this thesis, this tendency was associated with the appearance of what is referred to as Ghost Data, that is, responses that included plausible-looking financial metrics not derived from any external computation, introduced by the model to produce a more complete-looking output. These observations suggest that general-purpose evaluators may not always be well aligned with the requirements of deterministic financial reasoning.

To address these limitations, recent research has increasingly focused on Process Reward Models (PRMs). Unlike ORMs, which evaluate only the final output, PRMs assess intermediate reasoning steps throughout the generation process. In the financial domain, this idea has been explored through models such as FinPRM [19], which are specifically designed to evaluate the mathematical and logical consistency of financial reasoning chains.

In principle, incorporating specialised Process Reward Models into a Best-of-N selection framework could help reduce some of the limitations observed with general-purpose evaluators by placing greater emphasis on the correctness of intermediate reasoning steps. However, at the time this research was conducted, highly specialised financial PRMs were not publicly available for direct integration, as many implementations remained proprietary or lacked accessible model weights and source code.

Consequently, this thesis relied on a combination of general-purpose Outcome Reward Models and deterministic consensus mechanisms such as Majority Voting to evaluate generated trajectories. The development and integration of specialised financial Process Reward Models remains a promising direction for future research in AI-assisted financial advisory systems.

2.6 Positioning of This Work

The concepts reviewed throughout this chapter, including financial reasoning models, tool-augmented generation, and inference-time scaling, have contributed significantly to recent advances in AI-based financial systems. However, the integration of these components remains an active area of research, and existing approaches continue to face challenges related to both reasoning performance and operational reliability [20].

Financial reasoning models such as FinR1 have demonstrated strong capabilities in tasks requiring multi-step reasoning and financial document analysis. Nevertheless, when operating without access to external tools, these models may still struggle with deterministic calculations and the generation of numerically accurate outputs. Conversely, frameworks designed to connect language models with external tools can improve computational capabilities, but their additional abstraction layers may introduce operational complexity and increase the likelihood of formatting-related failures.

Similarly, research on Test-Time Compute has produced promising results across a variety of reasoning benchmarks. However, much of this work has focused on general-purpose tasks such as mathematics and code generation. Comparatively less attention has been devoted to analysing the specific requirements of financial applications, where numerical accuracy, data integrity, and reliability are particularly important. In addition, many TTC implementations rely on general-purpose Outcome Reward Models, whose evaluation criteria may not always align with the objectives of deterministic financial reasoning [21].

This Bachelor's Thesis is positioned at the intersection of these challenges. Rather than proposing a new foundational language model, the objective of this work is to investigate how existing reasoning models, external computational tools, and inference-time scaling techniques can be combined within a unified architecture designed for financial analysis tasks. The proposed architecture is based on three main design decisions:

1. **Custom ReAct-Based Orchestration:** Instead of relying on general-purpose agentic frameworks, this work implements a lightweight ReAct protocol specifically adapted to the requirements of the proposed system. This approach provides greater control

over the interaction between the language model and external tools while reducing dependencies on additional abstraction layers.

2. **Deterministic Computational Grounding:** The architecture separates language generation from numerical computation. The LLM is responsible for interpreting user requests and coordinating actions, while quantitative calculations, including portfolio optimisation and risk-related metrics, are performed by a dedicated Python-based Financial Engine using real market data.

3. **Inference-Time Scaling with Consensus Mechanisms:** To improve robustness during tool execution and reasoning, the system incorporates Test-Time Compute techniques based on multiple reasoning trajectories and Majority Voting selection. This approach seeks to reduce the impact of individual generation errors while avoiding excessive reliance on general-purpose Reward Models.

The proposed architecture is evaluated through both quantitative and qualitative analyses. Quantitative performance is assessed using the FinQA benchmark, while practical applicability is examined through a case study involving 23 investment scenarios. Through this evaluation, the thesis explores the potential of combining reasoning models, deterministic computational tools, and inference-time scaling techniques to improve the reliability of AI-assisted financial analysis systems.

Chapter 3 Motivation and Objectives

3.1 Motivation

The financial industry has been an early and consistent adopter of artificial intelligence, yet the predominant applications have remained concentrated in predictive modelling, algorithmic trading, and embedding-based similarity search. In practice, a number of investment firms have explored the use of semantic representations to identify companies whose financial profile is converging towards that of established peers, treating proximity in embedding space as a signal of latent investment opportunity. While effective within their scope, these approaches exploit the representational capacity of language models without engaging their reasoning capability. The emergence of reinforcement learning-trained reasoning models such as FinR1 suggests a qualitatively different possibility: a system that does not merely retrieve or compare, but plans, computes, and justifies investment recommendations step by step.

The gap between this possibility and its reliable realisation is, however, substantial. As the experimental evidence presented in Chapter 1 suggests, even the strongest available financial reasoning model produces outputs that are linguistically fluent but numerically unreliable when operating without external computational support. The specific failure modes identified, date blindness, sycophantic fabrication of metrics, and the substitution of optimisation results with rounded heuristic values, are not marginal edge cases. They appear to be systematic consequences of the probabilistic generation mechanism that underlies all current language models. Addressing them is a precondition for any deployment of these systems in a professional advisory context, where fabricated outputs carry regulatory and fiduciary consequences.

The research question investigated in this thesis emerges from the convergence of three conditions that coincided in 2024 and 2025. First, the availability of domain-specialised reasoning models capable of structured multi-step financial analysis. Second, the consolidation of the agentic paradigm as a mature engineering approach, in which language models act as semantic coordinators rather than self-contained knowledge sources. Third, the demonstrated effectiveness of Test-Time Compute techniques in improving the reliability of reasoning models in other quantitative domains. Whether these three components can be integrated into a coherent architecture that is both technically robust and practically useful for professional financial advisory had not been empirically investigated at the time this work was initiated. The motivation for this thesis is to investigate whether that gap can be partially addressed through the proposed architecture, not only by constructing the system but by subjecting it to an evaluation that goes beyond academic benchmarks and includes expert qualitative assessment of real investment scenarios.

3.2 Objectives

The **general objective** of this thesis is to design, implement, and empirically validate a hybrid architecture that seeks to reduce the structural reliability limitations of the FinR1

baseline in quantitative financial advisory tasks, without modifying the model's parameters, and to assess its practical utility through expert-validated investment scenarios.

This general objective is decomposed into three specific objectives:

1. **Development of a Zero-Abstraction ReAct protocol and a Deterministic Financial Engine.** Design and implement a custom orchestration layer that enables FinR1 to delegate all numerical computation to a verified Python backend, bypassing the formatting failures and process hallucinations observed in existing agentic frameworks during preliminary testing. The Financial Engine must support the quantitative operations required by professional advisory scenarios, including covariance matrix estimation, Sharpe Ratio maximisation, CAPM regression, and Value at Risk computation.

2. **Implementation and comparative evaluation of Test-Time Compute strategies.** Investigate whether inference-time scaling through Majority Voting, Best-of-N Vanilla, and Best-of-N Weighted selection can recover a meaningful fraction of the latent reasoning capacity of the tool-augmented system that is lost due to individual formatting failures, and characterise the degree of alignment between general-purpose Reward Model scores and mathematical correctness in quantitative financial tasks.

3. **Validation in real-world advisory scenarios.** Evaluate the integrated architecture against a battery of 23 expert-designed investment scenarios covering sector portfolio optimisation, macroeconomic strategy design, risk-capped exposure, and retrospective analysis, with qualitative review provided by a domain expert with professional experience in investment banking and FinTech research.

3.3 SDG Alignment

This project contributes to two United Nations Sustainable Development Goals.

SDG 8: Decent Work and Economic Growth. Professional-grade quantitative financial analysis has historically required access to specialised human expertise and expensive data infrastructure, concentrating advanced advisory capabilities within large financial institutions. The open-weight architecture developed in this thesis provides a reproducible framework that may help reduce these barriers, contributing to a more inclusive financial ecosystem in which smaller organisations and individual investors can access analytical tools of comparable quality.

SDG 9: Industry, Innovation and Infrastructure. The Zero-Abstraction ReAct protocol and the Deterministic Financial Engine represent a concrete engineering contribution to the responsible integration of AI in regulated industries. The explicit separation of language generation from deterministic computation establishes a design template that may be applicable to other high-stakes domains where the fabrication of outputs by AI systems poses unacceptable operational or regulatory risk.

Chapter 4 Methodology

4.1 System Architecture Overview

The experimental evidence presented in Chapter 1 identified three structural limitations in the baseline FinR1 model when applied to professional financial advisory tasks: the generation of fabricated numerical metrics, the inability to retrieve current market data, and a near-total absence of verifiable grounding in real data across the 23 evaluation scenarios. These limitations cannot be addressed through prompt engineering alone, since they arise from the probabilistic nature of language model generation rather than from a lack of instructions. Two complementary strategies were therefore investigated to address them: tool-augmented generation and inference-time scaling through Test-Time Compute (TTC).

The first strategy, tool-augmented generation, addresses the root cause of numerical unreliability by separating language generation from mathematical computation entirely. Under this approach, the language model no longer attempts to perform calculations or recall market data from its training parameters. Instead, it acts as a semantic coordinator: it interprets the user's request, identifies what information is needed, and delegates all quantitative tasks to an external deterministic Python backend. This ensures that every numerical value in the final report is the result of an explicit computation rather than a probabilistic generation step.

The second strategy, Test-Time Compute, addresses a residual vulnerability that persists even in the tool-augmented system. Even when the external engine is available, the model must still generate a syntactically correct tool invocation command in JSON format. A single formatting error in that command is sufficient to interrupt the entire reasoning process. Rather than relying on a single generation attempt, TTC generates $N = 8$ independent reasoning trajectories in parallel and selects the best one through an evaluation mechanism. This transforms a brittle single-point execution into a more robust ensemble: even if some trajectories fail due to formatting errors, others are likely to succeed.

The architecture that implements both strategies is deployed on a high-performance computing cluster equipped with NVIDIA DGX H200 nodes, each featuring 8 GPUs with 140 GB of VRAM. The implementation uses the Hugging Face `transformers` library [22] and runs in 16-bit brain floating-point precision (`bfloat16`). To improve memory efficiency and prevent Out-Of-Memory (OOM) errors during parallel trajectory generation, FlashAttention-2 [23] is enabled and dynamic CUDA memory allocation is configured through the `PYTORCH_CUDA_ALLOC_CONF` environment variable.

As illustrated in Figure 1, the system consists of three main components operating within a closed execution loop, coordinated by the TTC layer:

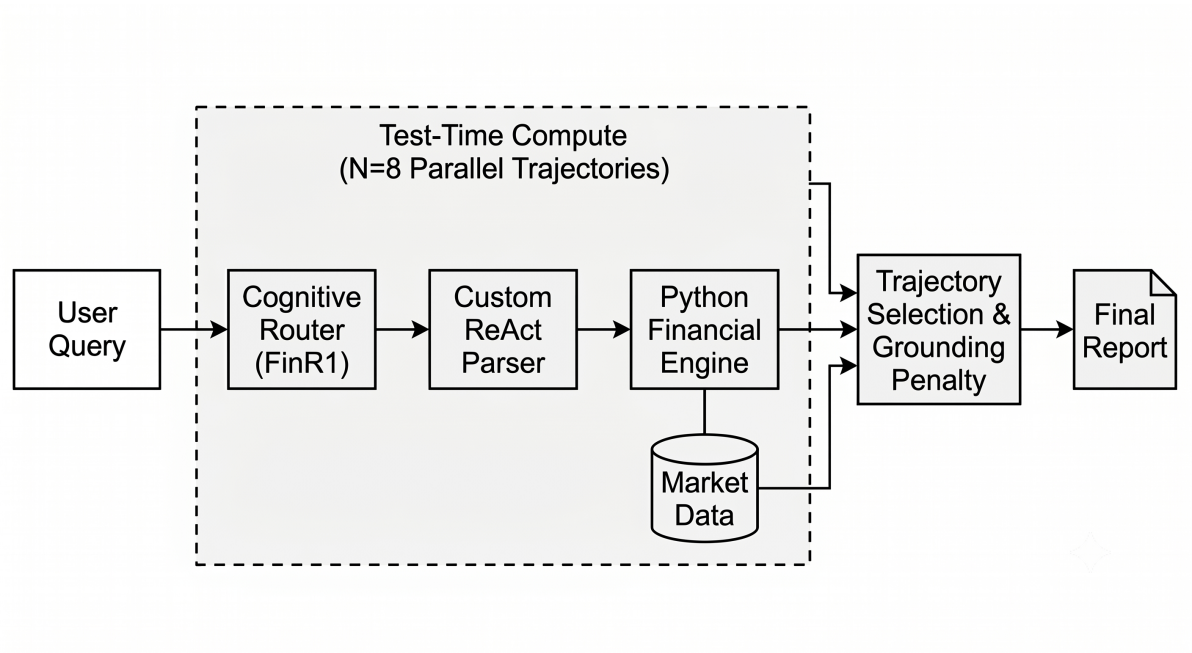

Figure 1: Block diagram of the proposed hybrid architecture, showing the flow from the user query to the final report through the three main components: the Cognitive Router (FinR1), the Custom ReAct Orchestration Layer, and the Deterministic Financial Engine. The TTC layer wraps the entire execution loop, generating $N = 8$ parallel trajectories and selecting the best candidate through the Trajectory Selection and Grounding Penalty stages.

1. **The Cognitive Router (FinR1):** This component is the reasoning core of the system. After receiving a user query, the model does not attempt to answer the financial question directly. Instead, it performs a Chain of Thought (CoT) reasoning process to identify what information and calculations are needed, and determines which external tool must be called to obtain them.

2. **The Custom Orchestration Layer (Zero-Abstraction ReAct):** This layer was developed from scratch for this project, rather than relying on existing frameworks such as LangChain [5] or SmolAgents [15], which produced formatting failures and process hallucinations during preliminary testing. The layer monitors the token stream generated by FinR1 in real time, detects tool invocation commands, and executes the corresponding Python function. The result is then injected back into the model's context window as a verified observation. This approach follows the ReAct paradigm [6] and is described in detail in Section 4.3.

3. **The Deterministic Financial Engine:** This component performs all numerical computations. It uses the scientific Python libraries NumPy, Pandas, SciPy, and `yfinance` to retrieve real S&P 500 market data and execute quantitative analyses, including portfolio optimisation, Sharpe Ratio maximisation, and Value at Risk estimation. All results are produced deterministically and are fully reproducible. The engine is described in detail in Section 4.4.

Operational flow. The execution of a query under the full architecture follows a branch-and-converge approach, illustrated in Figure 2. Rather than producing a single response, the system generates $N = 8$ independent reasoning trajectories through probabilistic sampling. Each trajectory executes its own complete ReAct cycle, moving through three sequential stages: the model reasons about the query and identifies the required tool (Thought), it issues a structured JSON command to invoke that tool (Action), and the engine result is returned and incorporated into the context (Observation). The trajectory is then completed with a final advisory report.

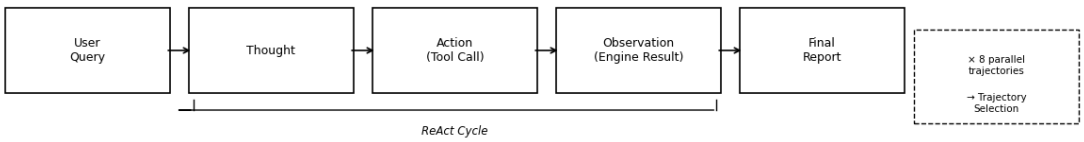

Figure 2: Execution flow of a single trajectory under the TTC pipeline. The model moves through three sequential stages, Thought, Action, and Observation, before generating the Final Report. Eight such trajectories are produced in parallel; the best candidate is selected through the Trajectory Selection mechanism described in Section 4.6.

Once the N trajectories are complete, the best candidate is selected through the Trajectory Selection mechanism. Two strategies are evaluated in this thesis: Majority Voting, which selects the answer supported by the largest number of independent trajectories [16], and Best-of-N, which uses an external Reward Model to score each candidate and select the highest-rated one [18]. Both strategies are described in detail in Section 4.6.

A key limitation of Reward Model-based selection is that general-purpose evaluators are trained to assess linguistic quality rather than mathematical correctness. As a result, a trajectory containing fabricated numerical values may receive a higher score than a shorter but factually correct one. To address this misalignment between the selection metric and the true objective, the architecture incorporates a Grounding Penalty mechanism, described in Section 4.4.8, which assigns a large negative score to any trajectory whose numerical outputs cannot be traced back to the engine observation. This ensures that factual grounding takes priority over linguistic quality in the selection process.

The following sections describe each component of the architecture in detail, following the same incremental structure used to motivate them: Section 4.2 covers the FinR1 model configuration, Section 4.3 describes the custom ReAct orchestration layer, Section 4.4 presents the Deterministic Financial Engine, Section 4.5 explains the TTC pipeline, Section 4.6 covers trajectory selection, and Section 4.7 defines the evaluation framework.

4.2 The Foundational Model (FinR1) Setup

The first step in building the proposed architecture was establishing a stable and reproducible configuration for the foundational language model. As described in Section 1.2, `SUFE-AIFLM-Lab/Fin-R1` was selected as the baseline reasoning model based on its superior performance on the FinQA benchmark relative to the other candidates evaluated. This section describes the technical decisions made to deploy the model reliably on the available hardware and to control its output format in a way that supports both automated evaluation and tool-augmented execution.

4.2.1 Model Initialisation and Precision Constraints

The model was deployed locally on the NVIDIA DGX H200 infrastructure using the Hugging Face `transformers` library [22]. To balance computational efficiency, memory usage, and numerical stability, the model weights were loaded in 16-bit brain floating-point format (`bfloat16`). Compared to standard 16-bit precision (`float16`), `bfloat16` preserves the same dynamic range as 32-bit precision (`float32`), which helps avoid numerical instability during the long reasoning chains that FinR1 produces before arriving at a final answer.

Distribution of the model across the eight available GPUs was handled automatically through the `device_map="auto"` parameter of the Hugging Face library, which balances tensor allocation across hardware resources without manual configuration. FlashAttention-2 [23] was explicitly enabled to reduce memory fragmentation during generation, which is particularly relevant when producing long Chain of Thought sequences.

4.2.2 Inference Hyperparameters and Control

The model is evaluated under two distinct execution settings that correspond to the two phases of the experimental design. The first is the baseline configuration, used to measure the performance of FinR1 without any architectural enhancements. The second is the TTC configuration, which enables the generation of multiple independent reasoning trajectories.

In the baseline configuration, greedy decoding is used (`do_sample=False`), meaning the model always selects the most probable next token at each step. This produces fully deterministic outputs and serves as the reproducible reference point against which all subsequent improvements are measured.

In the TTC configuration, probabilistic sampling is enabled (`do_sample=True`) with a generation temperature of $T = 0.7$ and a nucleus sampling threshold of $p = 0.90$. These settings introduce diversity across the $N = 8$ parallel trajectories while avoiding the generation

of very low-probability token sequences that could lead to incoherent reasoning. The rationale for the specific choice of $T = 0.7$ is discussed in Section 4.5.1. To ensure reproducibility across all experimental runs, a global random seed (`SEED = 42`) is set in both the PyTorch and Hugging Face environments before any inference call.

4.2.3 Context Window Management and Generation Budgets

Financial reasoning tasks require the model to process substantial amounts of context, including the original query, the tool documentation, the Chain of Thought reasoning, and the engine observation. To prevent context saturation and reduce the risk of Out-Of-Memory (OOM) errors on the GPU cluster, the maximum prompt length was capped at 4096 tokens (`MAX_LEN_PROMPT`).

The token generation budget is divided into two sequential stages that reflect the two-step execution structure of the tool-augmented architecture:

- **Step 1** (`MAX_NEW_TOKENS_STEP1`): allocated to the model's Chain of Thought reasoning process and the generation of the tool invocation command inside the `<action>` tag. This budget is set to 1024 tokens for the FinQA benchmark experiments and increased to 2048 tokens for the case study, where queries are more complex and require longer reasoning chains.
- **Step 2** (`MAX_NEW_TOKENS_STEP2`): allocated to processing the engine observation and generating the final advisory report. This budget is set to 512 tokens for the benchmark and 1024 tokens for the case study.

4.2.4 System Prompting and Output Grounding

A practical challenge when working with reasoning-oriented language models is that they tend to produce long reasoning traces before stating a final conclusion. For automated evaluation, it is necessary to reliably extract the final answer from the generated text without ambiguity.

To achieve this, all experiments use a system prompt that instructs the model to enclose its final answer inside a `\boxed{}` LaTeX tag immediately at the start of its response:

> *"You are a helpful financial expert AI. IMPORTANT: You must start your response IMMEDIATELY with the final result enclosed in LaTeX box format like* `\boxed{answer}`*. After providing the boxed answer, explain your step-by-step reasoning."*

This formatting constraint serves two purposes. First, it allows the evaluation backend to extract the final answer deterministically using a simple string parser, without having to interpret the reasoning trace. Second, it provides a clear structural anchor for the ReAct orchestration layer, which uses the presence of the `\boxed{}` tag to confirm that the model has completed its response. Answer correctness is then assessed by comparing the extracted value against the ground truth using `Qwen2.5-72B-Instruct` as an automated judge, replicating the evaluation methodology of the original FinR1 paper [3] to ensure comparability of results.

4.3 Custom ReAct Protocol and Robust Parsing

As established in Section 1.4, connecting a language model to an external execution environment requires the model to generate syntactically correct tool invocation commands at runtime. This is a non-trivial requirement: a single missing quotation mark, an unescaped character, or a conversational prefix surrounding the JSON payload is sufficient to cause the execution environment to fail and interrupt the entire reasoning process. Under a single-trajectory regime, this type of failure is unrecoverable.

This section describes the failure modes identified during preliminary testing with existing orchestration frameworks, the design of the custom ReAct loop built to replace them, and the five-layer parsing architecture implemented to make the system robust against the formatting variability introduced by probabilistic sampling.

4.3.1 Empirical Failure Modes of Commercial Frameworks

The initial experiments were conducted using two widely adopted orchestration frameworks: LangChain's Expression Language pipeline and Hugging Face's SmolAgents. Both were tested with restrictive system prompts and explicit JSON schemas. Despite these constraints, the integration with FinR1 revealed three categories of practical failure that made these frameworks unsuitable for the requirements of the system.

The first category is **token overhead and context growth**. Both frameworks inject additional system instructions and template text into the model's context window. When FinR1 generates long Chain of Thought reasoning traces, this overhead increases memory usage and inference latency, and raises the risk of context saturation on the GPU cluster.

The second category is **syntactic fragility**. Both frameworks rely on strict JSON parsing. If FinR1 added a conversational prefix before the JSON payload, such as *"Sure, I will compute that for you:"*, or used Python-style single quotes instead of JSON-standard double quotes when specifying tool arguments, the framework raised a `JSONDecodeError` and halted execution. These formatting deviations are common in reasoning models operating under probabilistic sampling.

The third category is **process hallucinations**. The additional abstraction layers introduced by these frameworks made it harder for the model to distinguish between its internal reasoning and the external execution environment. In several cases, FinR1 generated text that described a tool call and its result as if both had already occurred, without any actual interaction with the Python backend. The following is an example of this failure pattern, where the model fabricates both the call and the result instead of delegating to the engine:

```
<action>calculate_ratio(150, 300)</action> and the result is 0.5
```

These three failure modes motivated the decision to abandon commercial frameworks entirely and implement a custom orchestration layer from scratch.

4.3.2 Design of the Zero-Abstraction ReAct Paradigm

The custom orchestration layer is a lightweight synchronous loop that provides direct control over the token stream generated by FinR1. Rather than relying on hidden templates or internal parsing logic, the loop divides execution into a two-stage state machine, illustrated in Figure 3:

- **Stage 1, Tool Invocation:** The model receives the user query together with a concise description of the available tools. It is required to format any tool request using explicit XML-style markers wrapping a JSON payload:

  ```
  <action>{"tool": "function_name", "input": {"param": value}}</action>
  ```

  As soon as the orchestration script detects the closing tag `</action>`, generation is interrupted and the text is truncated at that point. This creates a hard execution checkpoint that prevents the model from generating a fabricated result after the tool call.

- **Stage 2, Final Report:** Once the Python engine executes the requested function and returns a result, that result is injected into the model's context window as a verified `<observation>`. The model then generates its final advisory report based exclusively on that observation.

This two-stage design eliminates process hallucinations by construction: the model cannot generate a tool result because execution is physically interrupted before it can do so.

4.3.3 The Five-Layer Cascade Parser Architecture

Even with the custom orchestration loop in place, the probabilistic nature of language model generation means that formatting errors will occasionally occur, particularly when $N = 8$ trajectories are produced in parallel under temperature sampling. A single formatting error should not terminate an entire trajectory. To handle this, a fault-tolerant parsing cascade was implemented that attempts to recover a valid tool invocation through five progressively more permissive layers, illustrated in Figure 3.

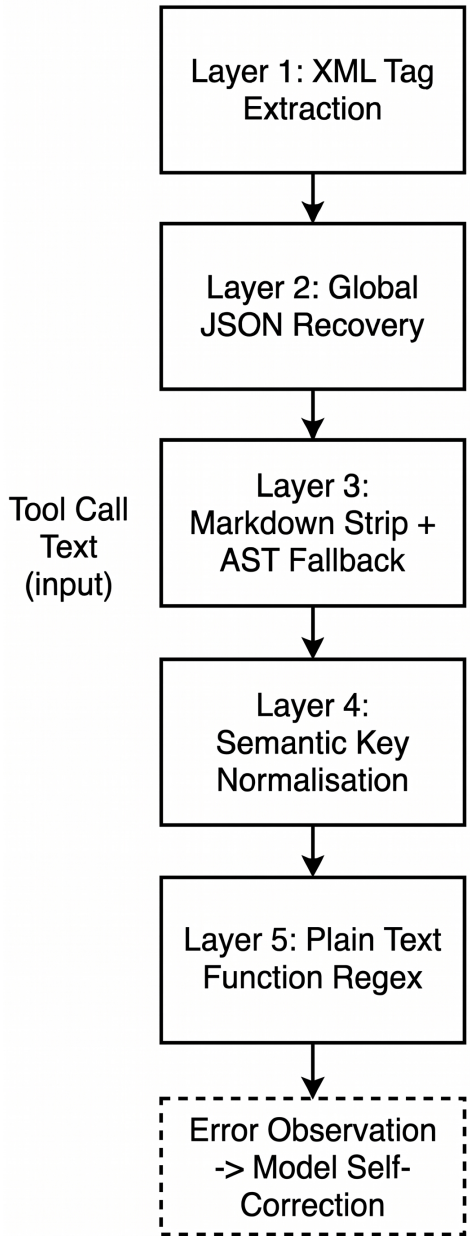

Figure 3: The five-layer cascade parser architecture. Each layer attempts to recover a valid tool invocation from the model's generated text using a progressively more permissive strategy. If all five layers fail, a structured error observation is returned to the model's context window, allowing it to self-correct in Stage 2.

1. **Layer 1, XML Extraction:** The parser searches for content enclosed within `<action>...</action>` tags using a regular expression. This is the preferred path and handles the majority of cases when the model follows the required format correctly.

2. **Layer 2, Global JSON Recovery:** If the XML tags are absent, the parser scans the entire generated text for any substring enclosed within curly braces, extracts it, and attempts to decode it with `json.loads()`.

3. **Layer 3, Markdown Stripping and AST Fallback:** FinR1 occasionally wraps JSON outputs inside markdown code blocks. This layer strips those formatting elements automatically. If standard JSON decoding still fails due to single quotes or incorrectly escaped characters, the parser falls back to Python's Abstract Syntax Tree library (`ast.literal_eval`), which can safely interpret Python dictionary literals.

4. **Layer 4, Semantic Key Normalisation:** Even when a valid dictionary is recovered, the model may use alternative field names. This layer maps synonyms such as `"method"` or `"function"` to the expected `"tool"` key, and `"parameters"` or `"args"` to the expected `"input"` key.

5. **Layer 5, Plain Text Function Signature:** As a final fallback, if the model generates a plain-text function call such as `Action: calculate_ratio (numerator=150, denominator=300)`, the parser extracts the function name and arguments using regular expressions and reconstructs the execution payload programmatically.

If none of the five layers recovers a valid tool invocation, the system returns a structured error message to the model's context window:

```
<observation>Error: Invalid tool calling format.</observation>
```

This message is passed to Stage 2, where the model can acknowledge the failure and attempt to answer the query using whatever information it has available. In the TTC pipeline, such trajectories typically receive a low selection score and are unlikely to be chosen as the final response.

4.4 The Deterministic Financial Engine

The baseline FinR1 model, when operating without external tools, consistently produced numerical outputs that could not be verified against real market data, as documented in Section 1.3. The root cause of this behaviour is structural: a language model generates numbers by predicting the most likely token sequence based on patterns in its training data, not by executing mathematical operations. No amount of instruction in the system prompt can change this fundamental constraint.

The Deterministic Financial Engine was built to address this limitation directly. Rather than asking the language model to perform calculations, the engine acts as an external Python backend that receives structured requests from the model, executes the required computations using real market data, and returns verified numerical results. The model's role is reduced to interpreting the user's request and deciding which tool to call. Every number that appears in

the final report is the result of an explicit Python computation, not a probabilistic generation step.

The development of the engine proceeded in two stages that reflect the incremental nature of the experimental design. In a first exploratory phase, a more advanced engine, referred to as `FinancialEngine`, was implemented as a Python class with substantially richer capabilities: Carhart four-factor Alpha estimation via OLS regression against Fama-French factors downloaded from `pandas_datareader`, implied volatility retrieval through numerical inversion of the Black-Scholes formula using Brent's root-finding algorithm [24], and a full annualised covariance matrix. However, the integration of this advanced engine with the TTC pipeline did not produce reliable results in the evaluated experiments. The richer and more structured JSON outputs it generated substantially increased the complexity of the model's parsing task, aggravating the syntactic failure modes described in Section 4.3 and preventing consistent tool execution across parallel trajectories. This finding informed the decision to develop a second, simplified engine with a more constrained tool interface, whose outputs the model could parse reliably under probabilistic sampling conditions.

The simplified engine deployed in the final evaluation is implemented using four scientific Python libraries: `numpy` for linear algebra operations, `pandas` for time-series management, `scipy` for numerical optimisation and statistical computation, and `yfinance` for retrieving historical market data from Yahoo Finance. All tools are registered in a single dispatch dictionary, `TOOL_FN_MAP`, which maps tool names to their corresponding Python functions. When the ReAct parser described in Section 4.3 extracts a tool name from the model's output, it looks up that name in this dictionary and executes the corresponding function directly.

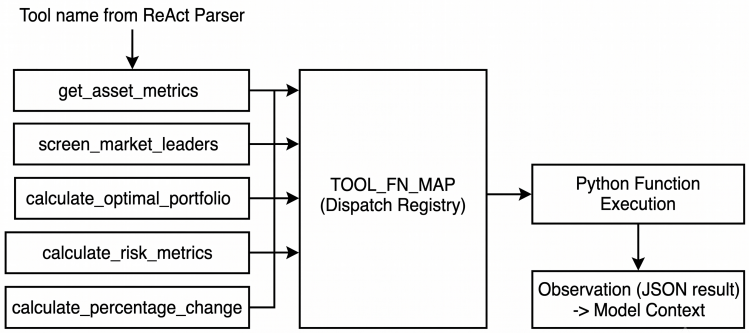

Figure 4: Overview of the Deterministic Financial Engine deployed in the final evaluation. The five tools registered in `TOOL_FN_MAP` are dispatched by name from the ReAct parser. Each tool executes a deterministic Python computation and returns a verified JSON result that is injected into the model's context window as an observation.

4.4.1 Temporal Synchronisation

A key requirement of the engine is that market data retrieval must respect the temporal context of each scenario. The 23 investment scenarios in the case study are explicitly anchored to specific dates, such as end-of-2024 or mid-2020. If the engine were to download data up to the current system date instead, the evaluation would be meaningless, since the model would be answering questions about one period using data from a different one.

To prevent this, the function `sync_simulated_time` is called once per query before any tool is invoked. It parses the query text to identify the relevant date and stores it in a module-level variable, `SIMULATED_TODAY`, which is then used as the upper boundary for all subsequent data downloads. The parsing follows a deterministic priority rule: queries containing the string `"2024"` set `SIMULATED_TODAY` to `"2024-12-31"`, while `"2020"` and `"2021"` map to `"2020-06-01"` and `"2021-01-01"` respectively. All other queries are resolved using the `dateparser` library with a fixed reference base of 17 February 2026.

4.4.2 Market Data Layer

All quantitative tools in the engine rely on a common data foundation: a time series of daily log-returns for a given set of assets. The function `get_historical_returns` builds this series by downloading two years of daily adjusted closing prices from Yahoo Finance and computing:

$$r_{i,t} = \ln\left(\frac{P_{i,t}}{P_{i,t-1}}\right), \tag{4.1}$$

where $P_{i,t}$ is the adjusted closing price of asset i on day t. Log-returns are used instead of simple percentage changes because they are additive over time, which simplifies the annualisation of statistics such as volatility and expected return.

The investable universe available to the agent is maintained in `get_sp500_data`, a static dictionary that maps ticker symbols to their GICS sector classifications. The universe was deliberately expanded between the two evaluation phases. The `FinR1 + Tools` configuration covered 30 tickers across Technology, Energy, and Financials, plus four macro instruments, `GLD` (gold), `USO` (crude oil), `TLT` (long-term US Treasuries), and `VNQ` (real estate). The final `FinR1 + Tools + TTC` configuration added sub-industry classifications and extended coverage to Industrials (`DAL`, `UAL`, `FDX`) and Materials (`DOW`), providing richer context for sector-rotation and commodity-exposure scenarios.

4.4.3 Asset Metrics Tool

The `get_asset_metrics` tool computes three standard performance metrics for a given list of assets. For each asset i, the engine calculates the annualised expected return μ_i, the annualised volatility σ_i, and the Sharpe Ratio S_i, which measures risk-adjusted performance by expressing how much return an asset generates per unit of risk taken:

$$S_i = \frac{\mu_i - R_f}{\sigma_i}, \tag{4.2}$$

where $R_f = 0.045$ is the annualised risk-free rate, calibrated to the US 10-year Treasury yield at the time of system development. Both μ_i and σ_i are computed from the daily log-return series and scaled to annual values by multiplying by 252 (the standard number of trading days per year) for the mean, and by $\sqrt{252}$ for the standard deviation. The results are returned as a JSON object and injected into the model's context window as a verified observation.

4.4.4 Market Screening Tool

The `screen_market_leaders` tool allows the agent to identify the best-performing assets within a given sector, ranked by Sharpe Ratio. Given an optional sector filter and a configurable number of results k, the function retrieves returns for up to 50 matching assets, computes their Sharpe Ratios using Equation (4.2), and returns the top-k ranked in descending order.

An important guardrail is built into this tool: if fewer than k assets are available in the requested sector, the observation payload includes an explicit warning instructing the model not to invent additional tickers. The severity of this warning was strengthened between the two evaluation configurations after observing that the original phrasing was occasionally insufficient to prevent the model from fabricating extra asset names:

- `FinR1 + Tools`: `System Warning: Only found {n} assets. Do not invent more.`
- `FinR1 + Tools + TTC`: `WARNING_CRITICAL: Only found {n} assets. DO NOT attempt to reach {k}. Adjust your analysis to {n}.`

4.4.5 Portfolio Optimisation Tool

The `calculate_optimal_portfolio` tool implements Markowitz mean-variance optimisation [25] to find the portfolio allocation that maximises the Sharpe Ratio. Given a list of assets, the engine estimates their expected returns and covariance matrix from the historical log-return series. The goal is to find the combination of weights that delivers the highest return per unit of risk taken. Formally, the engine solves:

$$\mathbf{w}^* = \arg\max_{\mathbf{w}} \frac{\mathbf{w}^\top \boldsymbol{\mu} - R_f}{\sqrt{\mathbf{w}^\top \boldsymbol{\Sigma}\, \mathbf{w}}}, \tag{4.3}$$

subject to the constraint that all weights sum to one and each individual weight lies in the interval $[0.01, 1]$. The lower bound of 0.01 was introduced during preliminary testing to prevent solver instability when optimising portfolios with a small number of assets. The

problem is solved using `scipy.optimize.minimize` with the SLSQP method, starting from an equally weighted initial guess.

In addition to the optimal portfolio, the tool also computes an equally weighted benchmark and returns both side by side. This benchmark was included after observing that, without a reference point provided in the observation, the model tended to generate its own comparison figures, introducing fabricated metrics into the final report.

All outputs are pre-formatted as percentage strings before being returned, for example `"14.37%"`, to avoid the rounding and unit conversion errors that were observed when raw floating-point values were passed directly to the language model.

4.4.6 Risk Metrics Tool

The `calculate_risk_metrics` tool computes the Value at Risk (VaR) for a given set of assets. VaR is a standard risk management metric that estimates the maximum loss expected over a given time horizon at a specified confidence level. For example, a daily VaR of 2% at 95% confidence means that, on any given day, losses are not expected to exceed 2% of the portfolio value with 95% probability.

The tool uses the parametric method, which assumes that daily log-returns are approximately normally distributed. Under this assumption, the one-day VaR at confidence level α is:

$$\mathrm{VaR}_{\alpha}^{(\mathrm{daily})} = z_{\alpha} \cdot \hat{\sigma} - \hat{\mu}, \tag{4.4}$$

where $z_{\alpha} = \Phi^{-1}(\alpha)$ is the quantile of the standard normal distribution at level α, computed using `scipy.stats.norm.ppf`, $\hat{\mu}$ is the sample mean of daily log-returns, and $\hat{\sigma}$ is their sample standard deviation. Annual VaR is then obtained by scaling with the square-root-of-time rule:

$$\mathrm{VaR}_{\alpha}^{(\mathrm{annual})} = \mathrm{VaR}_{\alpha}^{(\mathrm{daily})} \times \sqrt{252}. \tag{4.5}$$

Both metrics are returned as percentages alongside the `SIMULATED_TODAY` timestamp, so that the data window underlying the estimate is fully traceable.

4.4.7 Tool Registry and Interface

The five tools described above are registered in `TOOL_FN_MAP` and dispatched by name through the ReAct parser. Table 1 summarises the complete interface available to the agent.

Table 1: Tool registry of the Deterministic Financial Engine deployed in the final evaluation.

Tool identifier	Key inputs	Computed output
`get_asset_metrics`	`tickers`	Annualised μ_i, σ_i, S_i per asset
`screen_market_leaders`	`sector`, `limit` k	Top-k assets ranked by S_i
`calculate_optimal_portfolio`	`tickers`	Optimal weights $\mathbf{w}^*$, portfolio return and volatility, EW benchmark
`calculate_risk_metrics`	`tickers`, α	Daily and annual VaR_α
`calculate_percentage_change`	`new\_value`, `old\_value`	Arithmetic return (%)

4.4.8 Grounding Penalty Mechanism

Even with a deterministic engine providing verified numerical results, a residual risk remains: the language model might ignore the observation and generate its own numbers in the final report. This is particularly relevant in the TTC pipeline, where the Reward Model used to select the best trajectory evaluates linguistic quality rather than numerical accuracy. A response that fabricates plausible-looking metrics may receive a higher Reward Model score than a shorter but factually correct one, leading the selection mechanism to favour the wrong trajectory.

To address this misalignment between the selection metric and the true objective, the engine includes a post-generation consistency check called `evaluate_grounding_penalty`. After all $N = 8$ trajectories have been generated, this function scans each final report for decimal numbers and percentage values and checks whether each one appears in the JSON observation returned by the tool. If any numerical value in the report cannot be traced back to the observation, the trajectory receives a penalty of $-9{,}999$ points added to its Reward Model score. Since this penalty far exceeds any plausible Reward Model score, such trajectories are effectively eliminated from selection. The composite score for trajectory k is:

$$\tilde{s}_k = s_k^{\text{RM}} + \delta_k, \qquad \delta_k = \begin{cases} -9{,}999 & \text{if ungrounded numerical values are detected,} \\ 0 & \text{otherwise.} \end{cases} \tag{4.6}$$

The selected trajectory is $k^* = \arg\max_k \tilde{s}_k$. This mechanism ensures that factual grounding takes priority over linguistic quality in the selection process.

4.4.9 Compliance and Audit Guardrails

The engine also includes a lightweight function, `audit_guardrails`, that inspects the model's Step 1 output before the observation is injected. The function checks for two categories of logical error that were observed during intermediate evaluation rounds.

The first category involves attempts to short or underweight inverse instruments such as `TIP`, `TLTU`, or `TBF`. Shorting an instrument that is already inverse in nature represents a fundamental market logic error, and any such attempt is flagged with a corrective message before the observation is returned.

The second category involves manual VaR calculations. If the model's Step 1 output contains references to VaR without an accompanying call to `calculate_risk_metrics`, the system appends a warning instructing the model to delegate the computation to the tool rather than deriving the value from its parametric memory.

Both checks were introduced in direct response to failure modes observed during the evaluation of the `FinR1 + Tools` configuration, and contributed to the improvement in response quality observed in the final architecture.

4.5 Test-Time Compute (TTC) Pipeline

The tool-augmented architecture described in Sections 4.3 and 4.4 substantially reduced the fabrication of numerical metrics and asset identifiers observed in the baseline model. However, the `FinR1 + Tools` configuration still achieved a correct response rate of approximately 65% on the 23-scenario case study. Analysis of the remaining failures indicated that the limiting factor was not computational, since the Financial Engine consistently returned correct results when invoked, but syntactic. A single malformed JSON payload during the *Action* stage was sufficient to interrupt the reasoning process before the engine was ever called. Under a single-trajectory setup ($N = 1$), this type of failure has no recovery path.

Test-Time Compute (TTC) was introduced to address this fragility. Rather than relying on a single generation attempt, the system generates $N = 8$ independent reasoning trajectories and selects the best candidate through an evaluation stage. The underlying intuition is straightforward: while some trajectories may fail due to formatting errors, others are likely to produce a valid tool call and obtain a correct result from the engine. Generating multiple attempts in parallel increases the probability that at least one trajectory succeeds, without requiring any modification to the model's weights.

This decision was supported by the Oracle Pass@3 evaluations conducted during the initial benchmarking phase, described in Section 1.2. Oracle Pass@3 measures whether the correct answer appears at least once among three independent generation attempts for the same question. It can be thought of as asking: if the model were allowed three tries, would it get it right at least once? A model with a high Oracle Pass@3 score but a low single-attempt accuracy is one that knows how to reason correctly but does not do so consistently, which is precisely the profile observed in FinR1. This gap between what the model can do in its best attempt and what it delivers on average is exactly the space that Test-Time

Compute techniques are designed to exploit. Rather than hoping that the single generation attempt happens to be a good one, TTC generates multiple attempts and selects the best, systematically recovering the latent reasoning capacity that single-shot evaluation leaves on the table.

4.5.1 Generation Configuration

In the baseline and `FinR1 + Tools` configurations, all generation used greedy decoding (`do_sample=False`), which always selects the most probable next token and produces fully deterministic outputs. While this is appropriate for single-trajectory evaluation, it provides no diversity across parallel runs, since every trajectory would produce the same output.

For the TTC pipeline, probabilistic sampling is enabled (`do_sample=True`) with a temperature of $T = 0.7$ and a nucleus sampling threshold of $p = 0.90$. Temperature controls the diversity of the generated text: higher values produce more varied outputs, while lower values make the model more conservative and repetitive. The value $T = 0.7$ was selected after empirical evaluation of this trade-off for FinR1. Values above $T = 0.8$ produced excessive variability in the reasoning stage, leading to inconsistent tool selection across trajectories. Values below $T = 0.5$ reduced diversity to the point where multiple trajectories converged to the same JSON payload, including cases where that payload was incorrect, which defeats the purpose of generating multiple attempts.

To ensure reproducibility across experimental runs, a global random seed (`SEED = 42`) is set before any inference call. The token generation budgets are the same as those defined in Section 4.2.3: up to 2048 tokens for the reasoning and tool invocation stage, and up to 1024 tokens for the final report.

4.5.2 Two-Step Parallelised Execution

Each query is processed through the two-stage execution loop described in Section 4.3.2, replicated $N = 8$ times in parallel. The overall flow is illustrated in Figure 2.

Step 1, Parallel Chain of Thought and Tool Invocation. The system prompt and user query are replicated into a batch of N identical inputs, all processed simultaneously by the language model with `do_sample=True`. Each of the N outputs contains a reasoning trace followed by a tool invocation command inside `<action>` tags.

The firewall truncation mechanism described in Section 4.3.2 is applied independently to each trajectory: as soon as `</action>` is detected, generation is interrupted and the output is truncated at that point. This prevents the model from fabricating a tool result after the call. The five-layer cascade parser is then applied to each trajectory independently to extract the tool name and arguments.

If the parser fails to recover a valid tool call, a Hard Reset is triggered: the error is injected into the context and the model is asked to regenerate a clean JSON payload using greedy decoding with a budget of 512 tokens. If the retry also fails, the trajectory receives

a structured error observation and continues to Step 2, where it will typically receive a low selection score.

Step 2, Parallel Final Report Generation. For each trajectory, the engine observation (or error message) is appended to the Step 1 context together with an explicit instruction to base the final answer exclusively on the data provided in the observation, and to enclose it inside a `\boxed{}` tag. This instruction serves two purposes: it reduces the likelihood that the model reverts to its parametric memory for numerical values, and it ensures the final answer is in a format that can be parsed automatically for evaluation.

All N Step 2 prompts are processed simultaneously, producing N complete candidate trajectories, each containing a reasoning chain, an engine observation, and a final advisory report.

4.5.3 Batched Inference and the Divergence Anomaly

The N parallel trajectories can be generated in two ways: by calling the model N times sequentially, or by processing all N inputs in a single batched forward pass. Batched generation is faster, but an unexpected behaviour was identified during the FinQA benchmark experiments.

When N responses produced in a single batch were subsequently scored by a Reward Model also operating in batch mode, the score distributions differed systematically from those obtained during sequential scoring. This inconsistency is attributed to differences in how padding tokens and attention masks interact across the two models when processed in batch mode, and is referred to in this thesis as the *batched inference divergence anomaly*.

To mitigate this, the case study pipeline uses a hybrid approach: Step 1 generations are produced in a single batched forward pass for efficiency, while Reward Model scoring is performed in smaller sub-batches of four responses, with explicit GPU cache clearing between sub-batches. This configuration maintained generation throughput while producing more stable and consistent scoring distributions.

4.5.4 Memory Management

Running eight parallel trajectories of a model of this size in half-precision on the DGX H200 cluster requires careful memory management to avoid Out-Of-Memory errors. Four measures were implemented to this end.

First, the CUDA memory allocator is configured to use expandable segments before the model is loaded, which reduces fragmentation during variable-length generation. Second, the input and output tensors from Step 1 are explicitly deleted and GPU cache is cleared before the Reward Model is loaded, releasing the memory occupied by the generation model's activations. Third, FlashAttention-2 [23] is enabled where available, which reduces the memory footprint of the self-attention computation. Fourth, Reward Model scoring is performed

under automatic mixed-precision to avoid accumulating unnecessary full-precision intermediates.

Together, these measures allowed the complete TTC pipeline to run without memory errors across all 23 case study scenarios on the available hardware.

4.6 Trajectory Selection Mechanisms

Once the $N = 8$ parallel trajectories have completed both generation steps, the system needs to select a single response to deliver to the user. This is not a trivial decision: trajectories differ not only in their final numerical answer, but also in whether their reported values are grounded in the engine observation or fabricated from the model's parametric memory. A selection mechanism that rewards linguistic quality without verifying numerical accuracy could systematically favour well-written but incorrect responses over shorter but factually correct ones.

Three selection strategies of increasing sophistication were implemented and evaluated, illustrated in Figure 5. Each strategy addresses a different aspect of response quality.

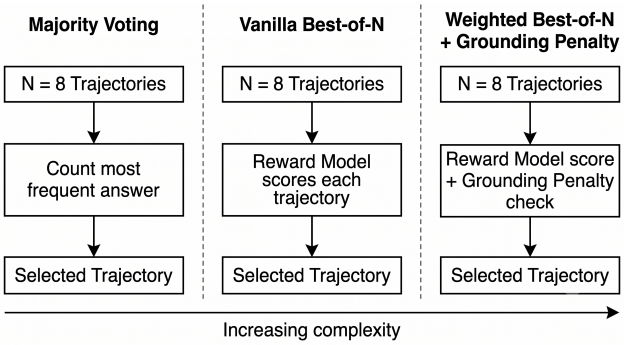

Figure 5: Comparative overview of the three trajectory selection mechanisms evaluated in this thesis. Majority Voting selects the most frequent answer across trajectories. Vanilla Best-of-N uses a Reward Model to score each candidate individually. Weighted Best-of-N combines Reward Model scoring with consensus, and the final variant adds a Grounding Penalty to eliminate trajectories containing fabricated numerical values.

4.6.1 Majority Voting (Self-Consistency)

Majority Voting [16] is based on a simple and intuitive idea: if multiple independent reasoning attempts produce the same answer, that answer is more likely to be correct than one that appears only once. The underlying assumption is that errors tend to be idiosyncratic, varying randomly across trajectories, while the correct deterministic answer, which is anchored to the same engine output, is more likely to appear repeatedly.

In practice, after generating N trajectories, the system extracts the final answer from each one by locating the content inside the `\boxed{}` tag. Trajectories that do not produce a parseable boxed answer are excluded from the vote. Among the remaining candidates, the

answer that appears most frequently is selected, and the full response of the first trajectory that produced that answer is returned as the final output. If no valid answer is found in any trajectory, the system falls back to the first generated response.

Majority Voting is particularly well suited to the FinQA benchmark, where each question has a single correct numerical answer. When the underlying computation is delegated to the deterministic engine, the correct result should appear consistently across successful trajectories. For the open-ended case study scenarios, where the final answer is a multi-component investment recommendation, exact answer matching is less meaningful. In this setting, Majority Voting still provides value as a basic quality filter by excluding trajectories that failed to produce a structured output, but its ability to discriminate between high-quality candidates is limited.

4.6.2 Best-of-N with Reward Model Scoring

Best-of-N (BoN) selection replaces the consensus criterion with an external evaluator. A Reward Model (RM) scores each of the N candidate trajectories and the highest-scoring one is selected. This approach can, in principle, capture quality dimensions that Majority Voting cannot, such as reasoning coherence, instruction following, and response completeness.

Three open-source Reward Models were evaluated for this role:

1. **FsfairX-LLaMA3-RM-v0.1** [18]: a LLaMA-3-based model trained via Direct Preference Optimisation on large-scale human preference data. This model produced the highest correlation with manual accuracy labels in the FinQA experiments ($r \approx 0.21$) and was selected as the primary evaluator for the FinQA TTC pipeline.

2. **Skywork-Reward-Gemma-2-27B-v0.2**: a 27-billion-parameter model that produced more stable score distributions but required substantially more GPU memory, limiting its practicality to small scoring batches.

3. **Skywork-Reward-Llama-3.1-8B-v0.2**: an 8-billion-parameter model used in the final case study pipeline. Its smaller size allowed it to co-reside in GPU memory alongside the generation model without triggering memory errors, making it the most practical option for the multi-step TTC pipeline.

All three are Outcome Reward Models, meaning they assign a single score to the complete response rather than evaluating individual reasoning steps. The implications of this design choice are discussed in Section 5.2.5.

Vanilla Best-of-N. The simplest variant scores each trajectory independently and selects the one with the highest Reward Model score. The score is obtained by passing the query and the full response through the Reward Model, formatted using its chat template, and reading the output logit. Trajectories for which scoring fails are assigned a score of $-\infty$ to ensure they are never selected.

Weighted Best-of-N. This variant introduces a two-stage aggregation that combines Reward Model scoring with soft consensus. Trajectories are first grouped by their extracted final answer. Within each group, the individual Reward Model scores are summed to produce an aggregated group score. The group with the highest aggregated score wins, and the individual trajectory within that group with the highest single score is returned. This approach rewards answers that are both highly rated and supported by multiple independent reasoning paths, combining the strengths of quality assessment and consensus reliability.

Best-of-N with Grounding Penalty. The final variant, used in the case study pipeline, addresses a key limitation of pure Reward Model scoring. General-purpose Reward Models are trained to assess linguistic quality and instruction following, not numerical accuracy. As a result, a response that fabricates plausible-looking financial metrics may receive a higher score than a shorter but factually correct one. This misalignment between the selection metric and the true objective is the source of the *TTC Paradox* discussed in Section 5.2.5.

To address this, the Grounding Penalty described in Section 4.4.8 is added to the Reward Model score before selection. Any trajectory whose final report contains numerical values that cannot be traced back to the engine observation receives a penalty of $-9{,}999$ points, which exceeds any plausible Reward Model score and effectively eliminates that trajectory from contention. The selection then proceeds among the remaining grounded trajectories, using the Reward Model score as the discriminator. This ensures that factual accuracy is a necessary condition for selection, and linguistic quality is only used to distinguish among responses that have already passed the grounding check.

4.6.3 Comparative Summary

Table 2 summarises the four selection strategies evaluated in this thesis, their selection criteria, whether they require a Reward Model, and the experimental context in which each was applied.

Table 2: Comparative summary of trajectory selection mechanisms.

Mechanism	**Selection criterion**	**Requires RM**	**Applied in**
Majority Voting	Most frequent answer	No	FinQA benchmark; case study
Vanilla Best-of-N	Highest RM score	Yes	FinQA benchmark
Weighted Best-of-N	Highest aggregated RM score by answer group	Yes	FinQA benchmark
BoN + Grounding Penalty	Highest RM score among grounded trajectories	Yes	Case study (final)

The empirical comparison of these mechanisms, presented in Chapter 5, revealed that Majority Voting proved robust for tasks with a single deterministic numerical answer, while the Reward Model-based strategies showed a systematic tendency to favour linguistically elaborate responses over factually correct ones. The Grounding Penalty was designed precisely to correct this tendency, and its impact on selection quality is analysed as part of the TTC Paradox discussion in Section 5.2.5.

4.7 Evaluation Framework and Case Study Design

Validating the proposed architecture requires measuring two distinct dimensions of improvement: whether it produces more accurate numerical outputs than the baseline, and whether it behaves reliably in the kind of open-ended financial advisory scenarios for which it was designed. These two dimensions call for different evaluation approaches, and neither alone is sufficient.

The first is addressed through a standardised academic benchmark that allows objective comparison with published results. The second is addressed through a purpose-built case study of real investment scenarios, evaluated by a domain expert. Together, the two phases provide both quantitative rigour and practical relevance.

4.7.1 Phase 1: Quantitative Benchmarking on FinQA

Dataset Selection. The primary benchmark used in this thesis is FinQA [1], a dataset of numerical reasoning questions over real financial documents from S&P 500 earnings reports. Each question requires the model to analyse a passage of text and one or more data tables, and to produce a single numerical answer through a sequence of arithmetic operations. Cor-

rectness is measured by exact-match accuracy after numerical normalisation, which means a response is either correct or incorrect with no partial credit.

FinQA was chosen for two reasons directly aligned with the objectives of this thesis. First, the single numerical answer format isolates mathematical reasoning performance from conversational quality. This is important when evaluating a tool-augmented architecture, because it allows the contribution of the external engine to be measured without interference from the model's ability to generate fluent prose. Second, FinQA is the benchmark used in the original FinR1 evaluation pipeline [3], which means the results obtained here are directly comparable to the published baseline figures, providing an objective external reference point for measuring the improvement introduced by the architectural modifications.

ConvFinQA [2] was considered and rejected as an alternative. ConvFinQA extends FinQA with multi-turn conversational questions, which introduces confounding factors related to dialogue management and context tracking that are orthogonal to the mathematical accuracy improvements targeted here. Using it would risk attributing performance differences to conversational capabilities rather than to the deterministic computation layer, which is the variable under experimental control in this thesis.

Evaluation Protocol. The evaluation was conducted on the validation split of `TheFinAI/FINQA_test_test`, comprising 1,147 instances. Five system configurations were evaluated over the full split under a fixed random seed (`SEED = 42`) to ensure reproducibility:

- FinR1 Base
- FinR1 + Tools
- FinR1 + Tools + Majority Voting
- FinR1 + Tools + Vanilla BoN
- FinR1 + Tools + Weighted BoN

Answer extraction used the `extract_boxed_answer` parser described in Section 4.6. Correctness was assessed using `Qwen2.5-72B-Instruct` as an automated judge, replicating the evaluation methodology of the original FinR1 paper to ensure score comparability.

4.7.2 Phase 2: Strategic Investment Case Study

Motivation and Design Rationale. Performing well on FinQA is a necessary but not sufficient condition for deployment in professional financial advisory. A system that correctly extracts a revenue growth figure from a 10-K filing is not necessarily capable of designing a risk-hedged sector rotation strategy anchored to real market data, managing temporal context, or recognising the boundaries of its own tool capabilities.

The case study was designed to test exactly these dimensions. It was developed in collaboration with the thesis co-director Carlos Bellón Nuñez-Mera, whose professional background

in FinTech and quantitative investment strategy provided the domain expertise needed to formulate scenarios that are both technically demanding and institutionally representative. The guiding criterion for scenario design was reproducibility under expert review: each scenario was formulated so that a qualified financial analyst could independently verify the correctness of the system's output against real market data, without relying on subjective judgement about the quality of the prose.

Scenario Taxonomy. The case study comprises 23 investment scenarios organised into seven categories, each targeting a distinct combination of quantitative capabilities and financial reasoning competencies. Table 3 describes the taxonomy.

Table 3: Taxonomy of the 23 strategic investment scenarios in the case study.

Category	Description	Scenarios	Primary capabilities tested
Q1	Optimal Portfolio Construction	3	Markowitz optimisation, Sharpe maximisation, covariance estimation
Q2	Best Single Company Selection	3	Carhart Alpha estimation,[1] risk-adjusted screening, multi-criteria ranking
Q3	Long-Only Macro Strategy[2]	5	Macroeconomic scenario mapping, sector-to-asset translation, VaR computation
Q3b	Long/Short Macro Strategy[3]	5	Short-side instrument selection, correlation analysis, strategy structuring
Q4	Capital-Capped Exposure	3	Options strategy design, implied volatility,[4] loss-bounded portfolio construction
Q5	Risk-Adjusted Market Screening	1	Full-universe screening, multi-metric ranking, sector and stock recommendation
Q6	Minimum Variance Portfolio	1	MVP optimisation, covariance matrix, equally-weighted benchmark comparison
Q7	Retrospective Strategy Validation	1	Historical backtesting, Long/Short Sharpe Ratio, post-hoc attribution
Total		**23**	

Scenario Characteristics. Several design decisions were made deliberately to maximise the discriminative power of the evaluation.

Portfolio construction scenarios (Q1, Q6) specify a fixed reference date of December 31, 2024, ensuring that the temporal synchronisation mechanism of the Financial Engine is exercised and that the model cannot substitute memorised approximations for computed results. Scenarios in Q2 and Q5 require Carhart Alpha estimation, which demands multi-factor regression against real market benchmarks and cannot be plausibly recalled from training data. Categories Q3 and Q3b present paired macro views, for example a rising crude oil scenario (Q3) and its short-side counterpart (Q3b), which allows the evaluator to check internal consistency between the long and short strategy recommendations. The capital-capped scenarios (Q4) explicitly request options strategies calibrated to real implied volatility data, testing whether the system recognises the limits of its tool capabilities and responds appropriately rather than fabricating option pricing parameters. Finally, the retrospective scenario (Q7) is anchored in 2025 historical data, requiring the system to identify which sectors benefited from the decline in crude oil prices and to compute the realised Sharpe Ratio of the resulting long/short strategy.

Evaluation Criteria. Given the open-ended nature of the case study responses, automated evaluation alone is insufficient. Each scenario was assessed against four criteria, validated with the co-director:

1. **Data integrity**: all numerical values in the response are traceable to the `<observation>` returned by the Financial Engine. Any figure not present in the observation is considered a fabrication.

2. **Computational correctness**: portfolio weights sum to one, VaR values are consistent with the reported volatility, and Sharpe Ratios are coherent with the reported return and risk-free rate.

3. **Strategic coherence**: the investment recommendation is logically consistent with the stated macro view and does not contain contradictions, such as recommending a long position in an inverse ETF[5] or shorting an asset positively correlated with the target exposure.

[1]Carhart Alpha is a measure of risk-adjusted excess return that extends the standard CAPM model by accounting for four systematic market factors: market risk, company size, valuation, and momentum. A positive Alpha indicates that the asset delivered returns above what its market exposure alone would predict.

[2]A long-only strategy involves taking exclusively positive positions in assets expected to increase in value, without selling any asset short.

[3]A long/short strategy combines positive positions in assets expected to rise with negative (short) positions in assets expected to fall, allowing the investor to profit from both upward and downward price movements.

[4]Implied volatility is the market's expectation of future price variability for an asset, derived from the current market price of its options. It represents a forward-looking measure of risk, as opposed to historical volatility, which is computed from past returns.

4. **Operational completeness**: the response addresses all components requested in the scenario, including weights, expected return, volatility, VaR, and ranking, without omitting any required element.

A scenario is classified as fully correct only if it satisfies all four criteria simultaneously. Partial credit is not awarded. This strict binary criterion reflects the zero-tolerance requirement for fabricated data that motivates the architecture developed in this thesis: in a professional advisory context, a report that is partially correct but contains invented metrics cannot be considered reliable.

Comparison Configurations. The case study was run under three system configurations to isolate the contribution of each architectural component:

- **FinR1 Base** (`FinR1.py`): the foundational model without external tools, using greedy decoding and a single trajectory. This configuration establishes the performance floor and documents the pathologies described in Section 1.3.
- **FinR1 + Tools** (`FinR1Tools.py`): the tool-augmented architecture with the Deterministic Financial Engine, greedy decoding, and a single ReAct trajectory. This configuration isolates the contribution of the external engine from inference-time scaling.
- **FinR1 + Tools + TTC** (`FinR1ToolsTTC.py`): the complete proposed architecture with $N = 8$ parallel trajectories, probabilistic sampling ($T = 0.7$, $p = 0.90$), Reward Model scoring, and the Grounding Penalty mechanism. This is the final system whose performance is reported as the primary result of the thesis.

The progressive comparison across these three configurations provides a component-wise attribution of the accuracy gains, allowing the contribution of the Financial Engine and the TTC layer to be quantified independently and the incremental value of each architectural decision to be assessed.

[5] An inverse ETF is a fund designed to move in the opposite direction to its benchmark index. Taking a long position in an inverse ETF is already equivalent to a short bet, so shorting it would cancel out the intended exposure.

Chapter 5 Experimental Results

5.1 Overview of the Evaluation Protocol

This chapter presents the experimental results obtained across the two evaluation phases described in Section 4.7. The central question driving the evaluation is whether each architectural addition, first the Deterministic Financial Engine and then Test-Time Compute, produces a measurable and consistent improvement over the previous configuration. Results are presented following the same incremental progression used to motivate the architecture in Chapter 1: from the baseline model alone, to the tool-augmented system, to the complete proposed architecture.

The first evaluation phase assesses mathematical reasoning performance on the FinQA benchmark, which provides a quantitative and reproducible measurement of accuracy across 1,147 instances with single numerical answers. The second phase examines practical applicability through the 23-scenario strategic investment case study, where performance is evaluated through a combination of automated scoring and qualitative expert review.

Both phases compare the following three system configurations:

- **FinR1 Base**: the foundational model operating without external tools, using greedy decoding and a single generation trajectory. This configuration establishes the performance floor.
- **FinR1 + Tools**: the tool-augmented architecture incorporating the Deterministic Financial Engine, greedy decoding, and a single ReAct trajectory. This configuration isolates the contribution of external computation.
- **FinR1 + Tools + TTC**: the complete proposed architecture with $N = 8$ parallel trajectories, an external Reward Model that scores each candidate response, and the Grounding Penalty mechanism that eliminates trajectories containing unverified numerical values. This is the final system whose performance is reported as the primary result of the thesis.

Two accuracy metrics are reported throughout this chapter. **Acc/Total** measures the number of correct responses divided by the total number of instances in the dataset, including those for which the system failed to produce a parseable answer. **Acc/Evaluadas** measures the number of correct responses divided only by the instances for which a valid answer was successfully extracted, excluding cases where the output was empty or could not be parsed. The distinction matters primarily in the TTC experiments: the external Reward Model used to select the best trajectory occasionally assigned the highest score to an empty or malformed response, which Acc/Total penalises but Acc/Evaluadas does not. Reporting both metrics provides a more complete picture of system behaviour.

For the FinQA benchmark, answer correctness was assessed using `Qwen2.5-72B-Instruct` as an automated judge, replicating the evaluation methodology of the original FinR1 paper [3]

to ensure that the scores obtained here are directly comparable to previously published results. For the case study, performance estimates were produced using Gemini 3.1 Pro as an automated judge, complemented by the qualitative expert review of the thesis co-director Carlos Bellón Nuñez-Mera, whose comments on each scenario group are incorporated into the analysis. Given the open-ended nature of the case study responses, the accuracy figures reported for that phase should be interpreted as indicative estimates of relative performance between configurations rather than as absolute measurements.

5.2 Phase 1: FinQA Benchmark Results

5.2.1 Baseline Model Comparison

The first step of the experimental evaluation was to establish a performance baseline by comparing the three candidate reasoning models on the FinQA benchmark without any external tools or inference-time scaling. This comparison served two purposes: to identify the strongest baseline model for the subsequent experiments, and to quantify the performance floor from which the architectural improvements would be measured.

Table 4 reports the results for all three models under standard sampling and greedy decoding. Greedy decoding, which always selects the most probable next token and produces fully deterministic outputs, is included as a reproducibility reference.

Table 4: Baseline model comparison on FinQA (1,147 instances, no tools, no TTC).

Configuration	Tool Use (%)	Acc/Total (%)
FinR1	**0.0**	**17.0**
FinR1 Greedy	0.0	16.8
FinO1	0.0	10.3
FinO1 Greedy	0.0	11.0
QwQ Baseline	0.0	5.5

The results in Table 4 indicate that FinR1 achieved the strongest performance among the three candidates, with an Acc/Total of approximately 17% under both sampling and greedy decoding. FinO1 reached around 10–11%, while QwQ obtained only 5.5% under the strict single-answer evaluation conditions of FinQA. The negligible difference between the sampling and greedy variants of each model suggests that, under a single-trajectory regime without inference-time scaling, generation diversity contributes little to performance.

These results support the selection of FinR1 as the baseline model for the proposed architecture, as documented in Section 1.2. It is worth noting that the absolute accuracy values are modest, which is consistent with the difficulty of the FinQA benchmark and with the published results for similarly sized models operating without external tools [3].

The relevant comparison throughout this chapter is not the absolute value but the relative improvement achieved by each architectural addition.

5.2.2 Tool Integration Strategies

Having established FinR1 as the baseline model, the next experimental step was to evaluate different strategies for integrating external tool use. Three prompting and orchestration approaches were tested, each representing a different level of control over how the model interacts with the Financial Engine:

- **MiddleGround**: a few-shot prompting strategy that provides the model with examples of tool use but does not enforce a strict output format. The model is encouraged but not required to invoke tools.
- **Forced**: a strategy that attempts to compel the model to always generate a tool call, regardless of whether it judges one to be necessary.
- **ReAct**: the custom zero-abstraction orchestration protocol described in Section 4.3, which structures execution into a strict Thought, Action, Observation cycle and physically interrupts generation after each tool call.

Table 5 reports the results for each strategy on the full 1,147-instance FinQA split.

Table 5: Tool integration strategies for FinR1 on FinQA (1,147 instances).

Configuration	Tool Use (%)	Acc/Total (%)
FinR1 MiddleGround	**72.9**	13.3
FinR1 Forced	0.3	7.9
FinR1 ReAct	52.3	**34.4**

The results in Table 5 suggest that the choice of orchestration strategy has a substantial impact on performance. The MiddleGround approach achieved a high tool invocation rate of 72.9% but only 13.3% Acc/Total, indicating that frequent tool use does not translate into correct answers when the orchestration layer does not enforce strict output format compliance. The Forced strategy performed worst among the three, with an effective tool use rate of only 0.3% and an Acc/Total of 7.9%, suggesting that attempting to compel tool calls unconditionally disrupted the model's reasoning process without producing valid invocations.

The ReAct protocol produced the strongest result in the evaluated setting, achieving 34.4% Acc/Total with a tool use rate of 52.3%. This represents a doubling of Acc/Total relative to the FinR1 baseline of 17.0%, and suggests that the structured Thought, Action, Observation cycle described in Section 4.3 is more effective than less constrained approaches in this experimental context. The tool use rate of 52.3% reflects the fact that not all FinQA questions require external computation: for straightforward arithmetic questions, the model correctly determines that no tool call is needed and answers directly.

5.2.3 Pass@3 Validation and the Case for TTC

Before committing to the computational overhead of generating $N = 8$ parallel trajectories, a preliminary experiment was conducted to assess whether inference-time scaling was likely to yield meaningful improvements for FinR1. This experiment measured Pass@1 and Oracle Pass@3 on a subset of 500 instances. Pass@1 corresponds to standard single-attempt accuracy. Oracle Pass@3 measures whether the correct answer appears at least once among three independent generation attempts, regardless of which one is selected. It can be interpreted as an upper bound on what a perfect selection mechanism could achieve with three candidates.

Table 6 reports the results for FinR1 with and without tools.

Table 6: Pass@1 vs Oracle Pass@3 for FinR1 on a 500-instance subset of FinQA.

Configuration	**Pass@1 (%)**	**Oracle Pass@3 (%)**
FinR1 Base	31.8	44.0
FinR1 ReAct	43.8	**69.4**
FinR1 MiddleGround	**44.6**	65.8

The gap between Pass@1 and Oracle Pass@3 is consistent and substantial across all configurations evaluated. For FinR1 Base, the oracle Acc/Total increases from 31.8% to 44.0%, a relative gain of approximately 38%. For FinR1 ReAct, the gap is even larger: Pass@1 of 43.8% versus an oracle ceiling of 69.4%, a relative gain of approximately 58%. This suggests that in more than a quarter of the instances where the single-attempt result was incorrect, a correct answer was available among the three attempts but was not selected.

These results provide empirical support for exploring Test-Time Compute techniques, as described in Section 4.5. The model appears to possess the capacity to produce correct answers more often than single-shot evaluation suggests. The practical question, addressed in the following subsection, is whether an automated selection mechanism can recover a meaningful fraction of that latent potential.

5.2.4 TTC Selection Strategies on FinQA

With the Pass@3 results supporting the potential of inference-time scaling, the full TTC pipeline was evaluated on the complete 1,147-instance FinQA split. Table 7 reports the four most informative configurations, selected to illustrate the incremental contribution of each architectural component. Both Acc/Total and Acc/Evaluadas are reported to allow a complete interpretation of system behaviour, as defined in Section 5.1.

Table 7: Key TTC results on FinQA (1,147 instances, fixed seed SEED = 42). Acc/Total includes all instances; Acc/Evaluadas excludes instances where no valid answer was extracted.

Configuration	Tool Use (%)	Acc/Total (%)	Acc/Evaluadas (%)
FinR1 Base	0.0	27.6	27.7
FinR1 Base + BoN Vanilla (Skywork-LLaMA)	0.0	27.8	27.8
FinR1 Tool ReAct	52.3	34.4	53.6
FinR1 Tool ReAct + BoN Vanilla (Skywork-LLaMA)	**55.6**	**51.4**	**51.4**

Several observations follow from Table 7. First, applying TTC to the baseline model without tools produced virtually no improvement: Acc/Total increased from 27.6% to 27.8%, a difference within the margin of experimental noise. This is consistent with the expectation that generating multiple trajectories of a model that lacks access to real market data will not improve the factual accuracy of its outputs, since all trajectories draw from the same parametric knowledge.

Second, the tool-augmented ReAct configuration without TTC achieved 34.4% Acc/Total but 53.6% Acc/Evaluadas. The large gap between these two metrics reflects the high rate of instances where the model successfully invoked the engine but failed to produce a parseable boxed answer in the final report, contributing to the denominator of Acc/Total but not of Acc/Evaluadas. This formatting fragility under a single-trajectory regime motivates the introduction of TTC.

Third, combining ReAct tools with BoN Vanilla selection using the Skywork-LLaMA Reward Model increased Acc/Total from 34.4% to 51.4%, a gain of 17 percentage points. This represents the most substantial single improvement in Acc/Total observed in the FinQA experiments, and suggests that generating multiple trajectories and selecting the best one substantially reduces the impact of individual formatting failures in the evaluated setting.

An important caveat applies to the interpretation of these results. As discussed in Section 4.5.3, the $N = 8$ trajectories in the TTC configuration are generated through a single batched forward pass, which produces slightly different generation dynamics than a single isolated call due to differences in padding and attention mask handling. Consequently, the improvement observed between single-trajectory and TTC configurations reflects not only the benefit of having multiple candidates to select from, but also the effect of generating under a different inference regime. This distinction is discussed further in Section 5.2.5.

5.2.5 Reward Model Analysis and the TTC Paradox

The results in the previous subsection show that TTC with Reward Model-based selection substantially improves Acc/Total on FinQA when tools are active. However, a closer exami-

nation of the Reward Model scoring behaviour reveals an important limitation that qualifies this improvement.

To assess how well each Reward Model aligns with the true evaluation objective, the Pearson and Spearman correlations between the Reward Model score assigned to each trajectory and its binary correctness label were computed across all $N \times 1{,}147$ scored pairs. Table 8 reports these correlations for the three Reward Models evaluated, together with the mean score assigned to correct and incorrect responses.

Table 8: Reward Model correlation with correctness on FinQA (FinR1 + Tools + BoN Vanilla, 1,147 instances, $N = 8$ trajectories). Pearson and Spearman correlations are computed between the RM score and the binary correctness label across all scored trajectory pairs.

Reward Model	**Pearson** r	**Spearman** r	**Score (correct)**	**Score (incorrect)**
FsfairX-LLaMA3	**0.211**	0.203	−1.39	−2.91
Skywork-LLaMA	0.209	**0.221**	**−8.16**	**−12.28**
Gemma-27B	0.062	0.062	−6.72	−7.01

The correlations reported in Table 8 are statistically significant across all three models (all $p < 0.0001$), indicating that each Reward Model captures some signal about response quality. However, the magnitude of these correlations is low. FsfairX-LLaMA3 and Skywork-LLaMA reach Pearson r values of approximately 0.21, while Gemma-27B shows a substantially weaker correlation of only 0.06. For context, a correlation of 1.0 would indicate perfect alignment between the Reward Model score and actual correctness. The values observed here suggest that all three evaluators assign scores primarily based on factors such as linguistic fluency, response length, and formatting quality, rather than on mathematical correctness.

This misalignment between the selection metric and the true objective is referred to in this thesis as the *TTC Paradox*. The phenomenon can be described as follows: generating more trajectories increases the probability that at least one correct answer is produced, but the selection mechanism may still fail to identify it. A trajectory that is longer, more elaborately structured, or more linguistically polished may receive a higher Reward Model score even if it contains fabricated numerical values. The Ghost Data phenomenon discussed in Section 4.4.8 is a direct consequence of this dynamic: the Reward Model rewards the appearance of completeness, inadvertently favouring the generation of plausible-looking but unverified financial metrics.

The Grounding Penalty mechanism introduced in Section 4.4.8 was designed precisely to address this limitation. By assigning a score of −9,999 to any trajectory whose numerical outputs cannot be traced back to the engine observation, it ensures that factual grounding takes precedence over linguistic quality in the selection process. The practical impact of this mechanism on real-world advisory scenarios is examined in Section 5.3.

5.3 Phase 2: Strategic Investment Case Study

The FinQA benchmark results presented in Section 5.2 provide a quantitative measurement of mathematical reasoning performance under controlled conditions. However, as argued in Section 4.7.2, benchmark accuracy alone is insufficient to assess whether the system is ready for deployment in professional financial advisory. A model that correctly extracts arithmetic results from earnings reports is not necessarily capable of designing a coherent investment strategy, managing temporal context, or recognising the limits of its own tool capabilities.

This section presents the results of the 23-scenario strategic investment case study, evaluated through a combination of automated scoring using Gemini 3.1 Pro as a judge and qualitative expert review by the thesis co-director Carlos Bellón Nuñez-Mera. The three system configurations are compared following the same incremental progression established throughout this thesis: FinR1 Base, FinR1 + Tools, and FinR1 + Tools + TTC. For each configuration, both an overall accuracy estimate and a qualitative characterisation of the dominant failure modes are provided.

Table 9 summarises the estimated accuracy for each configuration across the 23 scenarios. As noted in Section 5.1, these figures should be interpreted as indicative estimates of relative performance rather than as absolute measurements, given the open-ended nature of the scenarios and the use of an automated judge for scoring.

Table 9: Summary of estimated accuracy across the 23-scenario case study, evaluated using Gemini 3.1 Pro as an automated judge and complemented by qualitative expert review. A scenario is classified as correct only if it simultaneously satisfies all four evaluation criteria defined in Section 4.7.2.

Configuration	**Scenarios correct**	**Estimated accuracy (%)**
FinR1 Base	6 / 23	27
FinR1 + Tools	15 / 23	65
FinR1 + Tools + TTC	18 / 23	78

The results in Table 9 suggest a consistent and substantial improvement at each architectural step. The following subsections analyse the specific failure modes and strengths of each configuration in detail, drawing on the qualitative comments provided by Carlos Bellón across the seven scenario groups.

5.3.1 FinR1 Base: Pathology Analysis

The baseline configuration, operating exclusively on the model's parametric knowledge without any external tools or inference-time scaling, achieved an estimated accuracy of approximately 27% across the 23 scenarios. While this figure indicates that the model produced structurally plausible responses in roughly a quarter of the evaluated cases, the expert review suggested that none of the numerical outputs could be verified against real market data,

since the model had no mechanism to retrieve or compute them from an external source. The following failure modes were consistently observed across scenario groups.

Date blindness and temporal inconsistency. Several scenarios explicitly anchored queries to a specific reference date, such as December 31, 2024, or June 1, 2024. The baseline model was observed to systematically ignore these temporal constraints in the evaluated scenarios, producing responses based on historical patterns from its training data rather than on market conditions corresponding to the requested date. This failure is particularly consequential for portfolio construction scenarios (Q1, Q6), where asset weights and performance metrics are highly sensitive to the specific data window used.

Fabricated metrics and absent data integrity. Across all scenario groups, the baseline model generated numerical values, including Sharpe Ratios, expected returns, volatilities, and portfolio weights, that could not be traced back to any real market data source. In the expert review, Carlos Bellón noted that responses to Q1 were "very generic and only say how it should be done", and that responses to Q2 included Carhart Alpha estimates that were judged to be implausible. In scenario group Q3b, the model selected Tesla based on several metrics including Carhart Alpha, which the expert considered highly unlikely to be correct. These observations are consistent with the sycophancy behaviour described in Section 1.3: the model tends to generate plausible-looking values to satisfy the structure of the request, regardless of whether they are grounded in reality.

Approximated portfolio weights. Rather than solving the Markowitz optimisation problem described in Section 4.4.5, the baseline model assigned simplified rounded weights to portfolio construction scenarios in the evaluated cases, a pattern consistent with the behaviour documented in Section 1.3. The expert review noted that responses in group Q6 used round percentage allocations rather than optimised weights derived from the realised covariance matrix.

Strategic reasoning without grounding. For open-ended macroeconomic scenarios (Q3, Q3b, Q7), the baseline model produced responses that were conceptually reasonable at a high level but lacked the quantitative grounding required for professional advisory. Carlos Bellón noted that responses to Q6 and Q7 were "conceptually more or less correct" and that "the final result makes sense", but that the calculations were based on historical means and volatilities rather than on the forward-looking information explicitly provided in the scenario. This reflects the model's tendency to apply learned statistical patterns even when the scenario provides a directional market view that should override them.

5.3.2 FinR1 + Tools: Improvement and Residual Failures

The tool-augmented configuration achieved an estimated accuracy of approximately 65% across the 23 scenarios, representing a substantial improvement over the baseline in the eval-

uated setting. The integration of the Deterministic Financial Engine substantially reduced the fabrication of asset identifiers and portfolio weights: in correctly answered scenarios, all numerical values were derived from real S&P 500 market data retrieved via `yfinance`. However, the expert review identified three categories of residual failure that limited further improvement.

Operational fragility. The most frequent source of failure in this configuration was syntactic rather than financial. As documented in Section 4.3, a single malformed JSON payload in the Action stage was sufficient to interrupt the reasoning process before the engine was invoked, causing the model to fall back to its parametric memory for the final report. Process hallucinations were also observed in several scenarios: the model generated text describing a tool call and its result as if both had occurred, when inspection of the execution logs confirmed that no actual tool invocation had taken place.

Tool capability boundaries. A second category of failure arose when scenarios requested metrics that fall outside the tool registry defined in Section 4.4.7. In particular, scenario groups Q2 and Q5 explicitly required Carhart Alpha estimation, a multi-factor regression metric not implemented in the Financial Engine. The expert review noted that in these cases the model "only uses Sharpe ratio" because "it says it does not have access to computing Carhart Alpha". While this response is operationally honest, it results in an incomplete answer that cannot satisfy the full set of evaluation criteria. A similar pattern was observed in scenario group Q4, where the model was asked to design options strategies calibrated to real implied volatility data: the engine does not expose an implied volatility tool, and the model acknowledged this limitation rather than fabricating values, which is appropriate behaviour but still results in an incomplete response.

Sector classification errors. In scenario group Q3, the expert noted that the tool-augmented model "gives better numbers but makes mistakes in choosing companies from the sector, including many technology companies" when asked about financial sector strategies. This reflects a limitation of the static universe dictionary described in Section 4.4.2, which covers only a subset of S&P 500 constituents and may not always align with the sector classification implied by the scenario.

5.3.3 FinR1 + Tools + TTC: Final Architecture

The complete proposed architecture achieved an estimated accuracy of approximately 78% across the 23 scenarios, representing a further improvement of 13 percentage points over the tool-augmented single-trajectory configuration in the evaluated setting. The most notable observed effect of the TTC layer was a substantial reduction in syntactic failures: by generating eight parallel trajectories and selecting among them, the system reduced the probability that all trajectories would fail due to JSON formatting errors simultaneously.

The expert review provided a more nuanced picture of the qualitative changes between this configuration and the previous one. For scenario group Q1, the response "does not give an answer" in some instances. This behaviour is attributable to cases where the Grounding Penalty, which eliminates trajectories containing fabricated numerical values not traceable to the engine observation, removed all candidate trajectories that had attempted to supplement the engine output with values from parametric memory. The result is a conservative but appropriate behaviour: the system withholds a response in the evaluated instances rather than fabricating one. For scenario group Q2, the expert noted that responses "give results that cannot be true", suggesting that the Reward Model occasionally selected trajectories containing unverified numerical values, referred to in this thesis as Ghost Data, that had evaded the Grounding Penalty check.

The TTC configuration showed its most notable improvement in scenario groups Q6 and Q7, where the multi-trajectory reasoning approach appeared to enable the model to produce more coherent long/short strategy recommendations and more structured portfolio analysis. The expert noted that Q7 responses "explain better" than the tool-only configuration, though both configurations shared the limitation of over-relying on historical volatility and return estimates rather than incorporating the forward-looking market view provided in the scenario.

Table 10 summarises the expert assessment for each scenario group across the three configurations.

Table 10: Expert assessment summary by scenario group across the three system configurations. A = FinR1 Base, B = FinR1 + Tools, C = FinR1 + Tools + TTC. Assessments are based on the qualitative review by Carlos Bellón Nuñez-Mera.

Group	**A: FinR1 Base**	**B: FinR1 + Tools**	**C: FinR1 + Tools + TTC**
Q1	Generic, no real data. Describes methodology without computing results.	Correct numerical outputs. Minor sector classification errors.	Does not produce an answer in some instances.
Q2	Implausible Carhart Alpha estimates.	Only uses Sharpe Ratio; acknowledges missing Carhart Alpha.	Same limitation as B; results occasionally implausible.
Q3	Conceptually reasonable; no real data.	Better numbers; incorrect sector composition.	Same sector limitation; does not complete answer.
Q3b	Fabricated metrics; incorrect asset selection.	Only uses Sharpe Ratio; picks plausible assets.	Same as B.
Q4	Plausible structure; unverifiable values.	Acknowledges missing implied volatility tool.	Only retrieves two sector assets.
Q6	Conceptually correct; rounded weights.	Incorrect asset class selection; unexplained calculation.	Same issues as B; better explanation.
Q7	Most reasonable qualitatively; uses historical data inappropriately.	Uses historical data instead of forward-looking view.	Same limitation; better structured response.

5.4 Summary and Cross-Phase Discussion

This section synthesises the results obtained across both evaluation phases and discusses the broader implications of the experimental findings in the context of the research question stated in Section 3.2: whether the integration of external tool use and Test-Time Compute techniques can significantly improve the reliability and mathematical accuracy of a financial reasoning model.

5.4.1 Progressive Accuracy Gains Across Configurations

Table 11 consolidates the key accuracy figures from both evaluation phases, allowing the contribution of each architectural component to be assessed side by side. The FinQA figures correspond to different experimental conditions across rows: the baseline and tool-augmented results were obtained under single-trajectory greedy decoding, while the TTC result was obtained under batched probabilistic sampling with a fixed seed. This difference in generation conditions is relevant to the interpretation of the improvements and is discussed in Section 5.4.3.

Table 11: Consolidated accuracy summary across both evaluation phases. FinQA figures report Acc/Total on 1,147 instances. Case study figures report estimated accuracy on 23 scenarios evaluated with Gemini 3.1 Pro as judge, complemented by expert review. FinQA baseline and tool results use single-trajectory greedy decoding; TTC result uses batched probabilistic sampling with SEED = 42.

Configuration	**FinQA Acc/Total (%)**	**Case Study (%)**
FinR1 Base	17.0	27
FinR1 + Tools (ReAct)	34.4	65
FinR1 + Tools + TTC	**51.4**	**78**

The results in Table 11 suggest that each architectural addition produced a consistent improvement across both evaluation dimensions. The integration of the Deterministic Financial Engine approximately doubled Acc/Total on FinQA, from 17.0% to 34.4%, and raised the estimated case study accuracy from 27% to 65%. The subsequent addition of TTC increased FinQA Acc/Total from 34.4% to 51.4% and the estimated case study accuracy from 65% to 78%. These results are broadly consistent with the hypothesis that separating language generation from deterministic computation, and then reducing dependence on a single generation attempt, produces meaningful and cumulative improvements in both controlled benchmark performance and practical advisory reliability, though the limitations discussed in Section 5.4.4 apply to this interpretation.

5.4.2 Consistency Between Evaluation Phases

A notable feature of the results is that the direction and relative magnitude of the improvements are consistent across the two evaluation phases, despite the substantial differences in their design. FinQA measures exact-match accuracy on single numerical answers from structured financial documents, while the case study measures the quality of open-ended investment recommendations across complex multi-step scenarios. The fact that the same architectural progression produced improvements in both settings suggests that the observed gains may reflect genuine improvements in the system's reasoning and computation capabilities, rather than artefacts of a specific evaluation design.

There is, however, one important asymmetry between the two phases. On FinQA, applying TTC to the baseline model without tools produced virtually no improvement, with Acc/Total increasing from 27.6% to 27.8%. This is consistent with the expectation that generating multiple trajectories of a model drawing exclusively from parametric memory does not improve factual accuracy, since all trajectories draw from the same underlying knowledge. On the case study, the baseline model achieved 27% estimated accuracy despite having no access to real data, which is higher than one might expect. This likely reflects the tendency of the automated judge to reward responses that were conceptually reasonable and well-structured, even when their numerical content could not be verified against real market data, as documented in Section 5.3.1.

5.4.3 The TTC Paradox in Context

The correlation analysis presented in Section 5.2.5 revealed that the Reward Models evaluated in this thesis explain only a small fraction of the variance in response correctness, with Pearson correlations ranging from $r = 0.062$ for Gemma-27B to $r = 0.211$ for FsfairX-LLaMA3. This finding has two practical implications relevant to both evaluation phases.

First, on FinQA, the improvement from TTC is partially attributable to the tendency of batched generation to produce slightly different outputs than sequential generation, a phenomenon described in Section 4.5.3, and not solely to the Reward Model's ability to identify the best trajectory among the candidates. The results in Table 7 showed that applying TTC to the baseline model without tools produced no meaningful improvement in Acc/Total, suggesting that the benefit of multiple trajectories may be conditional on having a reliable external computation layer to anchor the results.

Second, on the case study, the Ghost Data observations noted in Section 5.3.3 illustrate the TTC Paradox in its most concrete form. The Reward Model occasionally selected trajectories that were linguistically elaborate and well-formatted but contained fabricated numerical values, that is, values not derived from any engine observation, precisely the failure mode that the Grounding Penalty was designed to prevent. The fact that such trajectories were observed even with the Grounding Penalty active suggests that the mechanism, while effective in the evaluated scenarios, does not fully resolve the misalignment between linguistic quality and factual correctness as selection criteria.

5.4.4 Limitations of the Evaluation

Several limitations of the experimental design should be noted when interpreting the results presented in this chapter.

The FinQA results are subject to the batched inference caveat described in Section 4.5.3: the comparison between single-trajectory and TTC configurations is not a pure measurement of the selection benefit, since the two regimes differ in their generation dynamics. Future work could address this by evaluating TTC with sequential rather than batched generation, at the cost of substantially higher computational requirements.

The case study accuracy figures are estimates produced by an automated judge and should not be interpreted as ground-truth measurements. The qualitative review by Carlos Bellón provides a more reliable basis for assessing the practical quality of each configuration, but covers seven scenario groups rather than individual scenarios, and does not assign a numerical score that would allow precise cross-configuration comparison.

Finally, the tool registry described in Section 4.4.7 covers a deliberately limited universe of financial computations. The failure modes observed in scenario groups Q2, Q4, and Q5, where the system acknowledged missing capabilities rather than fabricating values, reflect deliberate design choices rather than fundamental limitations of the architecture. Expanding the tool registry to include Carhart Alpha estimation and implied volatility retrieval would be expected to address the capability gaps identified in the case study evaluation, and represents one of the most direct directions for future development.

Chapter 6 Conclusions and Future Work

6.1 Summary of Contributions

This thesis set out to investigate whether the structural limitations of financial reasoning models could be addressed by separating language generation from deterministic computation and by increasing robustness through inference-time scaling. The experimental results obtained across both evaluation phases suggest that this approach produces consistent and meaningful improvements, and that the three architectural components developed in this work each contribute independently to that outcome.

The first contribution of this thesis is the design and implementation of a custom Zero-Abstraction ReAct orchestration layer (O1). Rather than relying on general-purpose agentic frameworks, which were found to introduce context overhead, syntactic fragility, and process hallucinations when integrated with FinR1, the orchestration layer was built from scratch with direct control over the token stream. The five-layer cascade parser at its core showed sufficient robustness in the evaluated experiments to recover valid tool invocations from malformed outputs across a wide range of formatting deviations, enabling stable tool execution under probabilistic sampling conditions.

The second contribution is the Deterministic Financial Engine (O2), a Python backend that decouples numerical computation from language generation entirely. The engine provides five quantitative tools covering portfolio optimisation, market screening, asset metrics, risk estimation, and arithmetic computation, all grounded in real S&P 500 market data retrieved via `yfinance`. The integration of this engine was the single largest source of improvement observed in the evaluated experiments. On the FinQA benchmark, Acc/Total increased from 17.0% to 34.4%, approximately doubling the baseline figure. On the case study, the estimated accuracy, as assessed by the automated judge, increased from 27% to 65%. Beyond these figures, the engine substantially reduced the fabrication of asset identifiers and portfolio weights that had rendered the baseline model unsuitable for professional advisory use in the evaluated scenarios.

The third contribution is the Test-Time Compute pipeline (O3), which replaces single-trajectory generation with a branch-and-converge approach that produces $N = 8$ independent reasoning trajectories and selects the best candidate through a combination of Reward Model scoring and a novel Grounding Penalty mechanism. The TTC layer produced a further improvement of approximately 17 percentage points in FinQA Acc/Total, from 34.4% to 51.4%, and approximately 13 percentage points in the estimated case study accuracy, from 65% to 78%. Notably, the Grounding Penalty, which assigns a large negative score to trajectories containing numerical values not traceable to the engine observation, was designed to address the misalignment between linguistic quality and factual correctness that general-purpose Reward Models exhibit in quantitative financial tasks, a phenomenon identified and characterised in this thesis as the *TTC Paradox*.

The fourth contribution is the dual-phase evaluation framework (O4), which combines quantitative benchmarking on FinQA with a purpose-built case study of 23 strategic invest-

ment scenarios developed in collaboration with a FinTech domain expert. This evaluation design allows the contribution of each architectural component to be measured both under controlled benchmark conditions and in the kind of open-ended advisory scenarios for which the system was designed, providing a more complete picture of practical applicability than either evaluation phase alone would offer.

6.2 Answer to the Research Question

The central research question of this thesis, stated in Section 3.2, is whether the integration of external tool use and Test-Time Compute techniques can significantly improve the reliability and mathematical accuracy of a financial reasoning model. The experimental results obtained across both evaluation phases suggest that the answer is affirmative, with important qualifications.

The results indicate that the two strategies investigated address different and complementary aspects of the baseline model's limitations. Tool-augmented generation addresses the root cause of numerical unreliability: by delegating all quantitative computations to a deterministic external backend, the architecture prevents the language model from generating unverifiable numerical values regardless of how the query is formulated. In the evaluated experiments, this intervention produced the largest single accuracy gain observed, suggesting that the separation of language generation from mathematical computation is a necessary condition for reliable financial advisory performance.

Test-Time Compute addresses a different limitation: the dependence of the tool-augmented system on a single syntactically correct generation attempt. The results suggest that generating multiple independent reasoning trajectories and selecting among them substantially reduces the impact of individual formatting failures, recovering a meaningful fraction of the latent reasoning capacity that single-shot evaluation leaves unexploited, as evidenced by the Oracle Pass@3 results presented in Section 5.2.3. However, the effectiveness of TTC was found to be conditional on the quality of the selection mechanism. The correlation analysis in Section 5.2.5 indicated that general-purpose Reward Models explain only a small fraction of the variance in response correctness in quantitative financial tasks, which limits the net benefit of generating additional trajectories when the selection criterion is misaligned with the true objective.

Taken together, these findings suggest that the proposed architecture represents a meaningful step towards more reliable AI-assisted financial reasoning, while also identifying a specific and well-characterised bottleneck, the alignment between selection metrics and mathematical correctness, that remains open for future investigation.

6.3 Limitations

The results presented in this thesis were obtained within a well-defined experimental scope, and several aspects of the system design and evaluation methodology impose boundaries on the generalisability of the conclusions. These limitations are documented here not as

unresolved problems but as deliberate constraints that define the operating envelope of the current architecture.

Simplified tool registry. The Deterministic Financial Engine deployed in the final evaluation exposes five tools covering the most common quantitative operations required by the case study scenarios. A more advanced engine was developed and explored during the research process, incorporating Carhart four-factor Alpha estimation, implied volatility retrieval via Black-Scholes inversion, and full Fama-French factor regressions. However, the richer outputs produced by this engine increased the complexity of the model's parsing task to a degree that compromised reliable tool execution under probabilistic sampling conditions in the evaluated experiments. The simplified engine was therefore adopted as the stable configuration for the final evaluation, with the understanding that certain scenario groups, particularly Q2, Q4, and Q5, could not be fully addressed within its capabilities.

Automated evaluation of the case study. The accuracy figures reported for the 23-scenario case study are estimates produced by Gemini 3.1 Pro as an automated judge, complemented by qualitative expert review. While the qualitative review by Carlos Bellón Nuñez-Mera provides a reliable basis for assessing the direction and character of the improvements, assigning a precise numerical score to open-ended investment recommendations is inherently difficult even for a domain expert, since responses may be partially correct, strategically sound but computationally incomplete, or correct in their conclusions but flawed in their justification. For this reason, the case study accuracy figures should be interpreted as indicative estimates of relative performance between configurations rather than as absolute measurements.

Batched inference and generation comparability. As discussed in Section 4.5.3, the single-trajectory and TTC configurations differ not only in the number of candidates generated but also in their generation dynamics, since batched forward passes produce slightly different outputs than sequential calls even under a fixed random seed. This means that the accuracy gains observed between the tool-augmented single-trajectory configuration and the TTC configuration reflect a combination of the selection benefit and a change in the underlying generation distribution, and cannot be attributed exclusively to the selection mechanism.

Reward Model alignment. The correlation analysis in Section 5.2.5 indicated that the general-purpose Outcome Reward Models evaluated in this thesis, models trained to assess linguistic quality and instruction following rather than mathematical correctness, explain only a small fraction of the variance in response correctness for quantitative financial tasks, with Pearson correlations ranging from $r = 0.062$ to $r = 0.211$. This misalignment between the selection metric and the true objective limits the effectiveness of Best-of-N selection as a standalone improvement mechanism, and represents the most significant structural limitation of the current TTC pipeline.

6.4 Future Work

The architecture developed in this thesis provides a functional foundation for AI-assisted financial reasoning, and the experimental results suggest that its core design principles are sound. The following lines of future work represent natural extensions of this foundation rather than corrections of fundamental problems.

6.4.1 Process Reward Models for Financial Reasoning

The most direct path to resolving the TTC Paradox identified in Section 5.2.5 is the development or adaptation of Process Reward Models (PRMs) specifically designed for quantitative financial tasks. Unlike the Outcome Reward Models evaluated in this thesis, which assign a single score to the complete response, PRMs assess the quality of individual reasoning steps throughout the generation process. In a financial context, this would allow the evaluator to verify, for example, whether the model correctly identified the relevant tool before invoking it, whether the observation was properly incorporated into the reasoning chain, and whether the final numerical values are consistent with the intermediate steps that produced them.

Recent work such as FinPRM [19] has begun to explore this direction specifically for financial reasoning tasks. At the time this research was conducted, publicly available implementations of domain-specific financial PRMs were limited, with many remaining proprietary or lacking accessible model weights. As this area matures, integrating a financial PRM into the Best-of-N selection pipeline described in Section 4.6 would be expected to address the correlation gap between selection scores and mathematical correctness observed in the evaluated experiments, potentially unlocking a larger fraction of the latent reasoning capacity evidenced by the Oracle Pass@3 results.

6.4.2 Advanced Tool Integration

The simplified tool registry adopted in the final evaluation was a pragmatic response to a specific integration challenge: the richer outputs produced by the advanced `FinancialEngine`, which incorporated Carhart four-factor Alpha estimation, implied volatility retrieval via Black-Scholes inversion, and full Fama-French factor regressions, increased the complexity of the model's parsing task to a degree that compromised reliable execution under probabilistic sampling conditions. The architecture itself, however, is designed to accommodate additional tools without modification to the orchestration layer.

A promising direction for future work is the investigation of output serialisation strategies that preserve the analytical richness of advanced financial computations while keeping the observation payload within the parsing capacity of the model under TTC conditions. Structured summarisation of complex outputs, progressive disclosure of results across multiple tool calls, that is, returning results in smaller and more digestible chunks across successive tool invocations rather than in a single complex payload, or the use of a secondary parsing model dedicated to interpreting rich engine outputs are among the approaches that could be explored. If successful, such strategies would allow the system to address the scenario groups

that remained incomplete in the current evaluation, particularly those requiring Carhart Alpha estimation (Q2, Q5) and implied volatility calibration (Q4), without sacrificing the syntactic reliability that the simplified engine provides.

6.4.3 Broader Applicability

The design principles underlying the proposed architecture are not specific to financial reasoning. The core idea, separating language generation from deterministic computation and using inference-time scaling to reduce dependence on individual generation attempts, is applicable to any domain where numerical accuracy and data traceability are critical requirements and where the cost of fabricated outputs is high.

Domains such as legal analysis, clinical decision support, and engineering assessment share these characteristics: they require precise quantitative computations, they are sensitive to the temporal validity of data, and they operate under regulatory or safety constraints that make hallucinated outputs unacceptable. The architecture developed in this thesis, and in particular the combination of the Zero-Abstraction ReAct protocol, the deterministic backend, and the Grounding Penalty mechanism, provides a reusable template that could be adapted to these domains by replacing the Financial Engine with a domain-specific computation layer while keeping the orchestration and selection infrastructure intact.

Chapter 7 References

[1] Z. Chen et al., "FinQA: A dataset of numerical reasoning over financial data", in *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, 2021, pp. 3697–3711.

[2] Z. Chen, S. Li, C. Smiley, Z. Ma, S. Shah, and W. Y. Wang, "ConvFinQA: Exploring the chain of numerical reasoning in conversational finance", in *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, 2022, pp. 6279–6292.

[3] Z. Wang et al., "Finr1: Financial reasoning models via reinforcement learning", *Working Paper/Preprint (SUFE-AIFLM-Lab)*, 2025.

[4] M. Sharma et al., "Towards understanding sycophancy in language models", *arXiv preprint arXiv:2310.13548*, 2023.

[5] H. Chase, *Langchain: Building applications with llms through composability*, `https://github.com/langchain-ai/langchain`, 2022.

[6] S. Yao et al., "React: Synergizing reasoning and acting in language models", *arXiv preprint arXiv:2210.03629*, 2022.

[7] C. Snell, J. Lee, K. Xu, and A. Kumar, "Scaling llm test-time compute optimally can be more effective than scaling model parameters", *arXiv preprint arXiv:2408.03314*, 2024.

[8] T. Loughran and B. McDonald, "When is a liability not a liability? textual analysis, dictionaries, and 10-ks", *The Journal of Finance*, vol. 66, no. 1, pp. 35–65, 2011.

[9] D. Araci, "Finbert: Financial sentiment analysis with pre-trained language models", *arXiv preprint arXiv:1908.10063*, 2019.

[10] S. Wu et al., "Bloomberggpt: A large language model for finance", *arXiv preprint arXiv:2303.17564*, 2023.

[11] H. Yang, X.-Y. Liu, and C. D. Wang, "Fingpt: Open-source financial large language models", *arXiv preprint arXiv:2306.06031*, 2023.

[12] Q. Team, "Qwen: Technical report and the qwq reasoning model", *arXiv preprint*, 2024. [Online]. Available: `https://qwenlm.github.io/`.

[13] Y. Su et al., "Fino1: Financial reasoning models via logic-driven reinforcement learning", *Working Paper/Preprint*, 2024.

[14] T. Schick et al., "Toolformer: Language models can teach themselves to use tools", *arXiv preprint arXiv:2302.04761*, 2023.

[15] Hugging Face Agent Team, *Smolagents: A tiny library to build production-grade agents*, `https://github.com/huggingface/smolagents`, 2024.

[16] X. Wang et al., "Self-consistency improves chain of thought reasoning in language models", *arXiv preprint arXiv:2203.11171*, 2022.

[17] J. Huang et al., "Large language models cannot self-correct reasoning yet", *arXiv preprint arXiv:2310.01798*, 2023.

[18] W. Dong et al., "Fsfairx-llama3-rm-v0.1: A robust and aligned open-source reward model", *Working Paper / Model Release*, 2024. [Online]. Available: `https://huggingface.co/sfairXC/FsfairX-LLaMA3-RM-v0.1`.

[19] Z. Luo et al., "Finprm: Evaluating and enhancing financial reasoning capabilities of large language models via process reward models", *Working Paper/Preprint*, 2024.

[20] E. Maliach et al., "Ai agents in financial services: A review of opportunities, risks, and governance frameworks", *Journal of Financial Regulation and Compliance*, 2024.

[21] D. Amodei, C. Olah, J. Steinhardt, J. Christiano Paul Schulman, and D. Mané, "Concrete problems in ai safety", *arXiv preprint arXiv:1606.06565*, 2016.

[22] T. Wolf et al., "Huggingface's transformers: State-of-the-art natural language processing", *arXiv preprint arXiv:1910.03771*, 2020.

[23] T. Dao, "Flashattention-2: Faster attention with better parallelism and work partitioning", *arXiv preprint arXiv:2307.08691*, 2023.

[24] R. P. Brent, "An algorithm with guaranteed convergence for finding a zero of a function", *The Computer Journal*, vol. 14, no. 4, pp. 422–425, 1971.

[25] H. Markowitz, "Portfolio selection", *The Journal of Finance*, vol. 7, no. 1, pp. 77–91, 1952.