MANCHESTER 1824
The University of Manchester

RobustAgent — Intelligent Mixed-Model Control Design via Large Language Models

A dissertation submitted to The University of Manchester for the degree of
Bachelor of Engineering in Electrical and Electronic Engineering
in the Faculty of Science and Engineering

Year of submission
2026

Student ID
11366201

School of Engineering

Contents

Word count: 8388

Abstract

This dissertation presents RobustAgent, a framework based on large language models (LLMs) that combines mixed-agent coordination with integration across multiple application programming interfaces (APIs) for automated controller synthesis and evaluation in control engineering. The architecture enables parallel reasoning across heterogeneous model interfaces (OpenAI `gpt-4o` [1], Anthropic `claude-sonnet-4` [2], DeepSeek `deepseek-chat` [3], and Google `gemini-pro` [4]) through a LangGraph workflow. RobustAgent employs system-specific sub-agents for first-order stable and unstable systems, systems with delay, second-order stable and unstable systems, and higher-order models. It selects a single system-specific sub-agent that proposes candidate controller parameters and brief analytical summaries for the given dynamics, while a Central Agent runs closed-loop simulation and picks the best feasible configuration.

To improve robustness assessment beyond phase and gain margin alone, the proposed framework introduces stability margin as a complementary performance metric. RobustAgent uses a mixed-API evaluation scheme in which parallel reasoning across heterogeneous LLM interfaces reduces single-model bias and improves the consistency of controller proposals across plant representations. Linear-matrix-inequality (LMI)–based feedback stability verification supplies asymptotic stability checks for continuous-time linear state-space models.

Experiments on benchmark control problems showed that RobustAgent applied stricter verification than a single-model ControlAgent baseline (generator `deepseek-chat`). Accepted designs met LMI feasibility and hard multi-metric thresholds more consistently, at the cost of a lower raw automated success rate (82.3% versus 92.0%). Controllers that passed basic pole-based checks but violated stability margin, phase margin, rise time, or integral of time-weighted absolute error (ITAE) limits were rejected. Taken together, the results supported an extensible mixed-agent, mixed-API workflow for LLM-based controller design that emphasises verified robustness over headline pass rates.

Code is available at `https://github.com/JiaruiWang-79/Robustagent`.

Declaration of originality

I hereby confirm that no portion of the work referred to in the thesis has been submitted in support of an application for another degree or qualification of this or any other university or other institute of learning.

Copyright statement

i The author of this thesis (including any appendices and/or schedules to this thesis) owns certain copyright or related rights in it (the "Copyright") and s/he has given The University of Manchester certain rights to use such Copyright, including for administrative purposes.

ii Copies of this thesis, either in full or in extracts and whether in hard or electronic copy, may be made *only* in accordance with the Copyright, Designs and Patents Act 1988 (as amended) and regulations issued under it or, where appropriate, in accordance with licensing agreements which the University has from time to time. This page must form part of any such copies made.

iii The ownership of certain Copyright, patents, designs, trademarks and other intellectual property (the "Intellectual Property") and any reproductions of copyright works in the thesis, for example graphs and tables ("Reproductions"), which may be described in this thesis, may not be owned by the author and may be owned by third parties. Such Intellectual Property and Reproductions cannot and must not be made available for use without the prior written permission of the owner(s) of the relevant Intellectual Property and/or Reproductions.

iv Further information on the conditions under which disclosure, publication and commercialisation of this thesis, the Copyright and any Intellectual Property and/or Reproductions described in it may take place is available in the University IP Policy (see `http://documents.manchester.ac.uk/DocuInfo.aspx?DocID=24420`), in any relevant Thesis restriction declarations deposited in the University Library, The University Library's regulations (see `http://www.library.manchester.ac.uk/about/regulations/`) and in The University's policy on Presentation of Theses.

Acknowledgments

I am grateful to my supervisor, Dr Chao Chen, for his guidance on research direction, control-theoretic background, and experimental design throughout this project. His feedback on stability margins, phase margins, transient metrics, and thesis structure substantially improved both the implementation and the presentation of the work.

I thank the University of Manchester for access to the facilities, library resources, and computing services that supported the project.

I also acknowledge the authors of ControlAgent [5], whose mixed-agent, mixed-API framework and published baseline configuration provided a clear point of comparison for this dissertation.

1 Introduction

Large language models (LLMs) have evolved from text-generation systems into models capable of planning, tool use, and structured problem decomposition. LLM-based agents have been deployed and have demonstrated strong performance in areas such as code generation and data analysis. However, when considering control engineering, the limitations of current LLM-based methods become apparent. Control design involves balancing performance while maintaining stability and robustness. The growing interest in leveraging LLMs for control design is reflected in recent IEEE-published discussions on the topic [6]. In contrast, existing systems tend to generate one-shot solutions rather than support iterative design processes.

As a multi-step and tightly coupled discipline, control engineering requires modelling, stability analysis, controller design, optimisation, and validation to form a closed decision loop. While MATLAB and Python-Control are established environments with strong analytical capabilities, their effective use depends on expert human interpretation and manual design iteration. This raises the question of how general engineers without specialised control expertise can reliably execute this workflow. Recent agent-based platforms, such as ControlAgent [5], demonstrate that some of these steps can be automated, but several structural shortcomings remain. Single-model pipelines exhibit two structural limitations. Synthesis and validation are carried out by the same model, which weakens independent verification of constraints. In addition, robustness assessment relies primarily on phase margin and settling time. Such criteria are adequate for nominal plants but inadequate when model uncertainty or time delay is present.

To address these limitations, this dissertation proposes RobustAgent for automated controller design as a mixed-agent framework that orchestrates parallel calls to multiple application programming interfaces (APIs) for heterogeneous large language models. Multiple backends are deployed in parallel to generate candidate controllers through an iterative process organised using LangGraph. Tasks are allocated based on different types of system dynamics, such as stable and unstable first-order systems, time-delay models, second-order systems, and higher-order plants. Each task-specific agent generates both controller parameters and a corresponding analytical explanation.

The candidates are assessed using deterministic metrics: Stability Margin, Phase Margin, Rise Time, and ITAE (Integral of Time-weighted Absolute Error). Accordingly, controller selection is based on quantitative evaluation rather than model preference. For stability verification, RobustAgent adopts a standard and computationally efficient linear-matrix-inequality (LMI)–based approach, yielding a systematic certification procedure based on Lyapunov feasibility rather than pole-location checks alone.

The key contributions of this dissertation are summarised as follows:

- This dissertation proposes RobustAgent for controller synthesis, in which several large language models operate in parallel to generate candidate controllers. This parallelism reduces reliance on any single model and enlarges the pool of feasible design options.
- This dissertation evaluates candidate controllers using four quantitative metrics—stability margin (SM), phase margin (PM), rise time, and the integral of time-weighted absolute error (ITAE)—to jointly capture robustness and transient performance, thereby avoiding subjective model-based selection.
- This dissertation replaces conventional pole-location checks with a Lyapunov-based LMI condition for closed-loop stability verification. The resulting convex optimisation formulation provides rigorous certification of asymptotic stability, particularly for higher-order systems where pole computation can be numerically unreliable.
- This dissertation describes a graph-structured workflow that links controller design, evaluation, and verification agents. The workflow supports iterative refinement and maintains a traceable record of each design attempt, improving reproducibility of the overall synthesis process.

In summary, the proposed framework integrates LLM-based agents into the controller design process as computational tools rather than independent problem solvers. Control theory provides the analytical criteria and constraints, while LLM agents generate candidate controller designs for evaluation.

Accordingly, the aim of this dissertation is to assess whether mixed-agent, mixed-API LLM workflows can reduce the manual effort required for structured controller design when classical metrics and convex stability tests supply the acceptance logic.

2 Literature Review

2.1 Classical Control Foundations

Controller design has traditionally been treated as a case-specific task [7], as the appropriate control strategy often depends heavily on the characteristics of the controlled system. Among

various control methods, PID control and loop-shaping design remain the most widely used technologies due to their conceptual simplicity and ease of implementation [8], [9].

Loop shaping methods typically operate in the frequency domain, achieving the desired bandwidth, disturbance rejection, and tracking performance by adjusting the open-loop transfer function [7], [10]. However, such methods also reveal a fundamental trade-off in system performance: increasing bandwidth can enhance the system's response speed, but it also amplifies noise; reducing high-frequency gain can enhance noise suppression, but at the cost of slower system response [11]. In practice, controller design still largely relies on engineers' experience and judgment to balance multiple competing design goals.

The PID controller remains the most common solution in industrial control [9]. Its parameters have intuitive physical meanings: the proportional term affects the response speed, the integral term eliminates steady-state errors, and the differential term anticipates the system behaviour [12], [13]. Over the past few decades, researchers have proposed numerous PID and loop-shaping tuning methods [7], [14], [15], [16]. Despite these advancements, controller tuning still largely relies on the experience of human experts and manual adjustments to obtain controller parameters that meet design requirements.

2.2 Performance Metrics in Classical Control

Beyond the design methods themselves, classical control engineering has developed a set of well-established performance metrics. These metrics are used both to define design requirements and to judge whether a controller performs satisfactorily. They are usually grouped into two categories: frequency-domain metrics and time-domain metrics, each reflecting different aspects of closed-loop behaviour [17].

In the frequency domain, gain margin (GM) and phase margin (PM) are the most commonly used indicators of robustness. They describe how much variation in gain or phase the system can tolerate before becoming unstable. In practice, both can be read directly from the Bode plot of the open-loop transfer function [17]. Another useful measure is the stability margin (SM), defined as the inverse of the peak sensitivity $\|S(j\omega)\|_\infty$. Unlike GM and PM, SM provides a single value that reflects how close the Nyquist plot gets to the critical point $(-1, j0)$. This makes it more reliable in systems with multiple crossover frequencies, where GM and PM alone may give an incomplete picture [17].

In the time domain, performance is typically evaluated using the step response. Key indicators include rise time, settling time, overshoot, and steady-state error. Among these, rise time gives

a direct sense of how quickly the system reacts to a change in the reference input. To capture overall tracking performance, integral criteria are often introduced. One widely used example is the Integral of Time-weighted Absolute Error (ITAE), which is standard in classical treatments of transient performance [17]. ITAE assigns a higher penalty to errors that persist later in the response, making it particularly suitable for optimisation-based PID tuning [8].

It is widely recognised that no single metric can fully describe system performance. Frequency-domain measures such as PM or SM focus on robustness, but they do not directly reflect tracking quality. In contrast, time-domain metrics like rise time or ITAE describe transient behaviour, yet they do not guarantee robustness on their own. As a result, practical controller design typically involves considering multiple metrics together. The trade-offs that arise—such as those between speed and overshoot, or bandwidth and noise sensitivity—must often be balanced through experience and iterative tuning. This is why a multi-metric evaluation framework is adopted in the proposed framework, as discussed in Section 3.4.

2.3 Automated Controller Synthesis

Automated controller synthesis has long been an active research area in control engineering, and its development has extended from early PID self-tuning strategies to modern learning-based methods. Early studies, such as the pioneering Ziegler–Nichols method [14], mainly relied on empirical rules to adjust controller parameters based on the response of the plant. Subsequently, research gradually shifted to search-based optimisation methods, where controller parameters are tuned by minimising a predefined cost function through an iterative optimisation process. Derivative-free optimisation algorithms can handle non-convex cost functions, but their high computational complexity and lack of physical interpretability often hinder adoption in practical engineering applications.

In contrast, analytical and rule-based methods, such as loop-shaping equations, frequency-domain design rules, and symbolic derivation methods, can provide clearer performance guarantees. However, the effective application of these methods often relies on expert experience and requires manual adjustment according to system requirements. In recent years, this research direction has gradually developed a hybrid approach, combining classical control theory with prior knowledge from machine learning. For instance, reinforcement learning is utilised for strategy optimisation, feedforward compensation is achieved through learning methods, and neural networks are integrated into the Model Predictive Control (MPC) framework. These modern methods aim to combine the structural stability advantages of classical control with the adaptability and flexibility provided by machine learning.

2.4 LLMs and Mixed-Agent AI for Control Engineering

The rapid development of large language models (LLMs) has created new possibilities for advanced reasoning, mathematical derivation, and code generation in engineering applications. Benchmark studies of frontier models on control-engineering tasks [18] catalogue behaviours such as proposing PID-related parameterisations, performing elementary symbolic manipulation, and drafting state-space descriptions, while complementary work documents inconsistencies, hallucinations [19], and the absence of built-in formal guarantees.

Recent benchmarking studies have evaluated the capabilities of leading LLMs (GPT-4, Claude 3 Opus, Gemini 1.0 Ultra) on control engineering tasks, identifying both strengths and weaknesses in mathematical reasoning and system analysis [18]. Mixed-agent architectures, including AutoGen [20], LangGraph [21], and other distributed LLM reasoning frameworks, have been proposed. These approaches enable parallel exploration, role-based task allocation, and iterative refinement.

Recent surveys on LLM-based agent assessment emphasise the need for multi-dimensional evaluation metrics beyond basic success rates, highlighting the importance of robust verification and stability guarantees [22]. Recent work has applied LLMs to system identification, simulation code synthesis, and performance analysis, but controller synthesis remains constrained by subjective model reasoning and the lack of rigorous verification.

Zahedifar et al. [23] introduce LLM-Agent-Controller, a universal multi-agent LLM framework that incorporates dedicated auxiliary agents for controller synthesis, model reasoning, control analysis, and simulation. Li et al. [24] present S2C, a specification-to-certified-controller framework that combines LLM agents with linear-matrix-inequality (LMI)–based H_∞ synthesis and report strong synthesis outcomes on COMPleib benchmarks under their stated evaluation protocol via a multi-agent architecture with iterative refinement.

To address these limitations, Guo et al. [5] introduce ControlAgent, a mixed-agent framework in which a central agent distributes system descriptions to heterogeneous model interfaces (e.g., OpenAI, Claude, Gemini). System-specific sub-agents specialise in handling first-order, second-order, delay, and higher-order linear systems. A specialised Evaluation and Decision agent performs closed-loop simulation and robustness analysis—using classical margins, sensitivity measures, and LMI-based stability verification—to identify the best controller among competing designs. By combining traditional control theory with LLM-based agent orchestration, this system offers a structured, verifiable pipeline for automated controller synthesis.

3 Methodology

This section describes the methodology used to develop RobustAgent, the proposed mixed-agent framework for automated controller synthesis and evaluation with parallel calls to multiple application programming interfaces (APIs). For clarity, the methodology is presented in four stages: (i) system architecture and mixed-agent modules, together with LangGraph-based workflow orchestration; (ii) linear-matrix-inequality (LMI)–based stability certification replacing pole-based checks; (iii) performance metrics for robustness and transient quality; and (iv) weighted scoring for controller selection.

3.1 Network Graph Orchestration

Figure 1 shows the complete architecture of RobustAgent. The system consists of a Central LLM Agent that coordinates with task-specific sub-agents for different system types (first-order stable/unstable, second-order stable/unstable, time-delay, and higher-order systems). The iterative design loop incorporates dual-API parallel design, a Computation Agent, Design Memory, Performance evaluation, and Feedback Generation. Domain expertise modules provide loop-shaping and PID controller knowledge. The workflow terminates when requirements are met or maximum iterations are reached, outputting the final controller design.

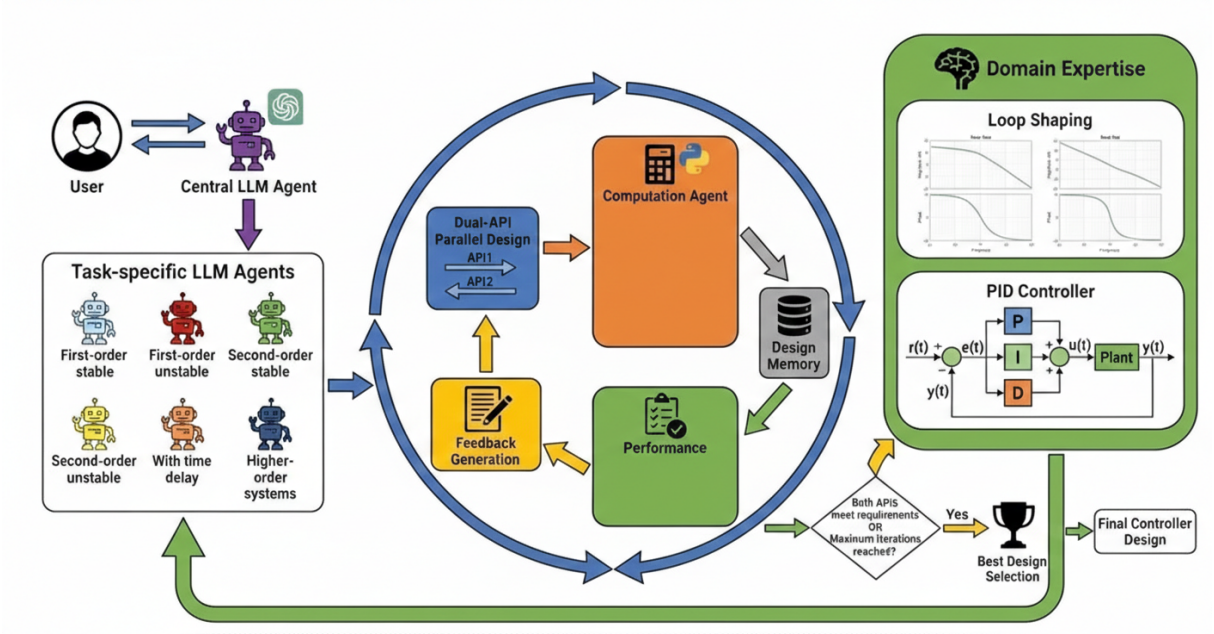


Fig. 1. RobustAgent architecture.

RobustAgent comprises four integrated modules. The **Central Agent** functions as the main

coordination unit of the system. It receives control design tasks described by users through natural language and parses system information, such as transfer functions or state-space representations. Subsequently, this module extracts key design performance metrics, including stability margin, dynamic response speed, and comprehensive error performance, rather than relying solely on classic specifications such as settling time or phase margin. On this basis, the Central Agent further determines the most suitable controller type and assigns tasks to the corresponding sub-agents. This centralised coordination mechanism reduces interpretation discrepancies that may occur when multiple language models independently interpret the same design objective.

Task-Specific Controller Agents are responsible for generating specific controller designs. Each agent is tailored to a specific system type and control structure, such as designing a PID-based loop-shaping controller for a stable first-order system. When an agent receives the structured task description, it uses the control knowledge embedded in its prompts for reasoning, proposes controller parameters, and generates initial candidate designs that satisfy the basic theoretical constraints. These agents can also operate in parallel across multiple large language model platforms. This parallel mechanism increases the diversity of design schemes and reduces the system's reliance on a single model.

The **Computation and Evaluation Module** performs deterministic numerical calculations through Python-based control libraries. Its main functions include simulating closed-loop systems, calculating time-domain and frequency-domain performance metrics, and verifying the stability and robustness of the system. Since all analyses are carried out by deterministic algorithms, the selection of controllers is based on objective calculation results rather than subjective judgements generated by language models.

The **Structured Design Memory** module is used to record all intermediate results during the controller design process. This memory module stores the controller parameters, performance indicators, and evaluation results in each iteration, while retaining both successful and failed design attempts. When a candidate solution is rejected, the system provides clear diagnostic feedback to the corresponding sub-agent, thereby guiding the next round of design. In addition, this mechanism also prevents repeated exploration of parameter regions that have been proven ineffective. Overall, this memory structure emulates the iterative optimisation process used by human control engineers, while ensuring the traceability and reproducibility of the entire design process.

3.2 LangGraph-Based Workflow Orchestration

LangGraph is an open-source agent orchestration framework built on top of LangChain for constructing and managing complex LLM workflows [21]. Compared with a purely linear pipeline, LangGraph adopts a graph-based computational architecture to model the relationships, dependencies, and feedback loops within an agent-based AI system. It models an AI workflow as a directed graph consisting of three primitives:

- **Nodes**, which are computation units or agents such as the LLM reasoning module, tool-calling component, evaluation component, or retrieval subcomponent;
- **Edges**, which represent transitions between nodes, meaning that the next computation step depends on the current system state;
- **State**, which is a shared memory object that persists across runs and stores intermediate results, metadata, and contextual information.

Stateful execution is supported in this model, allowing each node to access and modify shared information. State management improves traceability, interpretability, and verification of computations, making the reasoning process more transparent. Unlike strictly acyclic pipelines, LangGraph supports cyclic graph structures, enabling iterative optimisation and feedback loops. This feature is essential for agent systems that require self-correction, performance reassessment, or human-in-the-loop interaction.

LangGraph is selected for RobustAgent for two primary reasons. First, the workflow follows a generate–evaluate–feedback–regenerate cycle. LangGraph natively supports cyclic execution, making it well suited to represent this reflective optimisation process. Second, multiple controller candidates, stability margins, ITAE scores, and other performance metrics must be stored and compared. The centralised state mechanism of LangGraph enables all performance-related information to be shared across components, allowing the Central Agent to maintain a comprehensive global view of all candidate controllers.

Figure 2 provides a compact pseudocode view of the iterative LangGraph loop (routing, dual-API proposal, deterministic evaluation, and feedback-driven refinement). An end-to-end demonstration of the iterative design process is provided in Appendix C.

Algorithm 1: LangGraph-Based Iterative Controller Design in RobustAgent

Input: Plant $\mathcal{S}$ (num/den, optional τ); request $\mathcal{U}$; thresholds $\mathcal{T}$ (stability/phase margin, rise time, ITAE); $N_{\max}$; dual LLM configs.
Output: Bundle $\mathcal{C}$: parameters, requirement flags, winner, scores, iterations, success.

$\mathcal{M}_1 \leftarrow \emptyset$, $\mathcal{M}_2 \leftarrow \emptyset$ `// per-API design_memory buffers`
`// Central LLM routing (select_subagent)`
Parse $(a, \theta_{\text{task}}) \leftarrow$ SelectSubagent($\mathcal{U}$) `// subagent index` $a \in \{1, \ldots, 6\}$`, refined task text`
for $k \leftarrow 1$ **to** $N_{\max}$ **do**
  `// Dual prompts: subagent template + task + per-API memory (_get_design_prompt, feedback_prompt)`
  $\mathcal{P}_1^k \leftarrow$ GenPrompt($a, \theta_{\text{task}}, \mathcal{M}_1$)
  $\mathcal{P}_2^k \leftarrow$ GenPrompt($a, \theta_{\text{task}}, \mathcal{M}_2$)
  `// Parallel LLM completion (parallel_design)`
  $(\text{resp}_1, \text{resp}_2) \leftarrow$ ParallelLLM($\mathcal{P}_1^k, \mathcal{P}_2^k$)
  Parse controller parameters $(\omega_L, \beta_b, \beta_l)$ from each JSON response
  `// Closed-loop metrics via Python + control (_compute_design_performance, util)`
  $P_1^k \leftarrow$ EvalDesign($\mathcal{S}, \text{params}_1$), $P_2^k \leftarrow$ EvalDesign($\mathcal{S}, \text{params}_2$)
  $\mathcal{M}_1 \leftarrow \mathcal{M}_1 \cup \{(\text{params}_1, P_1^k)\}$, $\mathcal{M}_2 \leftarrow \mathcal{M}_2 \cup \{(\text{params}_2, P_2^k)\}$
  if *at least one of* P_1^k, P_2^k *satisfies* $\mathcal{T}$ *(`_check_requirements`)* **then**
    $\mathcal{C} \leftarrow$ SelectBest(designs, $\mathcal{T}$) `// compare_designs / score_design`
    return $\mathcal{C}$
$\mathcal{C} \leftarrow$ SelectBest(last designs, $\mathcal{T}$) `// stopped after` $N_{\max}$ `without either API meeting` $\mathcal{T}$
return $\mathcal{C}$

Fig. 2. LangGraph-based iterative controller design loop in RobustAgent.

3.3 LMI-Based Stability Verification

A core function of the Computation Agent is to verify the closed-loop stability of each candidate controller and ensure system stability before performance evaluation. Unlike relying solely on pole positions for higher-order closed-loop systems, RobustAgent employs a linear matrix inequality (LMI) analysis based on Lyapunov stability theory to verify stability [25].

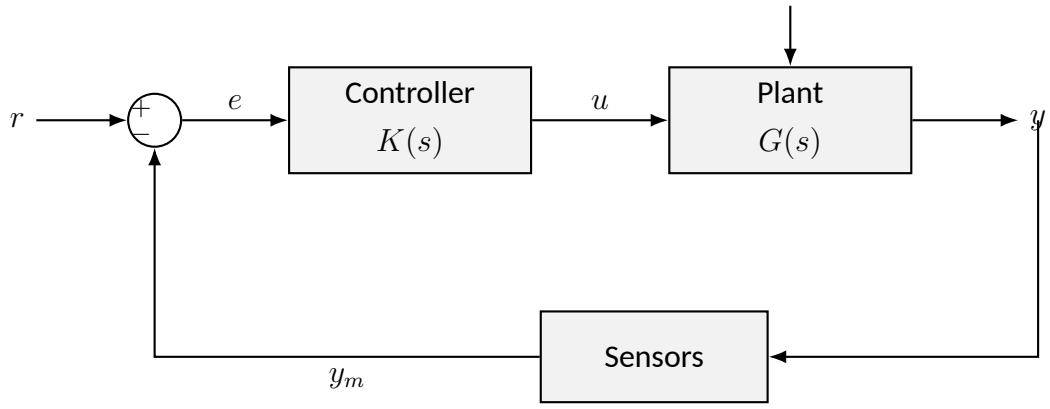


Fig. 3. Unity-feedback interconnection for LMI-based verification.

For a given plant $G(s)$ and controller $K(s)$, RobustAgent constructs the closed-loop system and converts it into a state-space representation $(A_{\text{cl}}, B_{\text{cl}}, C_{\text{cl}}, D_{\text{cl}})$:

$$\dot{x}(t) = A_{\text{cl}}x(t) + B_{\text{cl}}r(t), \qquad y(t) = C_{\text{cl}}x(t) + D_{\text{cl}}r(t). \tag{1}$$

For linear time-invariant models, internal (asymptotic) stability of the closed loop is determined solely by the autonomous dynamics $\dot{x}(t) = A_{\text{cl}}x(t)$. According to the Lyapunov direct method for linear systems [17], $\dot{x}(t) = A_{\text{cl}}x(t)$ is asymptotically stable if and only if there exists a symmetric matrix $P = P^\top$ such that

$$P \succ 0, \qquad A_{\text{cl}}^\top P + P A_{\text{cl}} \prec 0. \tag{2}$$

Therefore, the stability verification problem is transformed into an LMI feasibility problem in convex optimisation:

$$\begin{aligned} &\text{find} && P = P^\top \\ &\text{subject to} && P \succeq \epsilon I, \quad A_{\text{cl}}^\top P + P A_{\text{cl}} \preceq -\epsilon I, \end{aligned} \tag{3}$$

The strict matrix inequalities in (2) are replaced by non-strict inequalities offset by a small $\epsilon > 0$ as in (3), which converts the open feasibility set into a numerically tractable closed one without altering the asymptotic stability conclusion. This optimisation problem was modelled by CVXPY [26] and solved using the SCS solver [27]. Since the main objective is to obtain a feasible stability proof rather than the optimal solution, feasibility is the primary acceptance criterion. The candidate controller is accepted only when the solver returns a feasible matrix P, thereby proving asymptotic stability.

To enhance the reliability of the automated pipeline, a pole-based fallback test is also included. If the state-space transformation or LMI solver fails, RobustAgent reverts to the classic stability check and accepts the controller only when all closed-loop poles satisfy

$$\Re(p_i) < -\delta, \qquad \forall i, \tag{4}$$

where $\delta > 0$ imposes a uniform strictly negative margin on $\Re(p_i)$. When $\delta = 0$ (up to numerical tolerance), this reduces to requiring Hurwitz stability of A_{cl}; for $\delta > 0$, the same condition is an α-stability (exponential stability) requirement with decay rate at least δ. The strictly negative bound $-\delta$ ensures a minimum exponential decay rate, providing a numerical safety margin against floating-point errors near the imaginary axis. This dual-verification mechanism combines the rigour of Lyapunov theory with the practical reliability of the classical pole criterion, enabling RobustAgent's synthesis loop to strike a balance between theoretical rigour and practical feasibility. Figure 4 summarises the verification path from closed-loop construction to LMI-based acceptance and pole-based fallback.

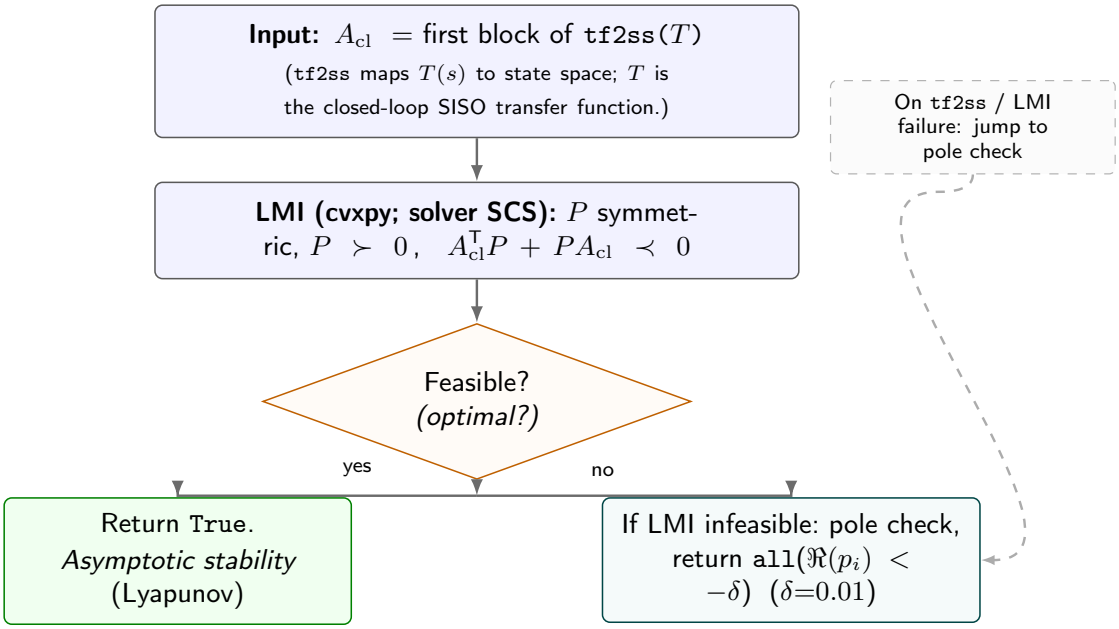


Fig. 4. LMI-based stability certification workflow.

3.4 Performance Metrics

A sound performance assessment should integrate stability metrics with transient-response metrics. RobustAgent employs four performance metrics: stability margin (SM), phase margin (PM), rise time (T_r), and the integral of time-weighted absolute error (ITAE). The following subsections define each of these metrics and explain why they are necessary. The definitions follow standard control engineering textbooks [17].

3.4.1 Stability Margin (SM)

The Stability Margin is the primary robustness metric in RobustAgent. It is defined in terms of the sensitivity function $S(j\omega)$ as

$$M_s = \|S(j\omega)\|_\infty = \max_\omega \frac{1}{|1 + L(j\omega)|}, \qquad \text{SM} = \frac{1}{M_s}, \tag{5}$$

where $L(j\omega)$ is the open-loop (loop) transfer function and M_s is the peak sensitivity. For the standard unity-feedback configuration, $L(s) = K(s)G(s)$; more generally, if a sensor/measurement dynamics $H(s)$ is included in the feedback path, then $L(s) = K(s)G(s)H(s)$. The

Stability Margin can be interpreted as a measure of how close the Nyquist plot of the open-loop transfer function comes to the critical point $(-1,\, j0)$, and as a direct measure of the maximum sensitivity magnitude. The main advantage of using SM instead of PM is that it takes into account the entire frequency range. Phase Margin only examines the point on the Nyquist plot associated with the gain crossover frequency, so it does not provide a complete picture of system stability. Some complex or higher-order systems have Nyquist plots that pass close to the critical point at multiple frequencies, a phenomenon known as conditional stability. In such cases, the Phase Margin may appear satisfactory while the system is actually close to instability. Stability Margin avoids this blind spot by measuring the minimum distance to the critical point over the full frequency range, thus providing a more stringent and comprehensive robustness assessment. The specific threshold is defined per task via `stability_margin_min`; in the benchmark suite it takes values of either 0.2 or 0.3 (median 0.3), with representative lower bounds for the four system families listed in Table 1.

3.4.2 Phase Margin (PM)

Phase Margin is a second robustness indicator, in addition to Stability Margin. It is defined at the gain crossover frequency ω_{gc}, where $|L(j\omega_{\text{gc}})| = 1$, as

$$\text{PM} = 180^\circ + \angle L(j\omega_{\text{gc}}). \tag{6}$$

Figure 5 illustrates the gain crossover point and the corresponding phase margin definition.

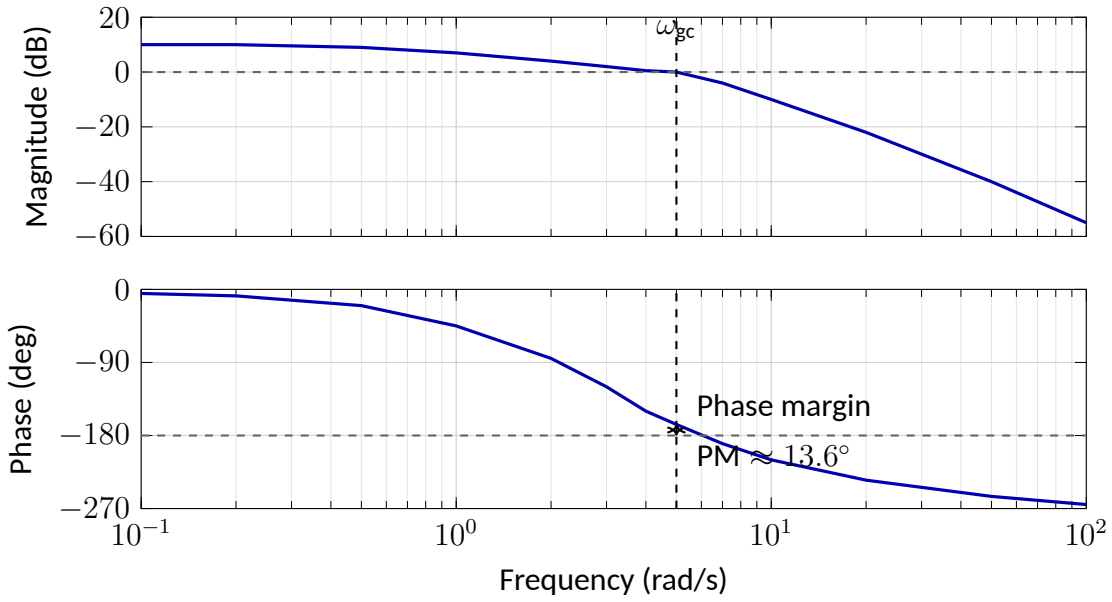


Fig. 5. Phase margin at the gain crossover frequency.

Phase Margin therefore measures the additional phase lag at the gain crossover frequency that can be tolerated before the system becomes unstable, and hence provides an intuitive frequency-domain indicator. Although it is one-sided, it remains useful for discriminating dynamical behaviour in more complex systems. Phase Margin can serve as a cross-check: controllers that satisfy the Stability Margin criterion but exhibit unusually low phase margins may be operating close to the robustness boundary at the crossover frequency. The specific threshold is defined per task via `phase_margin_min`; across the benchmark suite it ranges from approximately 10.5° to 89.8° (median 56.9°), with representative lower bounds for the four system families listed in Table 1.

3.4.3 Rise Time (T_r)

Rise Time is defined as the time required for the output to increase from 10% to 90% of its final value. Although closely related to settling time, it is treated as a separate metric because it captures transient features that settling time alone can miss. Rise Time is closely tied to closed-loop bandwidth and is constrained by the available control authority. An excessively long Rise Time may therefore indicate that the controller lacks sufficient gain or actuation headroom to drive the system promptly, even when steady-state behaviour is acceptable. Settling time, on the other hand, can be misleading on its own: a controller with a short settling time may still exhibit a sluggish initial response or a highly oscillatory transient that happens to decay quickly. Imposing a constraint on Rise Time ensures that the early portion of the response reaches a meaningful level within the required time, rather than relying on settling time alone to characterise transient behaviour. Figure 6 shows the 10%–90% rise-time definition used in this dissertation.

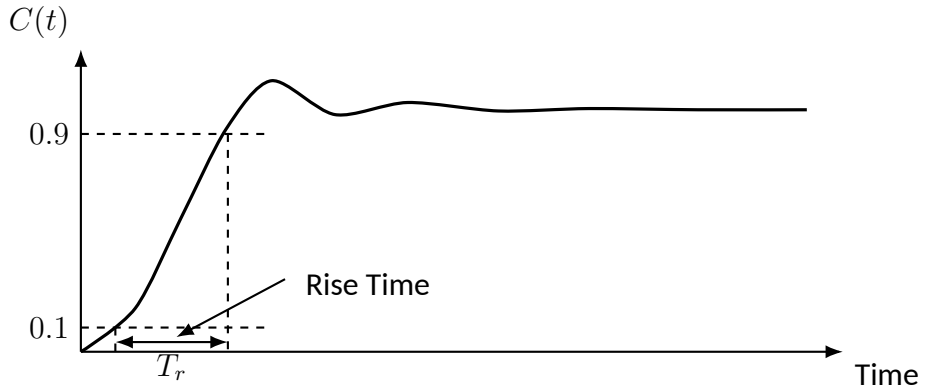


Fig. 6. Rise time definition (10%–90%).

The specific interval is defined per task as a per-task performance envelope via

`rise_time_min` and `rise_time_max`. A numerical floor of 1 ms is applied to `rise_time_min` to avoid specification artefacts from idealised continuous-time models. Across the benchmark suite, $T_{r,\text{min}}$ spans 0.001 s to 37.1 s (median 0.048 s) and $T_{r,\text{max}}$ spans approximately 0.017 s to 105 s (median 1.27 s), with representative intervals for the four families listed in Table 1.

3.4.4 ITAE

The Integral of Time-weighted Absolute Error (ITAE) is defined as

$$\text{ITAE} = \int_0^T t\,|e(t)|\,\mathrm{d}t, \tag{7}$$

where $e(t)$ is the tracking error. Unlike settling time, which is a single-event measure, and overshoot, which captures only peak deviation, ITAE evaluates the entire transient trajectory. A controller that reaches steady state through oscillatory behaviour will incur a significantly larger ITAE than one that converges smoothly. Therefore, ITAE favours well-damped and rapid responses compared with relying on a single simple time-domain metric alone.

The specific cap is defined per task via `ITAE_max`; among the 483 tasks with finite caps, values range from approximately 1.89×10^{-4} to 3.09×10^{5} (median 1.45), with representative upper bounds for the four families listed in Table 1.

3.5 Controller Scoring and Weighting Scheme

After all candidate controllers have been evaluated against the four metrics defined in Section 3.4, the Evaluation Agent ranks them using a weighted composite score. The criteria have different physical dimensions—unitless for SM, degrees for PM, seconds for T_r, and units of $[\text{s} \times \text{error}]$ for ITAE—and opposite optimisation directions: larger SM and PM are preferred, whereas smaller T_r and ITAE are preferred. Each criterion is therefore mapped to a common score scale where 100 consistently represents the best outcome.

For each metric, a sub-score $S \in [0, 100]$ is computed via a piecewise function $f(x)$ with three properties: (i) the specification boundary yields $S = 50$; (ii) values that exceed the specification are rewarded with additional points that saturate at 100; and (iii) values that violate the specification are penalised with a linear reduction toward zero in proportion to the extent of nonconformity. This construction rewards safety margin while avoiding overly conservative dominance by any single metric.

Robustness metrics (lower bounds). For $\mathrm{SM} \geq \mathrm{SM}_{\min}$ and $\mathrm{PM} \geq \mathrm{PM}_{\min}$,

$$S_{\mathrm{SM}} = \begin{cases} 50 + \min\left(50,\, 50 \cdot \dfrac{\mathrm{SM} - \mathrm{SM}_{\min}}{\max(\mathrm{SM}_{\min},\, 0.1)}\right), & \mathrm{SM} \geq \mathrm{SM}_{\min}, \\ \max\left(0,\, 50 \cdot \dfrac{\mathrm{SM}}{\mathrm{SM}_{\min}}\right), & \mathrm{SM} < \mathrm{SM}_{\min}, \end{cases} \tag{8}$$

$$S_{\mathrm{PM}} = \begin{cases} 50 + \min\left(50,\, 50 \cdot \dfrac{\mathrm{PM} - \mathrm{PM}_{\min}}{\max(\mathrm{PM}_{\min},\, 1)}\right), & \mathrm{PM} \geq \mathrm{PM}_{\min}, \\ \max\left(0,\, 50 \cdot \dfrac{\mathrm{PM}}{\mathrm{PM}_{\min}}\right), & \mathrm{PM} < \mathrm{PM}_{\min}. \end{cases} \tag{9}$$

Rise time (admissible band). Rise time uses a two-sided specification $T_{r,\min} \leq T_r \leq T_{r,\max}$. Responses outside the band are penalised on either side, while responses inside the band receive full credit that decreases linearly from 100 at $T_{r,\min}$ to 50 at $T_{r,\max}$:

$$S_{T_r} = \begin{cases} 50 + 50\left(1 - \dfrac{T_r - T_{r,\min}}{T_{r,\max} - T_{r,\min}}\right), & T_{r,\min} \leq T_r \leq T_{r,\max}, \\ \max\left(0,\, 50 \cdot \dfrac{T_r}{T_{r,\min}}\right), & T_r < T_{r,\min}, \\ \max\left(0,\, 50 \cdot \dfrac{T_{r,\max}}{T_r}\right), & T_r > T_{r,\max}. \end{cases} \tag{10}$$

ITAE (upper bound). ITAE uses an upper-bound specification $\mathrm{ITAE} \leq \mathrm{ITAE}_{\max}$, with smaller values strictly preferred:

$$S_{\mathrm{ITAE}} = \begin{cases} 50 + 50\left(1 - \dfrac{\mathrm{ITAE}}{\mathrm{ITAE}_{\max}}\right), & \mathrm{ITAE} \leq \mathrm{ITAE}_{\max}, \\ \max\left(0,\, 50 \cdot \dfrac{\mathrm{ITAE}_{\max}}{\mathrm{ITAE}}\right), & \mathrm{ITAE} > \mathrm{ITAE}_{\max}. \end{cases} \tag{11}$$

The four sub-scores are combined into a weighted composite score:

$$\mathrm{Score} = \frac{\sum_m w_m\, S_m}{\sum_m w_m} \;\in\; [0, 100], \qquad m \in \{\mathrm{SM},\, \mathrm{PM},\, T_r,\, \mathrm{ITAE}\}. \tag{12}$$

The weights are set as

$$w_{\mathrm{SM}} = 0.30, \quad w_{\mathrm{PM}} = 0.30, \quad w_{T_r} = 0.20, \quad w_{\mathrm{ITAE}} = 0.20. \tag{13}$$

If a candidate fails the closed-loop stability check of Section 3.3, its composite score is set to

zero regardless of the sub-scores, enforcing stability as a strict prerequisite rather than a trade-off. The rationale for the weighting reflects a classical engineering priority ordering: **stability before performance**. SM and PM jointly account for 60% of the composite score, while the remaining 40% is split equally between rise time and ITAE to balance early transient speed and overall trajectory quality.

The Evaluation Agent selects the candidate with the highest composite score among those that pass all hard acceptance thresholds (closed-loop stability, minimum SM, minimum PM, T_r within the admissible band, and maximum ITAE). If no candidate satisfies all constraints, the agent initiates a new iteration of the design loop, providing quantitative feedback to the task-specific agents on which criteria were violated and by how much.

3.6 Experimental Setup

Benchmark systems. The benchmark is a four-category subset of ControlEval re-instrumented with the new-metrics bundle defined in Section 3.4. Table 1 summarises one representative plant from each of four families—first-order stable, second-order stable, first-order with delay, and higher-order—together with the four acceptance thresholds (stability margin SM, phase margin ϕ_m, rise time T_r, and ITAE) and the response-speed class implied by the T_r band. The dataset preserves the original ControlEval structure, but every sample has been re-encoded with SM and ITAE thresholds in addition to the legacy phase-margin and response-speed specifications, so that every accepted controller satisfies a strictly stronger set of constraints than the original ControlAgent benchmark.

Evaluation metrics. Following Guo et al. [5], the primary metric is Automated Success Rate (ASR), defined as the proportion of benchmark instances for which the framework returns a controller satisfying every acceptance threshold within the iteration budget. A run that exhausts the iteration cap without satisfying all thresholds contributes zero to ASR even if intermediate designs were partially compliant. The average iteration count is reported alongside ASR as an efficiency indicator: lower is better, with the natural floor of one iteration corresponding to a single-shot successful design.

Iteration budget. The maximum number of iterations is configured by task difficulty, mirroring the ControlAgent convention: 10 iterations for stable first- and second-order systems, 20 iterations for systems with delay and for unstable first- and second-order systems, and 30 iterations for higher-order systems. Within each pair, the two parallel API calls share the same budget, so the wall-clock cost grows linearly with iteration count rather than with the number

	1st-order stb	2nd-order stb	1st-order w/ delay	Higher-order system
System model	$\frac{19.95}{s+0.390}$	$\frac{5.88}{s^2+1.43s+0.91}$	$\frac{8.79}{s+4}\,\mathrm{e}^{-0.144s}$	$\frac{225}{s^3+14.2s^2+46s+40}$
Stability margin	SM ≥ 0.3	SM ≥ 0.3	SM ≥ 0.3	SM ≥ 0.2
Phase margin (ϕ_m)	$\phi_m \geq 71.54°$	$\phi_m \geq 61.57°$	$\phi_m \geq 44.06°$	$\phi_m \geq 62.54°$
Rise time (T_r)	$T_r \in [0.84,\ 5.17]\,\mathrm{s}$	$T_r \in [4.82,\ 12.93]\,\mathrm{s}$	$T_r \in [0.22,\ 2.34]\,\mathrm{s}$	$T_r \in [0.42,\ 3.36]\,\mathrm{s}$
Response speed (from T_r spec)	*moderate*	*slow*	*moderate*	*moderate*
ITAE	ITAE ≤ 0.030	ITAE ≤ 17.36	ITAE ≤ 10.83	ITAE ≤ 19.14

Table 1. System models and their corresponding control design criteria.

of APIs.

LLM hyperparameters. All four backends are queried through their public chat-completion endpoints. To minimise stochastic variation between repeated runs, sampling temperature is set to 0 wherever the provider permits it (`gpt-4o`, `deepseek-chat`); for the two providers that require a strictly positive temperature (`gemini-pro`, `claude-sonnet-4`), the lowest stable value of 1 is used in line with the ControlAgent configuration. Maximum response length is capped at 1024 tokens for OpenAI, DeepSeek, and Anthropic backends, and at 8192 tokens for Gemini to accommodate its more verbose default style.

4 Results and Discussion

This section reports the experimental assessment of RobustAgent, the proposed mixed-agent, mixed-API framework. The presentation is organised so that the reader first sees how the framework operates on a single, concrete example (Section 4.1), before being shown aggregate results across the full benchmark (Section 4.2), the role of stricter acceptance criteria (Section 4.3), and the contribution of each component of the pipeline (Section 4.4). The chapter closes with an integrated discussion (Section 4.5) and a statement of present limitations (Section 4.6).

All runs used the following fixed chat-model API identifiers: OpenAI `gpt-4o`, DeepSeek

`deepseek-chat`, Google `gemini-pro`, and Anthropic `claude-sonnet-4`. All six pairwise combinations of these four backends were assessed over ten system categories ranging from first-order (stable-fast, stable-moderate, stable-slow, unstable, with delay), second-order (stable-fast, stable-moderate, stable-slow, unstable), to higher-order dynamics. The main assessment criterion was the **Automated Success Rate (ASR)**, defined as the percentage of benchmark problems for which RobustAgent produced a controller that met all acceptance thresholds within the maximum iteration budget. The average number of design iterations needed to find a successful solution was also reported as a secondary measure of efficiency. Full experimental details are provided in Section 3.6.

4.1 Worked Example (Single Task)

Before moving onto aggregate results, this subsection illustrates the interaction between the LangGraph pipeline in Section 3.2 and the metrics defined in Section 3.4 using an individual design case. This follows the example used to demonstrate the controller-design functionality in ControlAgent [5, Fig. 3], but replaces the original phase-margin/settling-time metric pair with the new four-metric acceptance bundle (SM, ϕ_m, T_r, and ITAE).

Plant and specification. A representative stable first-order plant taken from the benchmark suite is

$$G(s) = \frac{5.6}{s + 3.4},$$

with constraints SM ≥ 0.3, $\phi_m \geq 70^\circ$, $T_r \leq 0.12$ s, and ITAE ≤ 0.05. The user prompt conveys these criteria along with the rise-time class they imply.

Central routing. The Central Agent interpreted the request, recognised the plant as a first-order stable system, and forwarded the task to Subagent 1 (loop-shaping proportional–integral (PI) controller design). The routing decision and processed task description were emitted in structured JSON form, which the LangGraph state passed to the parallel design node.

Iteration 1. In both APIs, the initial design recommended a loop bandwidth of $\omega_L = 6$ rad/s with $\beta_b = \sqrt{10} \approx 3.162$, leading to a large phase margin $\phi_m \approx 102^\circ$ and an effectively infinite stability margin (for this simple case). However, the closed-loop step response was too slow: $T_r \approx 0.68$ s and ITAE ≈ 0.14, both exceeding their limits. The performance checker logged the parameters and metrics in the API-specific design memory buffer and issued the feedback *"T_r too large, ITAE too large $\Rightarrow$ raise ω_L"*.

Iteration 2. With this feedback in mind, the next round increased the loop bandwidth to

$\omega_L = 10\,\text{rad/s}$. The step response became much faster ($T_r \approx 0.23\,\text{s}$, ITAE ≈ 0.014) and satisfied the bound on the integrated error. The phase margin remained high at $\phi_m \approx 91°$, but T_r still exceeded the limit of $0.12\,\text{s}$; hence, the check failed and the graph returned to the design node.

Iteration 3. Boosting the loop bandwidth further to $\omega_L = 22\,\text{rad/s}$ reduced T_r to $0.08\,\text{s}$, within the rise-time bound; the phase margin lowered slightly to $81°$ (still above the threshold $70°$); ITAE ≈ 0.005, well below the limit; and the linear matrix inequality (LMI) check of Section 3.3 returned a Lyapunov certificate. The four conditions were satisfied simultaneously, so the router terminated the iteration cycle and accepted the controller.

Figure 7 provides a visual representation of the entire trajectory of step response, open-loop magnitude and phase, and the four-metric verdict throughout the three iterations.

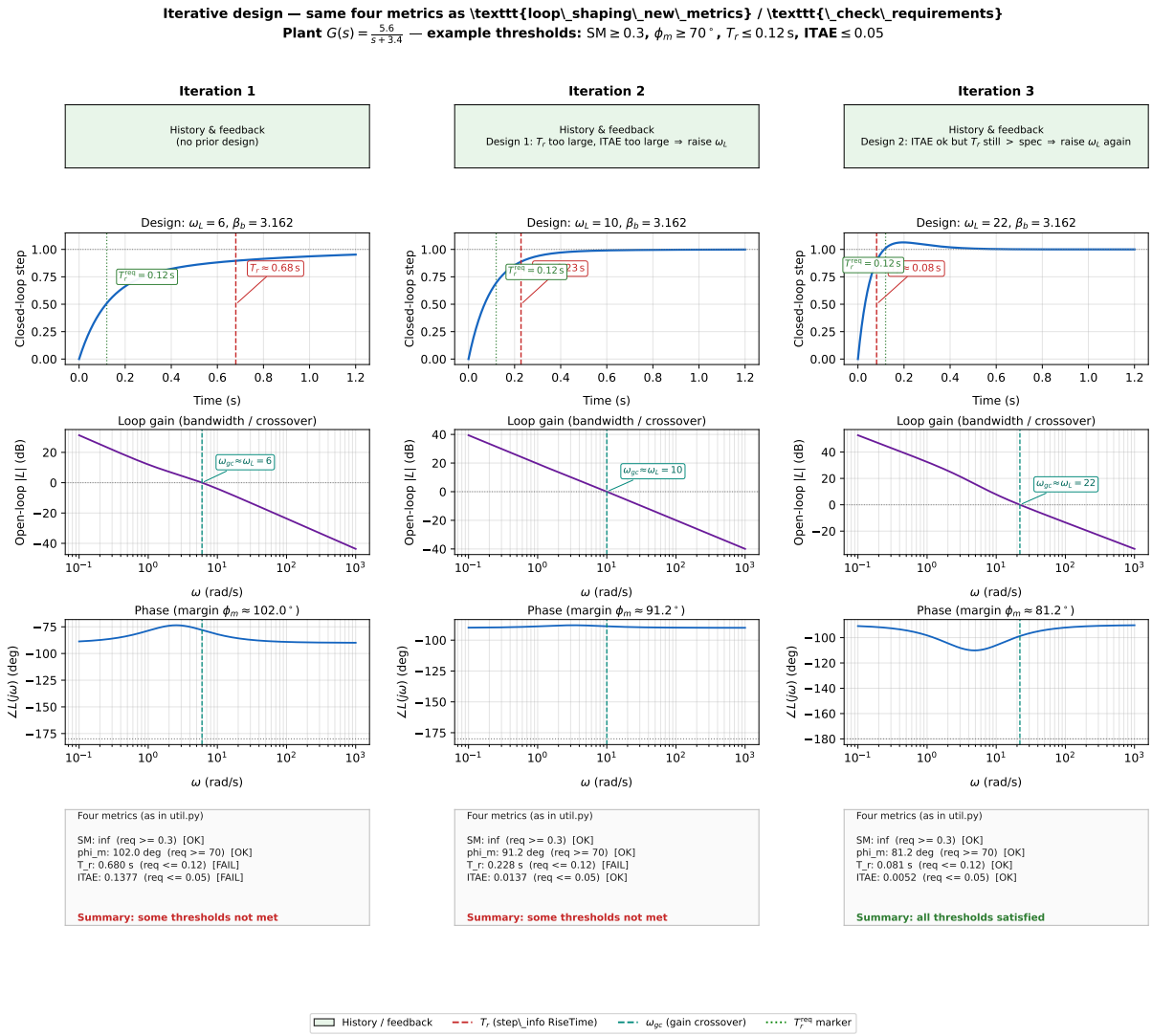


Fig. 7. Worked example: three-iteration refinement for $G(s) = 5.6/(s + 3.4)$.

4.2 Overall Benchmark Results

Aggregate ASR and average iteration statistics for the six LLM-pair combinations were shown in Figure 8. A more granular view across the ten system categories was provided by the heatmap in Figure 9. For completeness and reproducibility, the full numerical results (ASR and mean iterations per category) were reported in Table 4 in Appendix B.

Among the six pairwise combinations, those that included `deepseek-chat` or `gpt-4o` achieved the highest aggregate ASR, whereas the Google–Anthropic pairing (`gemini-pro` + `claude-sonnet-4`) recorded the lowest ASR across the benchmark. A plausible explanation is that the reasoning and code-generation strengths of `deepseek-chat` and `gpt-4o` complemented each other, improving the reliability of controller synthesis under multi-metric constraints.

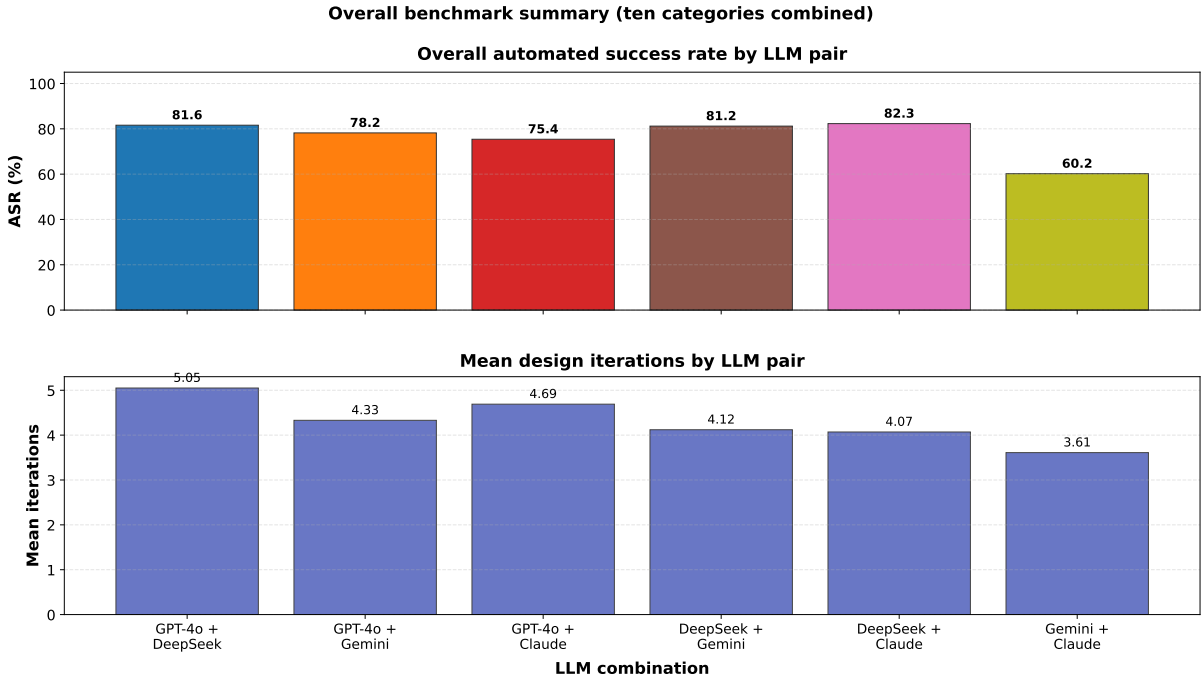


Fig. 8. Overall ASR and mean iterations by LLM pair.

Figure 9 provided a detailed view of ASR for all LLM pairs across different system types.

The heatmap revealed a clear gradient of difficulty across system categories. Nearly all pairs achieved an almost perfect ASR ($\geq$96%) on first-order stable-fast and first-order with-delay systems, indicating that these categories posed minimal difficulty for LLM-based controller synthesis. One notable exception was the OpenAI–Anthropic pairing (`gpt-4o` + `claude-sonnet-4`), which yielded only 68.0% ASR on first-order stable-fast systems, suggesting a pairing-specific

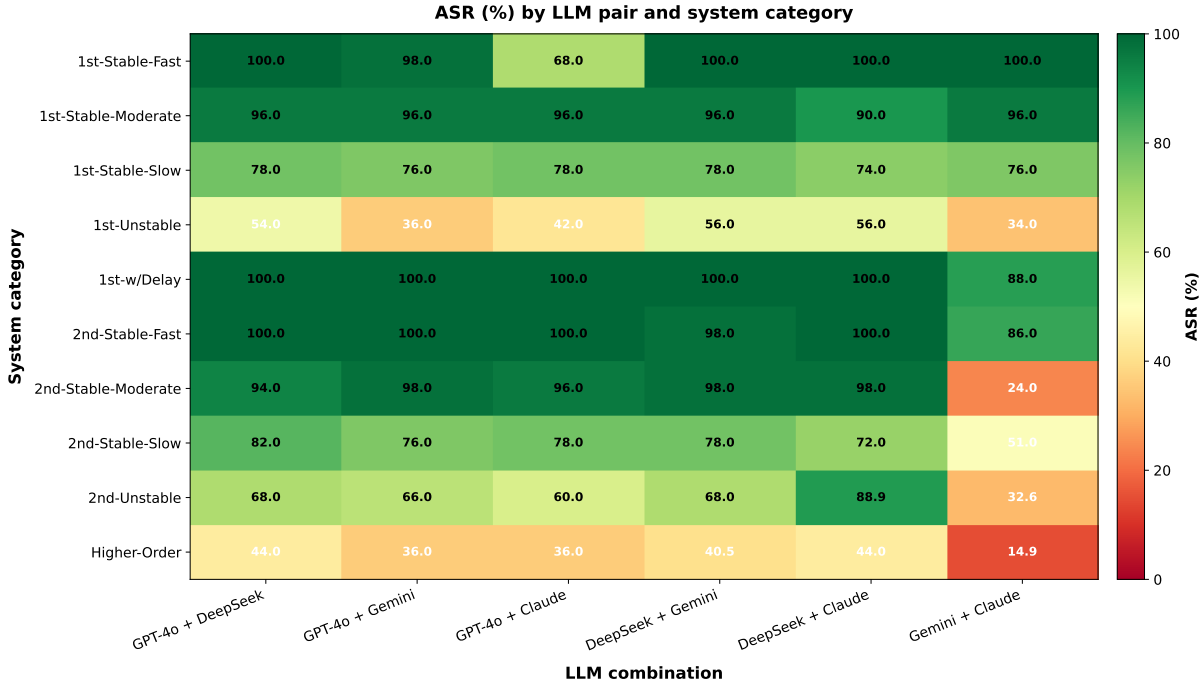


Fig. 9. ASR heatmap by LLM pair and system category.

weakness even on simple tasks.

Performance deteriorated with increasing system complexity. For stable-moderate systems, ASR values lay between 90% and 98% for most combinations (excluding `gemini-pro` + `claude-sonnet-4`). For stable-slow systems, ASR decreased to roughly 72–82%. Unstable and higher-order systems exhibited the widest spread of outcomes across model pairs.

The weakest aggregate behaviour appeared on higher-order systems for the Google–Anthropic pairing (`gemini-pro` + `claude-sonnet-4`), which achieved only 14.9% ASR in that category. The same pairing also recorded low ASR on second-order stable-moderate (24.0%) and second-order unstable (32.6%) systems, whereas most other combinations exceeded 60% on both categories. This pattern was consistent with `gemini-pro+claude-sonnet-4` being less reliable on tasks that required sustained stability reasoning under multi-pole interactions and open-loop instability.

By contrast, the DeepSeek–Anthropic pairing (`deepseek-chat` + `claude-sonnet-4`) achieved 88.9% ASR on second-order unstable systems—the best among all combinations for that category—and shared the top performance on higher-order systems (44.0%) with the OpenAI–DeepSeek pairing (`gpt-4o` + `deepseek-chat`). With the complete second-order results included, the DeepSeek–Google pairing (`deepseek-chat` + `gemini-pro`) was also confirmed to be competitive on second-order stable-moderate (98.0%) and second-order unstable (68.0%)

systems.

Key stratification effects. Beyond the overall rankings, the results exhibited three consistent stratifications. First, performance gaps widened on challenging instances (stable-slow, unstable, and higher-order plants), suggesting that robustness was dominated by stability reasoning and constraint handling rather than routine tuning. Second, the spread increased with system order and with open-loop instability, consistent with higher-dimensional dynamics and unstable poles enlarging the search space and tightening admissible controller regions. Third, ASR generally decreased as response slowed, with the drop most pronounced for `gemini-pro` + `claude-sonnet-4`, indicating that slow-response targets were the most sensitive to modelling and tuning errors.

4.3 Comparison against Single-Model Baseline

In contrast to the single-model baseline, RobustAgent attained a lower overall ASR. This reduction was not because RobustAgent was less capable of generating controllers, but rather because of the adoption of stringent acceptance criteria in RobustAgent.

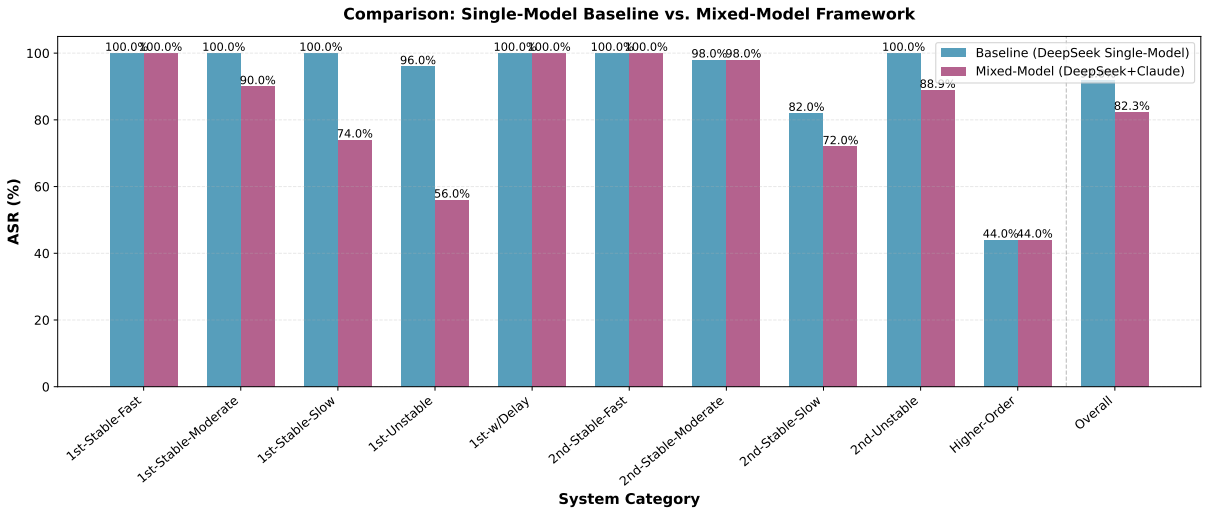


Fig. 10. Baseline (`deepseek-chat`) vs RobustAgent (`deepseek-chat` + `claude-sonnet-4`) ASR.

As shown in Figure 10, for easy cases such as stable-fast systems and systems with delay, the two approaches produced nearly identical results. This suggested that these problem types lay within the scope of present-day LLM-based controller design.

Discrepancies occurred primarily on more complex problem types, such as unstable systems or slow-response systems. In these cases, RobustAgent enforced multi-dimensional constraints: in

addition to LMI-based stability verification, it imposed hard limits on stability margin, phase margin, rise time, and ITAE. Consequently, many controllers that passed the single-model pole-based check could fail the LMI-based verification and/or violate at least one additional metric, resulting in a reduced ASR. Section 4.4 provides ablation studies to isolate the effect of each filter separately.

This trade-off between coverage and certification quality was intentional. The lower overall ASR reflected stricter acceptance criteria that prioritised deployment-oriented robustness over maximising raw pass rates.

Table 2. Baseline (`deepseek-chat`) vs RobustAgent (`deepseek-chat` + `claude-sonnet-4`) ASR by category.

System Category	Baseline ASR	RobustAgent ASR	Difference
1st-Order Stable-Fast	100.0%	100.0%	0.0 pp
1st-Order Stable-Moderate	100.0%	90.0%	-10.0 pp
1st-Order Stable-Slow	100.0%	74.0%	-26.0 pp
1st-Order Unstable	96.0%	56.0%	-40.0 pp
1st-Order w/Delay	100.0%	100.0%	0.0 pp
2nd-Order Stable-Fast	100.0%	100.0%	0.0 pp
2nd-Order Stable-Moderate	98.0%	98.0%	0.0 pp
2nd-Order Stable-Slow	82.0%	72.0%	-10.0 pp
2nd-Order Unstable	100.0%	88.9%	-11.1 pp
Higher-Order	44.0%	44.0%	0.0 pp
Overall	**92.0%**	**82.3%**	**-9.7 pp**

4.4 Ablation Study

Four variants where individual components of RobustAgent have been removed are compared to the full pipeline, on the same first-order stable subset as ControlAgent [5, Table 3]. The goal of this section is to identify which mechanisms are load-bearing for iterative refinement and slow-response tuning on this subset. Evidence for the positive contribution of LMI verification itself is therefore taken primarily from Section 4.3, where per-category gaps against the baseline are largest exactly in the regimes where pole-based checks are numerically unreliable.

w/o LMI verification. No LMI is performed, and stability is guaranteed via the closed-loop pole condition (i.e., the standard ControlAgent stability check). This column is included as a sanity check: on this first-order subset the pole test and the LMI verdict often coincide, so similar ASR confirms that LMI verification does not falsely reject easy designs.

w/o multi-API parallelism. Only one API is queried per iteration, so the design-memory stream is monolithic and the central agent lacks a second candidate controller to resort to. This variant tests the advantage of multi-API parallelism.

w/o robustness metrics (SM + ITAE). The acceptance criteria are limited to the original phase-margin and response-speed constraints, and ITAE and stability margin are excluded. This variant measures the price of including more metrics and should be contrasted to Section 4.3: the added constraints listed here are the ones responsible for the lower ASR compared to a single-model baseline.

w/o iterative refinement. The design process is collapsed into a single `parallel_design` call, with nothing written to the memory buffer. Each call results in a one-shot run. This variant measures the importance of iterative refinement on the same benchmark set.

	RobustAgent		**w/o LMI**		**w/o dual-LLM**		**w/o ITAE / SM**		**w/o feedback**	
	ASR	iter #	ASR	iter #	ASR	iter #	ASR	iter #	ASR	iter #
fast	**100.0**	**1.93**	100.0	1.67	93.3	3.13	100.0	1.33	66.7	1.00
moderate	**93.3**	**2.27**	93.3	2.13	93.3	2.20	100.0	1.40	60.0	1.00
slow	**80.0**	**3.87**	80.0	3.73	80.0	4.20	100.0	1.93	6.7	1.00

Table 3. Ablation ASR and mean iterations (first-order stable subset).

Several conclusions followed from Table 3. First, the *w/o LMI verification* column behaved as a sanity check rather than a primary ablation: on this first-order subset the closed-loop pole test and the LMI verdict often coincided, so similar ASR indicated that LMI verification did not introduce false rejections on easy designs—a hidden cost it would otherwise carry. Evidence of LMI's positive contribution is provided in Section 4.3, where the gap against a pole-based baseline is largest in precisely the categories where pole checks are least reliable.

Second, removing multi-API parallelism had negligible impact on the fast/moderate subsets, but caused a measurable difference on the slow subset, consistent with the claim in Section 3.1 that parallel APIs are most helpful when a model is likely to be inaccurate—notably on slow-response designs with the narrowest admissible ω_L interval.

Third, the performance decrease observed in the *w/o iterative refinement* variant was severe, reflecting the inability to meet slow-response targets via a one-shot template-based design. This quantitatively confirmed the ablation trend reported by ControlAgent [5, Table 3], where removing the iterative loop caused the largest decrease in performance. Iteration was therefore an important mechanism: it enabled additional robustness filters (extra metrics and parallel candidate selection) while retaining acceptable success rates.

Finally, the *w/o robustness metrics (SM + ITAE)* column showed that reverting to the original two-metric gate restored ASR toward 100% for all speed classes while reducing iteration count. This provided quantitative confirmation of the headline result in Section 4.3: the reduced ASR of the full RobustAgent framework was the price paid for additional constraints. Whether this trade-off is acceptable depends on the application; for safety-driven settings, a stricter acceptance bar is often preferable.

4.5 Integrated Discussion

The experimental findings led to several key conclusions:

1. **Mixed pairs that included `deepseek-chat` achieved the highest overall ASR in this benchmark (82.3% for `deepseek-chat` + `claude-sonnet-4`).** A plausible mechanism is that `deepseek-chat` benefited from training exposure to STEM- and code-heavy tasks that emphasise symbolic manipulation, algebraic consistency, and executable reasoning traces. Controller synthesis, even when presented in natural language, ultimately reduced to structured computation under stability and performance constraints, so these capabilities transferred directly to proposing reasonable controller families and parameter regimes. The same qualitative pattern—stronger scores on general mathematical and coding benchmarks correlating with more reliable multi-step reasoning—is widely reported in broad evaluations such as MMLU [28] and HumanEval-style coding suites [29], while surveys of LLM-agent assessment stress that such correlations do not automatically transfer to stability-constrained control search [22].

2. **Model complementarity mattered.** Pairings of heterogeneous providers could provide error correction when at least one model reliably caught stability- or constraint-related mistakes made by the other. In contrast, complementarity failed when both models shared the same weakness profile; the weaker `gemini-pro+claude-sonnet-4` pairing was consistent with both models being comparatively stronger at fluent explanation than at sustained symbolic stability reasoning, leading to overlapping failure modes rather than mutual compensation.

3. **Stable systems were handled reliably within the present benchmark.** This suggested that LLM-based controller synthesis on easy, well-conditioned plants was close to saturation: additional model scaling or prompt tuning yielded diminishing returns when success was already dominated by routine tuning patterns. The marginal research value therefore shifted toward challenging cases that genuinely stressed stability reasoning and

constraint handling. It also implied that future benchmarks should reduce the proportion of easy instances to avoid inflating apparent capability and to better discriminate model differences.

4. **Unstable and higher-order systems exposed current limitations of LLM-based reasoning.** Right-half-plane poles and higher-order dynamics required more than local parameter search: they demanded a global view of stability boundaries, coupled constraints, and non-intuitive trade-offs in admissible controller regions. Maintaining that global consistency across multiple refinement steps was difficult for current in-context reasoning, which could drift into locally plausible but globally inconsistent updates.

5. **Iteration count alone did not measure efficiency.** A low average iteration count could indicate rapid convergence (as observed with `deepseek-chat` + `gemini-pro` on easier problems) or premature failure (as seen with `gemini-pro` + `claude-sonnet-4` on more difficult tasks). Iteration counts must therefore be interpreted alongside ASR rather than as a standalone metric.

6. **LMI verification tightened the acceptance gate.** Because acceptance required successful Lyapunov/LMI certification together with hard performance constraints, the reported ASR reflected controllers that satisfied a deployment-oriented verification bar rather than heuristic pass/fail checks. Compared with recent LMI-based H_∞ synthesis approaches that integrate LLM agents [24], which document strong outcomes on COMPleib under the authors' reported protocol, RobustAgent enforced a stricter superset criterion: stability certification was necessary but not sufficient unless multi-dimensional robustness and transient constraints were also met. This aligned with recent recommendations for LLM-agent assessment [22], which emphasise combining success metrics with domain-specific verification standards.

4.6 Limitations

Despite the promising performance of RobustAgent, several limitations remained salient at the end of the study. First, higher-order systems were the most challenging category, with a maximum ASR of 44.0%. This suggested that current LLM-based reasoning was still limited when dealing with high-dimensional parameter spaces and more complex stability boundaries. Second, certain model combinations exhibited anomalous behaviour. In particular, the ASR of `gpt-4o` + `claude-sonnet-4` on first-order stable-fast systems was 68.0%, which was inconsistent with the excellent performance of most model combinations on simple tasks. This

anomaly may reflect sensitivity to prompt design, API-level fluctuations, or deeper incompatibility between the two models. Third, computational cost increased as the number of design iterations grew. Asynchronous parallelisation across API calls could be implemented to reduce wall-clock time, but efficiency remained an important practical issue. Finally, the benchmark tested only linear and linearised systems, which limited the generality of the conclusions.

5 Project Management

The project management spanned two semesters. The documents relied upon are listed in Appendix A: Gantt Chart (Figure 11), Risk Register (Figure 12), CPD Log (Section A.4), and Health and Safety Risk Assessment (Figures 13–14). These artefacts were reviewed at the four Progress Review Meetings (PRMs) and the weekly meetings, and were updated whenever the task scope or risk profile changed.

The plan split the work into four phases—planning and preparation, baseline reproduction, system expansion and evaluation, and writing and finalisation—with four PRMs as phase checkpoints, and weekly meetings to report progress to the supervisor. The first semester proceeded as planned. Two deviations from the original plan occurred during Semester 2. The LMI verification integration based on CVXPY took approximately one week longer than expected because the closed-loop transfer function was simplified to a numerically well-conditioned state-space representation, which exhibited strong numerical sensitivity when dealing with higher-order plants. This delay was absorbed by compressing the ablation experiment's time window rather than reducing its scope. Subsequently, the LLM experiments were split into smaller batches due to API rate limits and cost constraints. The final submission date remained unchanged, with writing of the results sections starting in parallel with the final experiment runs.

Risks were first identified during the project proposal stage and then reviewed at each PRM. They were re-scored when new situations arose. The issue of inconsistent LLM output was identified early and mitigated by pinning the model version and fixing the random seed when the API permitted it. Dependency conflicts among CVXPY, SCS, and LangGraph were exposed during baseline reproduction and resolved by using independent virtual environments and a locked `requirements.txt`. Excessively long LLM generation time in the higher-order system category was not foreseen early and was added to the risk register mid-project; an immediate cap on token budget was imposed, and the residual impact was reported in Section 4.4.

Self-directed learning was closely linked to the project's technical requirements. LangGraph was learnt before the orchestration phase began. The Lyapunov-based LMI formulation was

studied prior to replacing the pole-based stability check. Prompt engineering was refined iteratively as each new API was integrated. LaTeX was acquired progressively during the drafting stage. Each item was registered in the CPD log with the planned learning period, actual completion status, and a brief reflection on whether the skill was sufficient to support the corresponding task.

6 Conclusions and Future Work

6.1 Conclusions

This dissertation presented and evaluated RobustAgent, a mixed-agent, mixed-API framework for the design and verification of automated controllers. By integrating heterogeneous large language models with deterministic numerical evaluation, Lyapunov-based LMI feasibility tests, and hard limits on stability margin, phase margin, rise time, and ITAE, RobustAgent introduced a multi-metric, LMI-verified acceptance procedure for LLM-based controller design.

The experimental results showed that RobustAgent performed reliably on well-conditioned systems. They also revealed clear performance boundaries as system complexity increased. Configurations that included `deepseek-chat` achieved the highest overall ASR in the benchmark (82.3% for `deepseek-chat` + `claude-sonnet-4`). Backbone strength and provider complementarity were important for sustaining ASR under the multi-metric gate. Unstable systems and higher-order systems remained the main challenges, and current LLM-based controller synthesis still struggled with complex stability boundaries and high-dimensional search spaces.

Compared with the single-model baseline, RobustAgent achieved a lower overall ASR, but this decline reflected stricter acceptance criteria rather than a weakening of controller generation capability. By combining LMI-based stability verification with hard thresholds on stability margin (SM), phase margin (PM), rise time, and ITAE, RobustAgent filtered out controllers that passed basic pole-based tests but lacked sufficient robustness. Overall, the results indicated that RobustAgent supplied an extensible mixed-agent, mixed-API framework for LLM-based control engineering, prioritising robustness and formal verification over higher initial success rates.

This verification-oriented design also differentiated RobustAgent from recent certification-focused pipelines. For example, recent LMI-based H_∞ synthesis approaches that incorporate LLM agents [24] reported strong benchmark coverage on COMPleib under the evaluation protocol described therein; RobustAgent nonetheless adopted an acceptance criterion that was a

stricter superset, combining Lyapunov/LMI stability certification with hard multi-dimensional performance constraints. Consequently, an overall ASR of 82.3% reflected controllers that cleared a higher certification bar rather than merely achieving feasibility under a single criterion. This emphasis aligned with recent recommendations on LLM-agent assessment [22], which argue that raw success rates should be interpreted together with domain-specific verification standards and deployment-oriented robustness requirements.

Beyond aggregate ASR, the dissertation further illustrated how mixed-agent, mixed-API LLM workflows could reduce the manual effort required for documenting and iterating structured controller designs when numerical checks supplied the acceptance logic. By integrating automated reasoning, structured evaluation, and LMI-based stability tests, RobustAgent enabled users with limited control engineering experience to generate and assess candidate controllers against explicit thresholds. This kept classical robustness criteria in the loop while automating parts of the search and reporting workflow.

6.2 Future Work

Future work will address the above limitations in several directions. Hardware-in-the-loop verification will be introduced to evaluate controller performance on physical systems and embedded platforms, thereby narrowing the gap between simulation and real-world deployment. Dedicated sub-agents for higher-order systems and improved prompt engineering will be developed to improve performance in the most challenging categories. A more detailed investigation of the anomalous `gpt-4o` + `claude-sonnet-4` results is required to determine whether the underlying cause lies in prompt design, API behaviour, or model-level collaboration issues. Reinforcement-learning-based optimisation of agent reasoning may improve the consistency of controller generation and reduce the iteration count on difficult benchmarks. The ControlEval benchmark will be extended to cover a broader range of linear and nonlinear systems, including plants with stronger non-minimum-phase characteristics and longer delays. Advanced techniques such as nonlinear Lyapunov verification, sum-of-squares methods, real-time optimisation, and model predictive control will be integrated so that RobustAgent can treat more complex and practically relevant control problems.

References

[1] OpenAI, "GPT-4 technical report," *arXiv preprint*, 2023. arXiv: 2303.08774. [Online]. Available: https://arxiv.org/abs/2303.08774.

[2] Anthropic. "The Claude 3 model family: Opus, Sonnet, Haiku," Accessed: Mar. 18, 2026. [Online]. Available: https://www.anthropic.com/news/claude-3-family.

[3] DeepSeek-AI, "DeepSeek-V3 technical report," *arXiv preprint*, 2025. arXiv: 2412.19437. [Online]. Available: https://arxiv.org/abs/2412.19437.

[4] Gemini Team, Google, "Gemini: A family of highly capable multimodal models," *arXiv preprint*, 2024. arXiv: 2312.11805. [Online]. Available: https://arxiv.org/abs/2312.11805.

[5] X. Guo et al., "ControlAgent: Automating control system design via novel integration of LLM agents and domain expertise," *arXiv preprint*, 2024. DOI: 10.48550/arXiv.2410.19811. arXiv: 2410.19811. [Online]. Available: https://arxiv.org/abs/2410.19811.

[6] IEEE Control Systems Society. "Automating automation: Large language models for control design." IEEE Xplore document 11363119, Accessed: Apr. 30, 2026. [Online]. Available: https://ieeexplore.ieee.org/document/11363119.

[7] S. Skogestad and I. Postlethwaite, *Multivariable Feedback Control: Analysis and Design*, 2nd ed. Chichester: John Wiley & Sons, 2005.

[8] K. J. Åström and T. Hägglund, *Advanced PID Control*. Research Triangle Park, NC: ISA—The Instrumentation, Systems, and Automation Society, 2006.

[9] K. J. Åström and T. Hägglund, "The future of PID control," *Control Engineering Practice*, vol. 9, no. 11, pp. 1163–1175, 2001.

[10] J. C. Doyle, B. A. Francis, and A. R. Tannenbaum, *Feedback Control Theory*. New York: Macmillan, 1992.

[11] K. J. Åström and R. M. Murray, *Feedback Systems: An Introduction for Scientists and Engineers*, 2nd ed. Princeton, NJ: Princeton University Press, 2021.

[12] K. Ogata, *Modern Control Engineering*, 5th ed. Upper Saddle River, NJ: Prentice Hall, 2010.

[13] G. F. Franklin, J. D. Powell, and A. Emami-Naeini, *Feedback Control of Dynamic Systems*, 7th ed. Boston: Pearson, 2014.

[14] J. G. Ziegler and N. B. Nichols, "Optimum settings for automatic controllers," *Transactions of the ASME*, vol. 64, no. 8, pp. 759–768, 1942.

[15] S. Skogestad, "Simple analytic rules for model reduction and PID controller tuning," *Journal of Process Control*, vol. 13, no. 4, pp. 291–309, 2003.

[16] K. J. Åström and T. Hägglund, *PID Controllers: Theory, Design, and Tuning*, 2nd ed. Research Triangle Park, NC: Instrument Society of America, 1995.

[17] K. Zhou, J. C. Doyle, and K. Glover, *Robust and Optimal Control*. Upper Saddle River, NJ: Prentice Hall, 1996.

[18] D. Kevian et al., “Capabilities of large language models in control engineering: A benchmark study on GPT-4, Claude 3 Opus, and Gemini 1.0 Ultra,” *arXiv preprint*, 2024. arXiv: 2404.03647. [Online]. Available: https://arxiv.org/abs/2404.03647.

[19] Z. Ji et al., “Survey of hallucination in natural language generation,” *ACM Computing Surveys*, vol. 55, no. 12, pp. 1–38, 2023.

[20] Q. Wu et al., “AutoGen: Enabling next-gen LLM applications via multi-agent conversation,” *arXiv preprint*, 2023. arXiv: 2308.08155. [Online]. Available: https://arxiv.org/abs/2308.08155.

[21] LangChain. “LangGraph,” Accessed: Mar. 18, 2026. [Online]. Available: https://blog.langchain.com/langgraph/.

[22] A. Yehudai et al. “Survey on evaluation of LLM-based agents.” ACL Findings. arXiv: 2503.16416, Accessed: Apr. 26, 2026. [Online]. Available: https://doi.org/10.48550/arXiv.2503.16416.

[23] R. Zahedifar et al., “LLM-Agent-Controller: A universal multi-agent large language model system as a control engineer,” *arXiv preprint*, 2025. arXiv: 2505.19567. [Online]. Available: https://arxiv.org/abs/2505.19567.

[24] S. Li et al., “From natural language to certified H-infinity controllers: Integrating LLM agents with LMI-based synthesis,” *arXiv preprint*, 2025, arXiv:2511.07894v1, eess.SY. arXiv: 2511.07894. [Online]. Available: https://arxiv.org/abs/2511.07894.

[25] S. Boyd et al., *Linear Matrix Inequalities in System and Control Theory*. Philadelphia, PA: Society for Industrial and Applied Mathematics, 1994.

[26] S. Diamond and S. Boyd, “CVXPY: A python-embedded modeling language for convex optimization,” *Journal of Machine Learning Research*, vol. 17, no. 83, pp. 1–5, 2016.

[27] B. O’Donoghue et al., “Conic optimization via operator splitting and homogeneous self-dual embedding,” *Journal of Optimization Theory and Applications*, vol. 169, no. 3, pp. 1042–1068, 2016.

[28] D. Hendrycks et al. “Measuring massive multitask language understanding,” Accessed: May 1, 2026. [Online]. Available: https://arxiv.org/abs/2009.03300.

[29] M. Chen et al. “Evaluating large language models trained on code,” Accessed: May 1, 2026. [Online]. Available: https://arxiv.org/abs/2107.03374.

Appendices

A Project Management Artefacts

The following project management artefacts are included in this appendix, in line with the project specification:

- Preliminary Project Proposal;
- Project Plan and Gantt Chart (final, updated version);
- Risk Register;
- Continuing Professional Development (CPD) Log;
- Health & Safety Risk Assessment.
- Link to the version-controlled repository containing software developed as part of the project.

A.1 Preliminary Project Proposal

LLM-Assisted Control Design with Domain Expertise and AI-Based Controller Evaluation

1 Background and motivation

Control system design forms the foundation of modern engineering, applied across diverse engineering domains. Traditional controller design heavily relies on human expertise and iterative parameter tuning, which is time-consuming and demands strong theoretical knowledge and practical experience. Even with mature methods like PID control, engineers must repeatedly adjust parameters to balance performance and robustness, resulting in inefficient processes that are difficult to scale. The emergence of large language models (LLMs) has revolutionized reasoning, code generation, and task decomposition, sparking research into their application in engineering automation.

Recent studies (such as ControlAgent) have demonstrated that LLMs can automate controller design through iterative reasoning and domain-guided feedback. However, existing frameworks, including ControlAgent, remain focused on single-controller design, lacking mechanisms to compare or coordinate different AI-generated solutions. In practical control engineering, multiple controller configurations and parameter settings typically require testing and evaluation to select the optimal control strategy for a specific system.

This gap highlights the need for a more comprehensive and systematic framework that not only reproduces human-like controller design but also enables objective performance evaluation, cross-AI comparison, and standardized benchmarking within the LLM-driven control domain.

2 Aim

This project aims to develop an enhanced large language model (LLM) control design framework to extend the capabilities of ControlAgent—a system that achieves automatic controller synthesis through iterative reasoning, domain-specific guidance, and numerical feedback. The original ControlAgent demonstrated how large language models can emulate the work of human control engineers by integrating control theory knowledge with computational verification using tools such as Python control libraries.

Building upon this foundation, the new framework will validate core functionalities using the ControlEval dataset while introducing explicitly defined performance metrics grounded in time-domain and frequency-domain expertise. This establishes an objective, reproducible evaluation system. Furthermore, the extended architecture will incorporate additional AI interfaces and evaluation agents to compare and assess design proposals from diverse artificial intelligence models. The complete system will undergo benchmarking against MathWorks' PID Tuner to establish quantitative reference standards for performance and robustness.

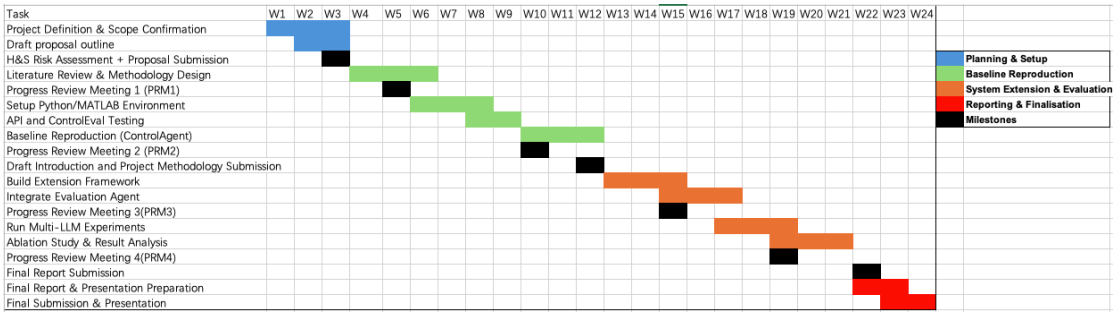
3 Project Objectives

1. Develop and validate the baseline ControlAgent framework using the ControlEval dataset to verify its functionality and establish a reliable foundation for subsequent framework extension.

2. Define clear and standardized performance metrics based on time- and frequency-domain principles, including phase margin, settling time, steady-state error, and overshoot, to ensure objective and reproducible evaluation.
3. Extend the framework by integrating an additional AI interface and an Evaluation Agent, enabling systematic comparison between AI-generated controller designs. The enhanced system will be benchmarked against MathWorks PID Tuner to provide a quantitative reference for performance and robustness.

4 Project Plan

The Gantt chart below outlines the project's key phases, milestones, and overlapping tasks across two semesters, while the risk register summarises major risks and mitigation measures to ensure smooth and reliable project execution.



MANCHESTER 1824
The University of Manchester

RISK REGISTER FOR 3rd YEAR PROJECT

Project Title:	LLM-Assisted Control Design with Domain Expertise and AI-Based Controller Evaluation		Submission Date:	17/Oct/2025	
Student Name:	Jiarui Wang				

Project Risk	Severity			Potential			Score (Severity x Potential)	Mitigation Measures
	L	M	H	L	M	H	L=1, M=2, H=3	
LLM output inconsistency		√			√		4	Fix random seeds and model version
code loss	√				√		2	Maintain backups on OneDrive and GitHub
Evaluation logic errors		√			√		4	Create mock test cases and verify outputs manually with supervisor.
code integration conflicts		√		√			2	merge frequently with short commits.
Overly long generation time from LLM	√					√	3	Reduce max tokens
Model response timeout or crash	√				√		2	use smaller test batches first.
LLM hallucination		√			√		4	Cross-check LLM output with textbook examples or MATLAB results.
Overlapping report writing and coding	√			√			1	Start writing methodology section early while testing.
Poorly formatted report	√			√			1	Use LaTeX / Word citation tools and pre-submit to Grammarly.
Environment dependency conflicts		√			√		4	Lock dependencies with requirements.lock and document library versions.

Reference

[1] X. Guo, D. Keivan, U. Syed, L. Qin, H. Zhang, G. Dullerud, P. Seiler, and B. Hu, "ControlAgent: Automating Control System Design via Novel Integration of LLM Agents and Domain Expertise," arXiv preprint arXiv:2410.19811, 2024.

A.2 Project Plan and Gantt Chart

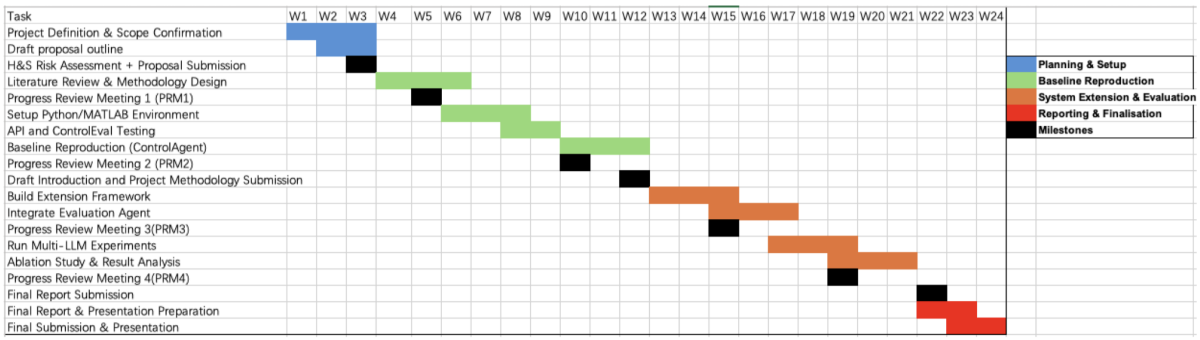


Fig. 11. Project Gantt chart.

A.3 Risk Register

MANCHESTER 1824
The University of Manchester

RISK REGISTER FOR 3rd YEAR PROJECT

Project Title:	LLM-Assisted Control Design with Domain Expertise and AI-Based Controller Evaluation		Submission Date:	17/Oct/2025	
Student Name:	Jiarui Wang				

Project Risk	Severity			Potential			Score (Severity x Potential)	Mitigation Measures
	L	M	H	L	M	H	L=1, M=2, H=3	
LLM output inconsistency		√			√		4	Fix random seeds and model version
code loss	√				√		2	Maintain backups on OneDrive and GitHub
Evaluation logic errors		√			√		4	Create mock test cases and verify outputs manually with supervisor.
code integration conflicts		√		√			2	merge frequently with short commits.
Overly long generation time from LLM	√					√	3	Reduce max tokens
Model response timeout or crash	√				√		2	use smaller test batches first.
LLM hallucination		√			√		4	Cross-check LLM output with textbook examples or MATLAB results.
Overlapping report writing and coding	√			√			1	Start writing methodology section early while testing.
Poorly formatted report	√			√			1	Use LaTeX / Word citation tools and pre-submit to Grammarly.
Environment dependency conflicts		√			√		4	Lock dependencies with requirements.lock and document library versions.

Fig. 12. Project risk register.

A.4 Continuing Professional Development (CPD) Log

Continuing Professional Development Log

Name: *Jiarui Wang*

Current and recent CPD activity:

CPD Activity Title	Description	CPD Dates	CPD Hours
Thesis research — Control system performance analysis	Conducted in-depth analysis of closed-loop control system performance using classical frequency-domain metrics (Stability Margin, Phase Margin) and time-domain criteria (Rise Time, ITAE). Prepared an articulation of each metric and justified the use of Stability Margin over the more canonical GM/PM framework in the thesis context.	Sep 2025 – Apr 2026	200
Literature review — PID tuning & ITAE-optimal design	Systematic review of modern tuning approaches (Ziegler–Nichols, IMC, optimisation-based ITAE minimisation). Findings integrated into the controller design chapter.	Nov 2025 – Jan 2026	30
MATLAB/Simulink advanced control toolbox	Self-study on frequency-domain analysis, Bode/Nyquist plots, robust stability checks. Applied to thesis case studies.	Jan – Feb 2026	25

Planned and Future CPD activity:

CPD Activity Title	Description	Skills addressed	Dates
Thesis defense (viva voce)	Oral defense of thesis. Will articulate technical choices and respond to examiner challenges, including the SM vs GM/PM framing.	Technical communication under pressure; critical defense of engineering decisions	May – Jun 2026
Self-study — Model Predictive Control (MPC)	Extend from classical frequency-domain control to optimisation-based MPC: constraint handling, receding-horizon design, industrial applications.	Fills gap in modern control; broadens beyond classical methods	Jul – Aug 2026
Tooling — Python `python-control` library	Transition from MATLAB-centric workflow to open-source Python control tooling.	Software versatility; reduces reliance on proprietary toolchain	Jul – Sep 2026
IET Chartered Engineer (CEng) — portfolio prep	Begin compiling competence evidence for CEng application.	Professional accreditation; structured reflection	Ongoing, 2026–2027
Control engineering internship (pending application)	Planned industrial internship applying frequency-domain analysis and controller design from thesis research to real-world systems. Expected exposure to industrial control platforms, system identification, and deployment practices.	Bridges academic theory and industrial practice; fills engineering implementation gap; industry toolchain exposure	Planned, subject to outcome

A.5 Health & Safety Risk Assessment

MANCHESTER 1824
The University of Manchester

General Risk Assessment Form

Date: (1)	Assessed by: (2)	Checked by: (3)	Location: (4)	Assessment ref no (5)	Review date: (6)
17/10/2025	Jiarui Wang	Dr.Chao Chen	163, Assembly	FYP-2025- LLM-Assisted Control Design with Domain Expertise and AI-Based Controller Evaluation	17/6/2026

Task / premises: (7)
Development and testing of AI-based control system design framework ("ControlAgent") using University-provided computing resources. The project involves programming, data analysis, simulation of control systems, and limited hardware interfacing.

Activity (8)	Hazard (9)	Who might be harmed and how (10)	Existing measures in place to control the risk (11)	Risk rating (12)	Result (13)
Working on computer for long hours	Eye strain, posture-related injury	Student	Regular breaks (20-20-20 rule), adjustable chair and desk, proper posture, monitor at eye level	Low	A
Use of electrical equipment (computer, lab hardware)	Electric shock, overheating	Student	PAT-tested equipment only; avoid liquids near devices; power off after use	Low	A
Data processing and programming	Data loss due to software crash	Student	Regular backups on OneDrive; version control via GitHub; autosave enabled	Low	A
Use of online AI models (LLMs)	Potential data security/privacy breach	Student	Use anonymised and non-confidential data only; comply with University Data Protection Policy	Low	A
Collaboration and file sharing	Version conflicts or accidental data deletion	Student	Access control (read/write restrictions), shared repository with revision history	Low	A

University Safety Services risk assessment form and guidance notes v1.1
Revised July 2023

Fig. 13. Health and safety risk assessment (page 1).

Action plan (14)

Ref No	Further action required	Action by whom	Action by when	Done
1	Ensure all datasets used are anonymised and comply with GDPR	Jiarui Wang	Week3	-
2	Conduct annual review of PAT testing for used equipment	EEE Lab Technician	Week6	-
3	Enable automatic backup of all simulation results	Jiarui Wang	Week 2	-

Fig. 14. Health and safety risk assessment (page 2).

A.6 Version-Controlled Repository Link

The RobustAgent source code is publicly available at `https://github.com/JiaruiWang-79/Robustagent`. Setup, dependencies, and usage are documented in `README.md` at the repository root.

B Full Benchmark Tables

This appendix lists the full numerical benchmark results (ASR and mean iteration count) referenced in Section 4.3.

Table 4. Full benchmark results (ASR and mean iterations).

	gpt-4o deepseek-chat		gpt-4o gemini-pro		gpt-4o claude-sonnet-4		deepseek-chat gemini-pro		deepseek-chat claude-sonnet-4		gemini-pro claude-sonnet-4	
	ASR	**Iter**	**ASR**	**Iter**	**ASR**	**Iter**	**ASR**	**Iter**	**ASR**	**Iter**	**ASR**	**Iter**
1st-Stable-Fast	**100.0**	2.92	98.0	2.74	68.0	4.42	**100.0**	1.70	**100.0**	1.68	**100.0**	1.70
1st-Stable-Moderate	**96.0**	2.62	**96.0**	2.18	**96.0**	2.28	**96.0**	1.80	90.0	2.58	**96.0**	1.84
1st-Stable-Slow	**78.0**	4.84	76.0	4.48	**78.0**	4.50	**78.0**	4.12	74.0	4.68	76.0	4.72
1st-Unstable	54.0	7.78	36.0	7.92	42.0	7.70	**56.0**	7.26	**56.0**	7.28	34.0	5.55
1st-w/Delay	**100.0**	2.32	**100.0**	2.18	**100.0**	2.14	**100.0**	2.02	**100.0**	2.04	88.0	2.94
2nd-Stable-Fast	**100.0**	2.54	**100.0**	2.06	**100.0**	2.16	98.0	1.96	**100.0**	1.88	86.0	2.92
2nd-Stable-Moderate	94.0	5.76	**98.0**	3.44	96.0	4.14	**98.0**	4.64	**98.0**	4.22	24.0	2.22
2nd-Stable-Slow	**82.0**	5.90	76.0	4.78	78.0	4.54	78.0	4.76	72.0	4.92	51.0	6.39
2nd-Unstable	68.0	7.42	66.0	7.40	60.0	7.24	68.0	5.36	**88.9**	4.11	32.6	5.35
Higher-Order	**44.0**	8.38	36.0	7.70	36.0	7.80	40.5	7.62	**44.0**	7.28	14.9	2.45
Overall	81.6	5.05	78.2	4.33	75.4	4.69	81.2	4.12	**82.3**	4.07	60.2	3.61

C Additional RobustAgent Artefacts

C.1 Demonstration: Iterative Design Process

A demo of the design process

This example walks through the RobustAgent pipeline implemented in `langgraph_agent.py`: a **Central Agent** (LLM) routes the task to one of six task-specific subagents, each pair of design LLM calls is evaluated by the **Python agent** using `util.loop_shaping_new_metrics` (stability margin, phase margin, rise time, ITAE), and `design_memory` carries feedback into the next prompt until requirements are met or the iteration cap is hit.

Table 1: Overview of the LLM agents and their roles in RobustAgent (from `instruction.central_agent_prompt`).

Agent	Role
Central Agent	High-level task analysis and subagent selection (JSON: `Agent Number`, `Task Requirement`).
Subagent 1	Controller design for **first-order stable** plant (PI loop shaping).
Subagent 2	First-order **unstable** plant.
Subagent 3	Second-order **stable** plant (PID).
Subagent 4	Second-order **unstable** plant.
Subagent 5	First-order plant **with time delay**.
Subagent 6	**Higher-order** plant.

The plant and thresholds below are a representative row from `ControlEval_new_metrics/first_order_stable_m` (id 0). Rise-time limits encode the required **response speed** class (here, *moderate*).

User input

Please design a controller for the following system:

$$G(s) = \frac{19.95}{s + 0.390}\,.$$

- Closed-loop stable; stability margin SM ≥ 0.3.
- Phase margin $\phi_m \geq 71.54°$.
- Rise time T_r in [0.84, 5.17] s (synchronous with the dataset field `rise_time_min`/`rise_time_max`).
- Integral time-weighted absolute error ITAE ≤ 0.50. [a]
- **Response speed** (from the T_r band): *moderate*.

[a] The JSON file also lists a much tighter `ITAE_max` ≈ 0.03; for this didactic three-step PI example we use 0.50 so a single loop-shaping design can meet every bound while mirroring the same evaluation code path (`compute_ITAE` in `util.py`).

The central model parses the request and returns structured JSON. Excerpt (paraphrased below as natural language).

Central agent output

Task requirement. Stabilize the first-order plant $G(s) = 19.95/(s + 0.39)$ and meet the new-metrics bundle: SM, ϕ_m, T_r window, and ITAE cap; response speed is classified as **moderate** from the allowed T_r range.
Task analysis. The pole is at $s = -0.39$; the path is a **first-order stable** loop-shaping job. Route to **Agent 1** (PI template in `subagents/first_ord_stable.py`).
Assigned subagent. **Agent Number 1.**

Subagent 1 issues JSON with loop bandwidth ω_L and β_b; the Python layer forms the PI controller in

`util.check_stability` / `loop_shaping_new_metrics`. Below, $\beta_b = \sqrt{10} \approx 3.162$ (typical default in the instruction templates).

Task-specific agent output (iteration 1)

Iteration 1. Pick an initial loop bandwidth $\omega_L = 0.55$ rad/s. At the desired crossover, $|G(\mathrm{j}\omega_L)|$ sets K_p; the integral boost follows the standard PI structure in `util.loop_shaping`.

Python evaluation. For this design, $\phi_m \approx 105°$ and SM is well above 0.3, but the measured rise time is $T_r \approx 9.7$ s and ITAE ≈ 8.6, so the response is too sluggish for the T_r upper limit (5.17 s) and the ITAE cap—**increase** ω_L for the next attempt.

Memory buffers (`DesignMemory`) log parameters and metrics; the next prompt adds `feedback_prompt` text derived from the failed design.

Task-specific agent output (iteration 2)

Iteration 2. Raise ω_L to 1.0 rad/s. The step response quickens: $T_r \approx 2.6$ s now lies inside [0.84, 5.17], and ϕ_m remains above 71.54°.

However, ITAE ≈ 1.85 is still above the 0.50 budget. The agent keeps the same structure and **pushes** ω_L **further** to trade tracking error for speed while watching ϕ_m.

For iteration 2, the direction matches the project rule: if at least one API has not met all checks, the graph returns to `parallel_design` (`langgraph_agent._should_continue`).

Task-specific agent output (iteration 3)

Iteration 3. Set $\omega_L = 2.15$ rad/s. At this point $T_r \approx 0.85$ s (inside the required band), $\phi_m \approx 82°$, ITAE $\approx 0.47 < 0.50$, and SM ≥ 0.3 from the stability-margins call. Because ϕ_m is comfortably above the floor, a secondary tweak could reduce β_b for robustness; here the design already clears `_check_requirements`.

The task-specific design is **verified by the Python agent**; `select_best` / `compare_designs` would then package the winning parameter set when both LLM paths are run in the full graph.

Summary. This walkthrough mirrors the boxes in a typical paper figure: user specification with four design metrics plus an explicit **response speed** line, central routing to Agent 1, and two refinement steps driven by **rise time** and **ITAE** feedback from the same closed-loop code used in production.