INFORMATICS INSTITUTE OF TECHNOLOGY
In Collaboration with
UNIVERSITY OF WESTMINSTER

AURA: Real-Time Environment Adaptation Using Neuro-Fuzzy Procedural Content Generation with Behavioural Clustering

Final Year Project Dissertation by
Mr. K.W. J. P. Geevinda
W1871471 | 20212016

Supervised by
Mr. Banu Athuraliya

Submitted in partial fulfilment of the requirements for the BEng in Software Engineering degree at the University of Westminster.
March 2026

ABSTRACT

Problem: Procedural content generation (PCG) in modern games is typically static or reactively unstable, failing to adapt to player behaviour across combat, collection, and exploration, resulting in reduced engagement, cold-start bias, and repetitive gameplay.

Methodology: AURA is a calibration-first, telemetry-driven framework for real-time adaptation. Phase 1 established a neutral baseline (7 participants, 3 modes, z-score NeutralityScore). Phase 2 modelled behaviour using K-Means soft memberships and a canonical ANFIS-inspired adaptation surface, approximated by a 6→16→8→1 MLP surrogate. Phase 3 deployed a decoupled TypeScript backend integrated with Unreal Engine, applying bounded PCG updates every 30 seconds.

Results: The surrogate achieved Test $R^2 = 0.9264$ (95% CI [0.874, 0.954]) and MAE = 0.0127. Clustering (3,240 windows) yielded a silhouette score of 0.3752. Sensitivity analysis identified soft_combat as dominant (0.1449). A delta ablation showed removing temporal features reduced R^2 from 0.9726 to 0.5891 (−38.35 pp), confirming soft membership alone is insufficient. Runtime latency was <200 ms, with 10/12 requirements validated and 62 automated tests.

Subject Descriptors:

- Computing methodologies → Artificial intelligence → Knowledge representation and reasoning → Logic programming and answer set programming
- Computing methodologies → Machine learning → Learning paradigms → Supervised learning → Supervised learning by classification
- Human-centered computing → Human-computer interaction (HCI) → HCI design and evaluation methods → User models

Keywords: adaptive procedural content generation, behavioural clustering, neuro-fuzzy surrogate inference, telemetry-driven adaptation, real-time game AI

DECLARATION

I hereby declare that this dissertation is the result of my own independent work and has not been previously submitted, in whole or in part, for any other academic qualification. All sources of information, ideas, and contributions from prior research have been appropriately acknowledged and referenced in accordance with academic standards.

I further confirm that this study has been conducted with integrity, and that no part of this work involves plagiarism or unauthorized use of material.

Student Name: K. W. J. P. Pasindu Geevinda
Student ID: w1871471
Programme: BEng (Hons) Software Engineering
Date: 17 April 2026

Signature : 

RESEARCH OUTPUTS

Game Telemetry and Player Archetype Modelling (AURA)

Platform: Kaggle Datasets

URL: https://www.kaggle.com/datasets/pasindugeevinda/game-telemetry-and-player-archetype-modeling-aura

Status: Published and publicly accessible

Description: Phase 2 gameplay telemetry dataset comprising 3,240 telemetry windows from 45 participant sessions, including normalised features, IDW soft membership values, temporal deltas, and canonical ANFIS-inspired target multiplier labels. Supports full reproducibility of the AURA behavioural clustering and MLP surrogate training pipeline.

Full AURA Pipeline

Platform: Kaggle Code (Notebook)

URL: https://www.kaggle.com/code/pasindugeevinda/full-aura-pipeline

Status: Published and publicly accessible

Description: End-to-end executable notebook implementing the complete 8-stage AURA pipeline: data loading, MinMax normalisation, activity scoring, K-Means clustering (K=3), IDW soft membership computation, temporal delta calculation, ANFIS-inspired target generation, and MLP surrogate training (6→16→8→1). Reproduces Test $R^2 = 0.9264$ reported in Chapter 8.

ACKNOWLEDGEMENTS

This research represents the culmination of sustained academic effort and extensive technical investigation, a journey made possible through the support and guidance of many individuals.

Foremost, I extend my sincere gratitude to my supervisor, Mr. Banu Athuraliya, whose mentorship was fundamental throughout this project. His valuable insights, thoughtful guidance, and continuous encouragement enabled me to approach the technical and research challenges of this study with clarity and confidence. His confidence in the direction of this work significantly contributed to its successful completion.

I also wish to acknowledge the lecturers and academic staff of the Informatics Institute of Technology and the University of Westminster for their commitment to developing competent and industry-ready graduates. The knowledge and skills gained during this degree programme formed the essential foundation upon which this research was developed. I further appreciate the support provided by the institutional administration at various stages of this academic journey.

My appreciation is extended to the domain experts, technical evaluators, and survey participants who generously contributed their time and expertise to the evaluation process, as well as to all participants involved in the calibration study and focus group sessions. Their feedback and perspectives were vital in strengthening the validity and credibility of the findings presented in this dissertation.

Finally, I express my deepest gratitude to my parents and family for their unwavering support, sacrifices, and constant encouragement throughout this journey. I am also thankful to my friends and peers for their continuous motivation and support, which played an important role in reaching this milestone.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

Abbreviation	Definition
AI	Artificial Intelligence
ANFIS	Adaptive Neuro-Fuzzy Inference System
APCG	Adaptive Procedural Content Generation
API	Application Programming Interface
AURA	Adaptive User-Responsive Architecture
CI	Continuous Integration
DDA	Dynamic Difficulty Adjustment
EDPCG	Experience-Driven PCG
FIS	Fuzzy Inference System
FR	Functional Requirement
HCI	Human-Computer Interaction
IDE	Integrated Development Environment
IDW	Inverse Distance Weighting
JSON	JavaScript Object Notation
LSTM	Long Short-Term Memory
ML	Machine Learning
MLP	Multi-Layer Perceptron
NeSy	Neuro-Symbolic
NSIL	Neuro-Symbolic Inference Layer
NFR	Non-Functional Requirement
PAD	Pleasure, Arousal, Dominance
PCG	Procedural Content Generation
PPCG	Progressive PCG
RQ	Research Question
SBPCG	Search-Based PCG
SRS	Software Requirements Specification
UE	Unreal Engine

CHAPTER 01: INTRODUCTION

1.1. Chapter overview

This chapter establishes the foundation for AURA (Adaptive User-Responsive Architecture), a calibration-first, telemetry-driven adaptive procedural content generation framework. It outlines the limitations of static and reactive systems, defines the research problem and proposed solution, and critically reviews existing work. The chapter further identifies research gaps, presents contributions to both the problem and research domains, and concludes with the research challenges, questions, aims, and objectives.

1.2. Problem background

1.2.1. Limitations of static procedural content and fixed difficulty curves

Procedural Content Generation (PCG) has traditionally relied on rule-based and stochastic methods to automate environment creation. While effective for scalability, these approaches often produce generic experiences that fail to account for differences in player skill, learning rate, and behavioural preference (Yu, 2024; Alyaseri, 2025). This one-size-fits-all design limits long-term engagement. Fixed difficulty curves further exacerbate this issue, leading to the difficulty paradox, where skilled players experience boredom while less experienced players face frustration and disengagement (Wheat et al., 2016; Kyiv, Ukraine and Sulyma, 2025).

1.2.2. Instability and limitations in reactive Dynamic Difficulty Adjustment systems

Dynamic Difficulty Adjustment (DDA) was introduced to address static limitations by adapting challenge based on player performance. However, rule-based and threshold-driven approaches lack expressiveness and scalability, requiring extensive manual rule definition that often fails to capture diverse behavioural patterns (Xue et al., 2017). Simple reactive strategies, such as increasing difficulty after success, overlook nuanced differences in player performance. More critically, reactive adaptation can introduce instability, where continuous over-correction between player and system results in oscillatory gameplay that disrupts immersion (Hunicke and Chapman, 2004; Mortazavi, 2024).

1.2.3. Cold start biases and behavioural modelling challenges

Adaptive systems face a fundamental cold-start problem, where early-session behaviour lacks sufficient data for reliable personalisation (Bawa and Aponso, 2025). Initial player actions often reflect exploration rather than stable behavioural patterns, leading to inaccurate modelling if adaptation is applied prematurely (Xue et al., 2017). Research recommends an initial observation phase in which telemetry is collected without adaptation, allowing a neutral behavioural baseline to be established before personalisation begins (Mi and Gao, 2025). Without this separation, early bias propagates through subsequent adaptation.

1.2.4. Interpretability, player agency, and engagement equilibrium

Behavioural modelling introduces challenges related to interpretability and player agency. Hard clustering enforces rigid player categories, despite real behaviour being fluid, making soft membership models more appropriate (Bawa and Aponso, 2025). Neural approaches capture behavioural complexity but reduce transparency, limiting developer control (Xue et al., 2017; Mortazavi, 2024). Opaque or poorly explained adaptations can undermine player agency and immersion (Hunicke and Chapman, 2004; Chorrojprasert, 2025). Although Flow theory defines an optimal balance between challenge and skill, existing systems rarely implement this in a stable and interpretable manner (Wilson et al., 2019).

1.3. Problem definition

Current adaptive PCG approaches remain limited both methodologically and architecturally. Static systems fail to accommodate individual player differences, while reactive DDA introduces instability and inconsistent gameplay responses. Early adaptation without sufficient data leads to cold-start bias, distorting behavioural models. From a modelling perspective, hard clustering oversimplifies behaviour, whereas black-box neural models sacrifice interpretability. Additionally, abrupt or opaque adaptations can reduce player agency and disrupt immersion. Although Flow theory provides a theoretical foundation for balancing challenge and skill (Wilson et al., 2019), existing systems struggle to operationalise this within a stable, interpretable, and real-time adaptive framework.

1.3.1. Problem statement

Current real-time procedural content generation and adaptive difficulty systems lack a calibration-separated, behaviourally interpretable, and stability-preserving architecture capable of delivering consistent and agency-respecting environment adaptation in modern interactive environments.

1.4. Research motivation

Developing a stable and interpretable real-time adaptive procedural system is technically challenging. Such systems must process live telemetry, infer evolving player behaviour, and update environment parameters without introducing instability or excessive computation overhead. Maintaining a balance between responsiveness and stability requires careful temporal aggregation of telemetry, while accurately modelling behaviour adds further complexity. Neural approaches provide flexibility but reduce interpretability, whereas rule-based systems remain transparent but lack adaptability. Addressing these trade-offs within a calibration-separated, behaviour-aware framework under real-time constraints requires carefully structured modelling and architectural design.

1.5. Existing work

A full summary of each work, including individual contributions and identified limitations, is provided in *Section A.1*.

The reviewed literature demonstrates a progression from static procedural methods toward data-driven and hybrid adaptive systems. Key works include player-telemetry-driven map generation (Yu, 2024), a systematic taxonomy of DDA approaches (Mortazavi, 2024), fuzzy control for adaptive difficulty (Lara-Alvarez et al., 2021), and hybrid RL-clustering approaches to cold-start adaptation (Zhang and Goh, 2021). A full summary of each work, including individual contributions and identified limitations, is provided below.

1.5.1. Critical Review

The literature demonstrates a progression from static PCG toward adaptive, data-driven systems.

While static methods are scaled effectively, they lack personalisation. Rule-based DDA offers transparency but limited expressiveness, whereas reinforcement and deep learning approaches enable stronger adaptation at the cost of high computational overhead, limited interpretability, and poor real-time suitability. Neuro-fuzzy systems improve interpretability and uncertainty handling but often require expert configuration and are rarely integrated within calibration-separated architectures. Across all approaches, cold-start stabilisation and behavioural baseline separation remain insufficiently addressed. Collectively, these limitations highlight the absence of a solution that is stable, interpretable, latency-aware, and suitable for real-time adaptive gameplay.

1.6. Research gap

The following gaps were identified through a critical review of the literature in the Problem Domain (Adaptive PCG in Games) and the Research Domain (Neuro-Fuzzy Hybrid AI Systems):

1.6.1. Gaps in Problem Domain

1. **Cold-start instability and baseline contamination:** Research shows that adaptive systems struggle with cold-start bias because early-session data is insufficient, and defining a stable data window remains unresolved.(Caldas et al., 2020; Fernandes, Lopes and Prada, 2024) Immediate adaptation risks contaminating baseline measurements and triggering unstable control loops.
2. **Limited real-time stable adaptive pipelines:** Existing adaptive frameworks report ongoing challenges in ensuring that continuous adaptation does not harm runtime stability or system responsiveness (Caldas et al., 2020; Lopes, Fachada and Fonseca, 2025). Real-time deployment within interactive environments remains a practical bottleneck.

1.6.2. Gaps in Research Domain

1. **Rigid behavioural categorisation and lack of fluidity modelling:** Binary and hard clustering approaches cannot capture within-session behavioural shifts or short-term preference changes (Fernandes, Lopes and Prada, 2024; Bawa and Aponso, 2025). Dynamic soft membership modelling remains underexplored.
2. **Unresolved integration of explainable hybrid reasoning in adaptive systems:** Recent

surveys show that unified neural-symbolic architectures capable of balancing adaptability and interpretability remain an open research challenge (Alyaseri, 2025; Liang, Wang and Tong, 2025). Real-time explainable adaptive control in PCG contexts is still limited.

1.6.3. Summary

Existing adaptive game systems typically rely on static rules, opaque black-box models, or reactive strategies lacking a neutral behavioural baseline. Moreover, prior work rarely integrates calibration-first profiling, overlap-aware archetype modelling, interpretable neuro-fuzzy control, and low-latency runtime inference within a unified architecture. This highlights a clear gap for a system that is both behaviourally grounded and practically deployable for real-time gameplay adaptation.

1.7. Contribution to body of knowledge

This dissertation addresses the identified gap through AURA, a calibration-first adaptive framework integrating behavioural clustering, soft memberships, offline neuro-fuzzy modelling, and a runtime surrogate for real-time environment adaptation. Its contribution is twofold: it introduces an interpretable adaptive pipeline in the research domain and demonstrates how telemetry-driven behavioural inference can dynamically update procedural gameplay parameters in the problem domain. The neuro-fuzzy component is realised as a canonical ANFIS-inspired adaptation surface whose reasoning logic is approximated at runtime by an MLP surrogate to meet real-time latency constraints.

1.7.1. Problem Domain Contribution

- **Calibration-Separated Real-Time Adaptive Framework:** This research introduces a structured calibration-first architecture that explicitly separates baseline stabilisation from live adaptation. By using a fixed telemetry window, enforcing phase separation, and preventing continuous runtime retraining, the system directly addresses unresolved challenges around data window definition and runtime stability (Caldas et al., 2020). This contributes a deployable stabilisation strategy for real-time adaptive systems.

- **Session-Level Behavioural Adaptation with Soft Membership:** The proposed framework uses window-based telemetry aggregation and soft behavioural membership modelling to capture within-session preference shifts. This puts into practice the dynamic clustering principles suggested by recent research (Bawa and Aponso, 2025) and provides a working implementation of fluid behavioural modelling in interactive environments.
- **Latency-Aware Backend-Mediated Adaptive Deployment:** By decoupling inference from the game engine and running a surrogate reasoning model through a backend pipeline, this work provides a real-time stable adaptive architecture aligned with the responsiveness limitations identified in ML-driven adaptive systems (Lopes, Fachada and Fonseca, 2025). The bounded multiplier mechanism ensures controlled procedural modulation without destabilising gameplay.
- **Agency-Preserving Forward-Only Adaptation:** The framework enforces forward-only procedural modulation and bounded parameter adjustment, preserving perceptual continuity and player agency. This contributes a practical design principle that addresses ethical and trust-related concerns raised in adaptive difficulty literature (Kyiv, Ukraine and Sulyma, 2025).

1.7.2. Research Domain Contribution

- **Neuro-Fuzzy Surrogate for Interpretable Adaptive Control:** This research introduces a hybrid reasoning architecture combining behavioural clustering, fuzzy inference, and neural surrogate approximation, addressing challenges in neural-symbolic integration and explainability in adaptive AI systems (Alyaseri, 2025; Liang, Wang and Tong, 2025).
- **Temporal Delta-Based Behavioural Momentum Modelling:** The use of consecutive-window deltas extends static clustering into temporal modelling, enabling tracking of behavioural changes over time and supporting dynamic, session-aware adaptation (Bawa and Aponso, 2025).
- **Stability-Constrained Adaptive Control Strategy:** By integrating fixed observation windows, calibration isolation, bounded outputs, and smoothing, the framework provides a stability-aware adaptive control approach. This combination was not identified in the reviewed adaptive PCG literature, positioning the architecture as a structured contribution to stability-aware adaptive design (Caldas et al., 2020).

- **PAD-Grounded Archetype Formalisation**: Maps archetypes to PAD (Pleasure, Arousal, Dominance) dimensions, establishing a theoretically grounded behavioural ontology for adaptive PCG (Yu, 2024). Such a formalisation was not identified in the reviewed literature.

1.8. Research challenges

- **Stabilising Adaptation Without Contaminating the Baseline:** Separating calibration from real-time adaptation required preventing early telemetry from being affected by adaptive intervention. Defining statistically neutral baseline conditions and freezing calibration artefacts without disrupting later inference was a significant methodological challenge.
- **Determining an Optimal Telemetry Window:** Selecting a fixed 30-second window required balancing responsiveness against stability. Short windows amplified noise and induced oscillation and longer windows delayed adaptation. Finding the right size required iterative experimental validation.
- **Modelling Behavioural Fluidity Without Hard Classification:** Hard clustering produced abrupt behavioural switches and misclassification during transitional playstyles. Designing a soft membership model with continuous archetype proportions, while keeping it interpretable, was a core modelling challenge.
- **Integrating Temporal Deltas Without Amplifying Noise:** Adding behavioural momentum through consecutive-window deltas risk amplifying signal noise. Ensuring that deltas improved sensitivity without causing oscillatory adaptation required controlled scaling, bounded output design, and smoothing constraints.
- **Deploying Neuro-Fuzzy Reasoning Under Real-Time Constraints:** Running ANFIS directly within the game engine proved computationally unsuitable. Designing a surrogate neural approximation of the fuzzy reasoning surface, while keeping outputs bound and interpretable, required architectural restructuring and a backend-mediated inference approach.
- **Preserving Player Agency While Enabling Real-Time Control:** Adaptive systems risk disrupting perception when modifying active game states. Enforcing forward-only procedural modulation, bounded multipliers, and temporal smoothing was necessary to prevent

retroactive manipulation and maintain fairness and immersion.

1.9. Research questions

Based on the identified gaps and challenges, the following research questions guide this study:

- **RQ1:** How can a calibration-first architecture reduce cold-start bias and stabilise real-time adaptive procedural content generation?
- **RQ2:** What telemetry aggregation and temporal modelling strategy produces responsive yet stable behavioural adaptation in real time?
- **RQ3:** Can soft behavioural clustering combine with temporal deltas to effectively capture within-session behavioural fluidity without triggering oscillatory control dynamics?
- **RQ4:** How can neuro-fuzzy reasoning be approximated and deployed in a latency-aware architecture while preserving interpretability, bounded adaptation, and player agency?

1.10. Research aim

The aim of this research is to design, implement, and evaluate a calibration-first, telemetry-driven adaptive control framework capable of delivering stable, interpretable, and real-time procedural environment modulation. The study seeks to reduce cold-start bias, model behavioural fluidity through soft clustering and temporal dynamics, and deploy neuro-fuzzy surrogate reasoning within a latency-aware backend architecture, while preserving player agency and perceptual continuity during adaptation.

1.11. Research objectives

The table below presents the fourteen research objectives (R01–R14), each mapped to its project phase and corresponding research question. Full learning outcome mappings and objective descriptions are provided in *Section A.2* for the full research objectives mapping table.

ID	Phase	Research Objective	RQ
R01	Problem ID	Identify and formalise limitations of existing adaptive PCG systems, including cold-start bias, instability, and interpretability gaps	RQ1
R02	Problem	Define a calibration-first adaptive architecture establishing phase-	RQ1

	ID	separation between baseline acquisition and live inference	
R03	Lit. Review	Conduct a comprehensive critical literature review on DDA, adaptive PCG, telemetry-driven modelling, and neuro-fuzzy control	RQ1, RQ4
R04	Lit. Review	Investigate soft clustering and neuro-symbolic reasoning strategies for interpretable behavioural fluidity modelling	RQ3, RQ4
R05	Req. Elicit.	Identify system-level requirements for real-time adaptive control, including behavioural modelling, latency, and bounded adaptation constraints	RQ2, RQ4
R06	Design	Design the three-phase AURA adaptive architecture ensuring stability, forward-only adaptation, and backend decoupling	All RQs
R07	Design	Design the telemetry feature engineering and temporal delta modelling strategy, including the 30-second window and behavioural momentum signals	RQ2, RQ3
R08	Implement	Implement K-Means behavioural clustering with IDW soft membership and partition-of-unity constraint	RQ3
R09	Implement	Implement the ANFIS offline training pipeline and MLP surrogate approximation for runtime inference	RQ4
R10	Implement	Integrate the backend-mediated adaptive inference pipeline with the Unreal Engine client via VaRest HTTP	RQ2, RQ4
R11	Testing	Evaluate adaptation stability, multiplier boundedness, and oscillation absence across testing conditions	RQ2, RQ3
R12	Testing	Evaluate backend inference performance and round-trip latency against the sub-200 ms threshold	RQ2
R13	Testing	Evaluate the interpretability of behavioural archetype proportions and adaptive decisions through expert evaluation	RQ4
R14	Sharing	Document and disseminate research findings through academic publication area of adaptive AI for games. Dataset and reproducible pipeline notebook published	All RQs

Table 1: Research Objectives

1.12. Project scope

This section defines the boundaries of the research in terms of what is included and excluded from the implementation and evaluation of the proposed adaptive framework.

1.12.1. In scope

The in-scope boundary covers the complete AURA pipeline, including calibration, behavioural modelling, backend inference, and bounded real-time adaptation. Performance evaluation aspects such as latency, FPS impact, and multiplier stability are also included. A detailed breakdown of in-scope items is provided in *Section A.3*.

1.12.2. Out of scope

The out-of-scope boundary excludes features such as in-engine ANFIS execution, biometric sensing, reinforcement learning-based generation, multiplayer scenarios, and large-scale production optimisation. A detailed list of exclusions is provided in *Section A.4*.

1.13. Hardware/software requirements

All software tools were selected to align with the three-phase pipeline architecture. Full specifications, version numbers, and justification for each tool selection are documented in the appendix.
See, Section A.5 for hardware specifications and Section A.6 for the full software requirements table.

1.14. Chapter summary

This chapter established the research context for AURA by identifying limitations in static PCG and reactive DDA systems, including cold-start bias, instability, and lack of behavioural interpretability. The research problems and gaps were defined across both problem and research domains, followed by the proposed contributions. Key research challenges, questions, aims, and objectives were presented and aligned with the three-phase AURA architecture. Finally, the project scope was defined to outline the implementation boundaries.

CHAPTER 02: LITERATURE REVIEW

2.1. Chapter overview

Reviews key foundations of adaptive procedural content generation and behavioural modelling, including DDA, telemetry-driven modelling, clustering, and neuro-fuzzy systems. It critically analyses existing approaches to identify limitations and defines research gaps that motivate a calibration-first, interpretable, and stability-aware AURA framework.

2.2. Concept Map

A concept map was constructed to map the theoretical relationships among the core areas reviewed in this chapter: adaptive procedural content generation frameworks, telemetry-driven player modelling paradigms, behavioural clustering approaches, neuro-fuzzy hybrid reasoning systems, and evaluation methodologies. The map provides a structured overview of how these areas interconnect and collectively define the research gap addressed in this study. The full concept map is provided in *Appendix B*Appendix B - Concept Map.

2.3. Problem domain

Adaptive Procedural Content Generation (APCG) emerges at the intersection of procedural generation, player modelling, adaptive control, and ethical system design. While static PCG and reactive DDA frameworks have advanced engagement management, methodological instability, behavioural rigidity, and agency violations remain unresolved challenges. This section synthesizes the core theoretical and methodological foundations defining the problem domain.

2.3.1. Overview of adaptive procedural content generation

Limitations of static and rule-based PCG

Traditional procedural content generation systems rely heavily on fixed probability distributions, handcrafted heuristics, and rule-based parameterization. Such systems suffer from scalability limitations often described as the “mountain of content problem,” where manual tuning restricts expressive diversity (Freiknecht and Effelsberg, 2017).

Static difficulty curves further exacerbate this issue. Since players exhibit heterogeneous skill levels, learning rates, and motivational patterns, fixed configurations frequently result in mismatched challenge states either trivializing the experience or inducing frustration (Xue et al., 2017). Methodologically, static PCG cannot account for temporal skill evolution, producing generic outputs that lack personalization (Yu, 2024).

Emergence of Adaptive Procedural Content Generation (2024–2025 definitions)

Recent literature (Yu, 2024) reframes APCG as a closed-loop system integrating telemetry sensing, player modelling, and dynamic content modulation. Rather than generating content in isolation, APCG dynamically adjusts generated levels, encounters, and pacing in response to observed behavioural patterns (Yu, 2024; Alyaseri, 2025; Lopes, Fachada and Fonseca, 2025; Mi and Gao, 2025).

Contemporary APCG frameworks are characterised by four interdependent design properties: real-time monitoring of player state, **behaviour-aware content generation**, **engagement optimisation objectives**, and **closed-loop feedback** between sensing and output (Yu, 2024; Lopes, Fachada and Fonseca, 2025). When all four properties are active, the system can detect shifts in player behaviour and translate those shifts into meaningful content adjustments without manual intervention. The closed-loop structure distinguishes APCG from earlier **data-driven PCG** approaches, which focused primarily on aesthetic generation without **adaptive intent** (Alyaseri, 2025).

Generative AI techniques have further positioned APCG as a data-driven system capable of learning content distributions from historical designs and adapting them dynamically with minimal manual intervention (Alyaseri, 2025).

Stability risks in reactive adaptive systems

Reactive DDA systems that lack baseline calibration are vulnerable to oscillatory feedback loops (Xue et al., 2017). When adaptation responds immediately to transient performance spikes (e.g., a single failure), difficulty may fluctuate excessively creating “rubber-banding” effects perceived as manipulative (Chorrojprasert, 2025).

Such oscillation arises from three compounding effects: short-term noise is treated as a persistent skill change, overcorrection induces alternating difficulty extremes, and the absence of a **neutral reference frame** prevents the system from anchoring its inference to stable behaviour. Without a **calibrated baseline**, the system cannot distinguish between a temporary performance lapse and a genuine shift in **skill level**. Purely reactive systems therefore tend to respond to the wrong signal, reinforcing the wrong trajectory and further degrading model quality over time (Bontchev, 2016; Chorrojprasert, 2025)

Calibration-based vs immediate reactive pipelines

Calibration-based adaptation introduces a preliminary observation phase often termed a "sleep phase" during which telemetry is collected without intervention (Mi and Gao, 2025). This produces a neutral behavioural reference prior to adaptive control.

In contrast, immediate reactive DDA modifies parameters during active gameplay, reacting to short-term triggers such as death events or sudden score drops. This approach risks perceptual discontinuity because players can perceive the game as adjusting to catch them rather than responding authentically to their skill. Calibration-based approaches aim for neutral parameterisation to collect unbiased data, while reactive pipelines often act on transient signals that do not reflect **persistent behavioural patterns** (Kowalik and Kapusta, 2025; Sakyev, 2025).

2.3.2. Importance of adaptive PCG and player modelling

Telemetry-driven modelling and engagement retention

Large-scale empirical studies demonstrate a correlation between telemetry-driven adaptation and increased engagement retention. Adaptive difficulty frameworks have shown measurable improvements in long-term retention metrics, including D30 retention increases and extended lifecycle participation (Kyiv, Ukraine and Sulyma, 2025).

Telemetry enables churn risk prediction, skill-challenge balance maintenance, and **personalised pacing adjustments**, establishing **behavioural modelling** as central to the viability of **adaptive PCG** (Xue et al., 2017; Kyiv, Ukraine and Sulyma, 2025).

Beyond engagement prediction, telemetry-driven modelling allows systems to build stable longitudinal profiles of player behaviour. By sampling behavioural states at regular intervals rather than reacting to discrete events, these systems produce a continuous model of player intent that is less sensitive to momentary fluctuation (Bevilacqua, Engström and Backlund, 2019). Research suggests that **30-second aggregation windows** represent a practical balance between **temporal resolution** and **noise suppression** for skill-based gameplay environments (Kowalik and Kapusta, 2025).

Intent-based vs outcome-based metrics

Outcome-based metrics (win/loss, score, deaths) provide coarse performance indicators but fail to capture underlying decision processes (Xue et al., 2017). Intent-based metrics analyze behavioural trajectories such as navigation paths, interaction density, and action frequency (Etheredge, Lopes and Bidarra, 2013). While Outcome-based metrics are computationally efficient, they can misclassify player experiences (e.g., a player wins but feels bored), intent-based metrics can help distinguish between a player who is exploring the environment and a player who is efficiently rushing to a goal. Table 2: Comparison of behavioural metric paradigms compares these paradigms.

Dimension	**Outcome-Based**	**Intent-Based**
Captures Strategy	Limited	High
Sensitivity to Noise	Moderate	Lower
Interpretability	Surface-level	Behaviourally grounded
Adaptation Quality	Reactive	Context-aware

Table 2: Comparison of behavioural metric paradigms

Hard clustering limitations

Hard clustering assigns players to single rigid archetypes, failing to represent behavioural fluidity (Etheredge, Lopes and Bidarra, 2013). Since players exhibit mixed motivations and evolving playstyles, rigid classification can induce misaligned adaptations.

Specific limitations include **boundary misclassification** at archetype thresholds, loss of nuance for players who exhibit mixed motivations, and an inability to model **hybrid behaviours** where multiple archetypes are simultaneously active (Etheredge, Lopes and Bidarra, 2013; Kanwal, Siddiqui and Ali, 2025).

These limitations are particularly significant in gameplay contexts where players routinely shift strategy mid-session. A player who opens aggressively may transition to exploratory behaviour once resources are secured, yet a **hard-clustering model** permanently assigns them to a single archetype, triggering inappropriate adaptations for the remainder of the session. This **structural rigidity** creates a persistent mismatch between classified identity and actual behaviour, reducing both **adaptation accuracy** and **player satisfaction**.

Soft membership modelling

Soft clustering allows partial membership across archetypes, capturing behavioural fluidity. Degree-based representation enables smooth transitions rather than abrupt profile shifts.

This approach supports hybrid behavioural modelling, continuous adaptation, and relative player profiling, making it better aligned with the dynamic and evolving nature of player behaviour (Etheredge, Lopes and Bidarra, 2013).

Soft membership also reduces the sensitivity of the adaptation system to cluster boundary effects, where small changes in a player's behaviour could previously cause a disproportionate parameter adjustment. By maintaining a proportional weighting across archetypes, the adaptation signal varies smoothly rather than discretely, supporting the gradual content modulation associated with positive player experience (Maddalon et al., 2024). This smooth-transition property makes soft clustering particularly suited to decoupled backend pipelines where adaptation parameters are computed incrementally between game events.

Application areas of adaptive PCG

The PAD (Pleasure, Arousal, Dominance) model (Mehrabian and Russell, 1974) represents emotional states across three dimensions. Pleasure reflects emotional valence and influences

approach or avoidance behaviour (Bontchev, 2016). Arousal indicates the level of mental activation and physiological alertness (Khoshnoud, Alvarez Igarzábal and Wittmann, 2020). Dominance represents the player's perceived control within the environment (Bontchev, 2016; Bran and Vaidis, 2020).

In interactive systems, misalignment between gameplay and player affect reduces engagement. Difficulty mismatches lead to frustration and early disengagement (Bontchev, 2016), whereas maintaining affective alignment improves emotional outcomes in adaptive systems (Kowalik and Kapusta, 2025). This aligns with flow theory, where optimal engagement occurs under positive Pleasure and appropriate Arousal levels (Khoshnoud, Alvarez Igarzábal and Wittmann, 2020). The 75% engagement drop in non-adaptive procedural systems (Bontchev, 2016) can be attributed to failure in maintaining this balance.

The PAD model is suitable for telemetry-driven systems as its dimensions can be inferred from behavioural data such as movement and interaction patterns (Yannakakis and Togelius, 2011; Yu, 2024). This supports its use in AURA, where behavioural archetypes such as Combat, Collection, and Exploration represent observable patterns aligned with PAD dimensions, enabling adaptive responses without intrusive measurement.

2.3.3. Application areas of adaptive PCG

Telemetry-driven adaptive PCG has been evaluated across a wide range of interactive domains, including platformers, massively multiplayer online role-playing games, exergames, educational systems, first-person and third-person shooters, racing games, and virtual reality environments (Bontchev, 2016; Xue et al., 2017; Kaimara, Oikonomou and Deliyannis, 2021).

In each domain, the core adaptation targets remain broadly consistent: procedural modulation is applied to enemy spawn rates, resource availability, **environmental geometry, AI aggressiveness**, and **pacing mechanisms** in response to observed behavioural patterns (Hunicke and Chapman, 2004; Bontchev, 2016).

Architectural patterns

Three primary inference architectures have been identified in the literature. Embedded inference runs adaptation logic directly within the game client, using on-device fuzzy controllers or heuristic rules for low-latency response (Mortazavi, 2024). Backend-mediated pipelines offload player modelling and content generation to a remote server, enabling greater computational capacity at the cost of network latency (Mortazavi, 2024). Hybrid and federated models train across distributed devices to preserve user privacy while updating a shared global model (Zhang et al., 2025). Backend-mediated pipelines enable computational scalability but require **latency-aware design** to remain viable in real-time gameplay.

Backend-mediated pipelines enable computational scalability but require latency-aware design.

2.3.4. Challenges in adaptive PCG

Cold-start bias

Cold-start bias arises when insufficient data exists for accurate personalization (Xue et al., 2017). Early mismatches may distort player modelling and contaminate future inference.

Mitigation strategies include **neutral default parameterisation** to prevent early distortion, bootstrapping via population-level priors to approximate initial player state, and **calibration-phase** isolation to ensure that early telemetry is collected without **adaptation influence** (Xue et al., 2017; Bevilacqua, Engström and Backlund, 2019). Without such strategies, early mismatches in **player modelling** may persist and compound through repeated adaptive cycles, reducing the long-term accuracy of the system.

Oscillatory feedback loops

Reactive systems fail when they overcorrect for player skill, creating **oscillatory feedback loops** (Bontchev, 2016). This "pendulum effect" swings difficulty between trivial and punishing, breaking the player’s flow and making the game's logic feel frustratingly obvious (Bontchev, 2016).

Temporal aggregation trade-offs

Short aggregation windows increase responsiveness but amplify noise. Long windows stabilize behaviour but delay adaptation (Xue et al., 2017; Huo et al., 2023). Balancing this aggregation window is therefore a central design decision in any telemetry-based adaptive system, as both extremes introduce different forms of modelling error that degrade adaptation quality over time.

Ethical and agency risks

Retroactive world-state modification undermines fairness perception and player trust (Xue et al., 2017). Covert adaptation may reduce agency and violate immersion expectations. Ethical concerns span perceived manipulation of player outcomes, exploitation of adaptation mechanisms as **dark-pattern monetisation** tools, and the use of **inconsistent rule sets** that undermine **fairness expectations** (Xue et al., 2017). Addressing these concerns requires explicit design constraints around the range and transparency of **adaptive authority**.

2.3.5. Problem-domain framing of adaptive architecture

The preceding subsections reveal that instability, cold-start bias, behavioural rigidity, and interpretability limitations remain persistent weaknesses in contemporary adaptive procedural systems. This subsection synthesizes architectural requirements derived from the problem domain rather than describing implementation details.

Necessity of separating calibration from adaptation

Literature consistently demonstrates that immediate adaptation contaminates early behavioural signals, leading to biased inference and unstable feedback loops (Bontchev, 2016). When adaptation begins during initial exposure, transient experimentation behaviour may be encoded as persistent player traits, amplifying modelling errors through recursive control.

Separating baseline acquisition from adaptive inference provides three methodological safeguards:

1. Cold-start mitigation - prevents premature overfitting to sparse early data.
2. Baseline neutrality - ensures behavioural clustering is grounded in uncontaminated telemetry.
3. Reference stability - provides a fixed anchor against which future behavioural deltas can be measured.

Calibration therefore functions as a control condition, preserving internal validity in adaptive systems (Mi and Gao, 2025). Without such separation, player modelling risks self-reinforcing distortion.

Requirement for interpretable adaptive reasoning

The literature highlights the tension between predictive performance and interpretability in adaptive systems. While deep neural models offer flexibility, their opaque reasoning undermines transparency, fairness auditing, and developer control (Leong, 2025).

Neuro-fuzzy systems offer a hybrid alternative by encoding interpretable IF-THEN rules, supporting gradual membership transitions, allowing bounded parameter control, and preserving **human-readable reasoning structures**. Compared to purely **black-box approaches**, neuro-fuzzy inference improves the **traceability** of adaptive decisions while retaining sufficient learning capacity (Bontchev, 2016; Xue et al., 2017; Talpur et al., 2022).

In problem-domain terms, interpretability is not merely a technical preference but a prerequisite for:

- Ethical adaptation
- Agency preservation
- Controlled procedural modulation.

Need for computationally efficient reasoning layers

Real-time adaptive PCG must operate under strict latency constraints. Literature identifies surrogate modelling as a viable strategy for approximating computationally intensive simulations while preserving inference fidelity (Xue et al., 2017).

Surrogate reasoning provides deterministic latency bounds, reduced runtime overhead, deployment portability across platforms, and scalability across distributed systems. Within the problem domain, this implies that any adaptive framework must reconcile reasoning complexity with real-time responsiveness (Liu et al., 2021).

Control-theoretic constraints and bounded adaptation

Oscillatory feedback loops and perceptual instability are frequently attributed to unbounded parameter modulation (Pianini and Kalogeraki, 2023). Control-theoretic approaches introduce saturation limits, rate constraints, and safe operating intervals to ensure adaptation remains within perceptually acceptable bounds.

Forward-only adaptation strategies further reduce regression effects that may undermine perceived competence (Huo et al., 2023). By constraining difficulty modulation to bounded ranges, systems prevent excessive overshoot past the stable difficulty target, regressive simplification that undermines player achievement, and chaotic fluctuation that breaks the perceived logic of the adaptive system. Stability therefore emerges not only from modelling accuracy but from **constrained adaptive authority**.

Temporal delta modelling and behavioural momentum

Static behavioural snapshots are insufficient for modelling evolving player skill. Temporal delta modelling emphasizes behavioural trajectory rather than instantaneous state (Xue et al., 2017).

This distinction enables systems to detect learning momentum, identify **fatigue-driven performance decline**, separate transient noise from **structural behavioural change**, and prevent the **static misclassification** of evolving players. By focusing on change rather than static proportion alone, adaptive systems gain the **temporal sensitivity** essential for stable control (Xue et al., 2017; Huo et al., 2023).

Architectural synthesis of problem-domain requirements

From the reviewed literature, five architectural requirements emerge:

Requirement	**Justification Source**
Calibration-first baseline	Cold-start mitigation (Mi and Gao, 2025)
Soft behavioural modelling	Fluidity representation (Etheredge, Lopes and Bidarra, 2013)
Interpretable reasoning	Transparency requirement (Bontchev, 2016)

Bounded forward-only control	Stability & fairness (Liu et al., 2021)
Temporal delta integration	Momentum detection (Xue et al., 2017)

Table 3: Architectural Requirements

Collectively, these requirements define the methodological constraints that any stable real-time adaptive procedural framework must satisfy.

2.4. Existing work

This section critically reviews existing adaptive game systems and procedural content generation approaches. The discussion focuses on four dominant paradigms: rule-based Dynamic Difficulty Adjustment (DDA), reinforcement learning-based adaptation, telemetry-driven behavioural modelling, and neuro-fuzzy control systems. The aim is to identify methodological strengths and structural limitations that motivate the proposed framework.

2.4.1. Rule-Based and Threshold-Driven DDA Systems

Rule-based DDA systems modify game parameters using predefined thresholds. For example, if player accuracy drops below a certain value, enemy difficulty is reduced. These systems are widely adopted due to interpretability and implementation simplicity (Kyiv, Ukraine and Sulyma, 2025; Leong, 2025).

A hybrid balancing framework combining heuristic rules with fairness metrics such as win-rate distribution was proposed to improve game balance equity (Leong, 2025). Heuristic adaptation was applied across 330,000 players, introducing a cold-start phase to initialise baseline win rates, though manual rule authoring and limited scalability remained constraints (Kyiv, Ukraine and Sulyma, 2025). Although these systems improved retention, they required manual rule authoring and lacked scalability.

A central limitation is oscillatory behaviour. When thresholds are crossed repeatedly, difficulty increases and decreases abruptly, producing instability (Hunicke, 2005; Kyiv, Ukraine and Sulyma, 2025).

This oscillatory phenomenon is conceptually illustrated in Figure 1: Oscillatory behaviour in threshold-based dynamic difficulty adjustment, comparing static progression, threshold-triggered oscillation, and a smoothed adaptive response.

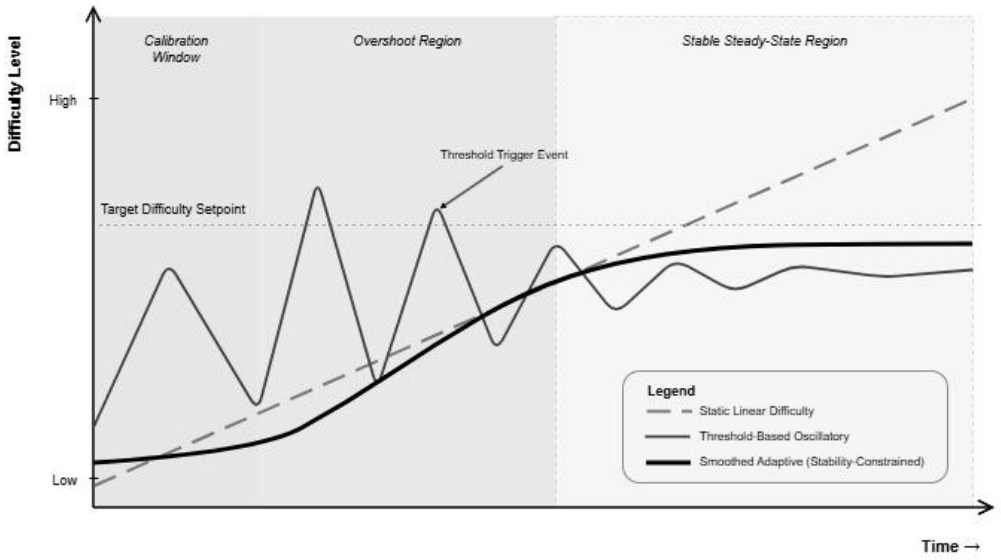


Figure 1: Oscillatory behaviour in threshold-based dynamic difficulty adjustment (Self Composed)

The reactive threshold model exhibits repeated overshoot before reaching a steady-state region, whereas smoothed control mechanisms reduce instability.

This instability can be described using feedback control theory. Adaptive controllers often follow the proportional integral derivative (PID) formulation:

$$u(t) = K_p e(t) + K_i \int e(t)dt + K_d \frac{de(t)}{dt}$$

Where:

- $e(t)$represents performance deviation from the target challenge level.
- $K_p$controls immediate correction,
- $K_i$accumulates past error,
- $K_d$predicts future trend.

Improper tuning leads to overshooting, defined as:

$$M_p = \frac{y_{max} - y_{ss}}{y_{ss}}$$

The control-theoretic behaviour of adaptive difficulty systems can be visualised using a classical PID response curve, as shown in Figure 2: PID Control Response in Dynamic Difficulty Adjustment.

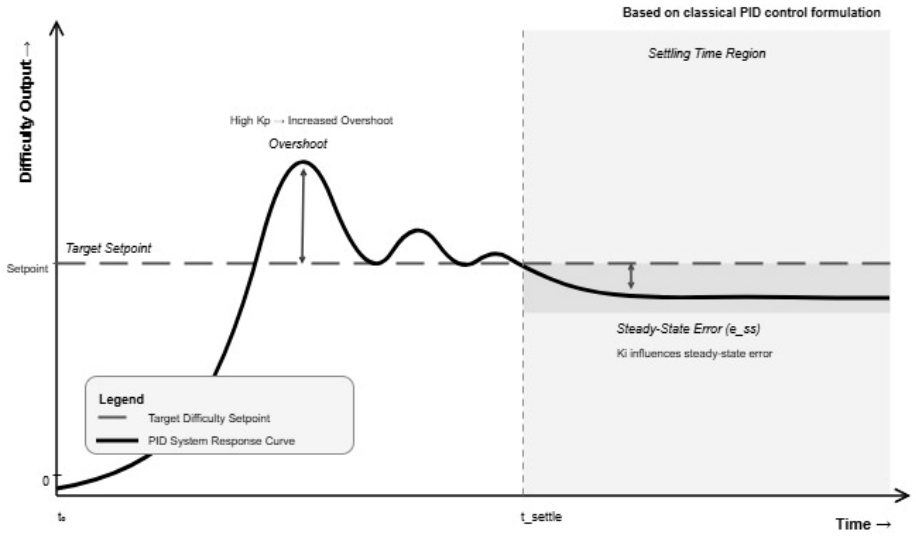


Figure 2: PID Control Response in Dynamic Difficulty Adjustment (Self Composed)

The figure highlights overshoot, settling time, and steady-state error, which are critical indicators of adaptive stability.

Overshoot reflects how much difficulty exceeds the stable target. Healthcare adaptive prototypes using hybrid PI control restricted overshoot below 3% (Caldas et al., 2020), but such stability mechanisms are rarely embedded in game DDA frameworks.

Additionally, static difficulty curves assume a median player's progression and fail to accommodate heterogeneous learning trajectories, completely failing to account for the massive diversity in player experiences, learning rates, and playing styles (Xue et al., 2017). This misalignment contributes to boredom or frustration.

2.4.2. Reinforcement Learning-Based Adaptive PCG

Reinforcement Learning (RL) has been used to generate adaptive content policies. G-PCGRL models content as graph structures and modifies edges through reward-guided optimisation (Rupp and Eckert, 2024). EDRL formulates level generation as a Markov Decision Process (MDP), optimising fun and playability metrics (Shu, Liu and Yannakakis, 2021). ARLPCG introduces adversarial co-training between generator and solver agents in 3D environments (Gisslen et al., 2021). PCGPT combines transformer networks with offline RL for structured level generation (Caldas et al., 2020).

While RL enables automated policy optimisation, several structural challenges remain unresolved. Deep reinforcement learning models have high data requirements, exhibit convergence instability during training, produce **opaque decision-making policies**, and require significant computational resources during inference (Gisslen et al., 2021; Alyaseri, 2025). RL systems operate as **black-box models**, limiting design interpretability and difficulty transparency.

From a deployment perspective, RL policies may fail to generalise to real player behaviours that deviate from training assumptions. This generalisation gap, combined with the opaque nature of learned policies, makes RL-based approaches difficult to debug and maintain in iterative game development workflows (Gisslen et al., 2021). These considerations suggest that RL-based adaptive PCG remains better suited to research prototypes than deployable **real-time game systems** at present.

2.4.3. Telemetry-Based Behavioural Modelling

Adaptive systems rely on telemetry to infer player state. Outcome-based metrics (e.g., win rate, survival time) measure observable performance misinterprets a player's intent. Intent-based modelling integrates physiological signals, stress indicators, or behavioural trajectories to capture underlying player state (Sakyev, 2025).

Window size significantly influences adaptation stability. Short windows increase responsiveness but amplify noise while longer windows increase stability but delay correction (Bevilacqua, Engström and Backlund, 2019).

To prevent early baseline contamination, several systems implement a “Sleep Phase” where behaviour is observed without adaptation (Mi and Gao, 2025). This approach improves modelling reliability.

Advanced models incorporate temporal deltas to capture behavioural momentum rather than static snapshots (Kowalik and Kapusta, 2025). Without such modelling, systems act reactively instead of predictively.

The temporal granularity of telemetry collection also determines what behavioural signals are visible to the modelling layer. Short aggregation windows may expose momentary strategic shifts but are vulnerable to noise from random performance variation, while longer windows produce smoother profiles at the cost of adaptation delay. Research consistently indicates that a balance between these extremes is necessary to accurately capture both the immediate context and the underlying **behavioural trajectory** of a player session (Bevilacqua, Engström and Backlund, 2019; Kowalik and Kapusta, 2025).

2.4.4. Behavioural Clustering Approaches

Clustering methods group players into archetypes for personalised adaptation. K-Means clustering assigns players to fixed categories based on reaction time, accuracy, and physiological features. However, hard assignment assumes stable identity and cannot capture behavioural fluidity (Xue et al., 2017).

Fuzzy clustering introduces soft membership across multiple archetypes. This reduces abrupt parameter shifts and better represents dynamic playstyle transitions (Maddalon et al., 2024).

Rigid archetype assignment may lead to misclassification when players evolve their strategy mid-session (Rodríguez, Puig and Rodríguez, 2022). Consequently, hard clustering can produce unstable adaptation behaviour.

The structural difference between hard and soft behavioural modelling is illustrated in Figure 3: Hard Clustering vs Soft (Fuzzy) Membership Modelling

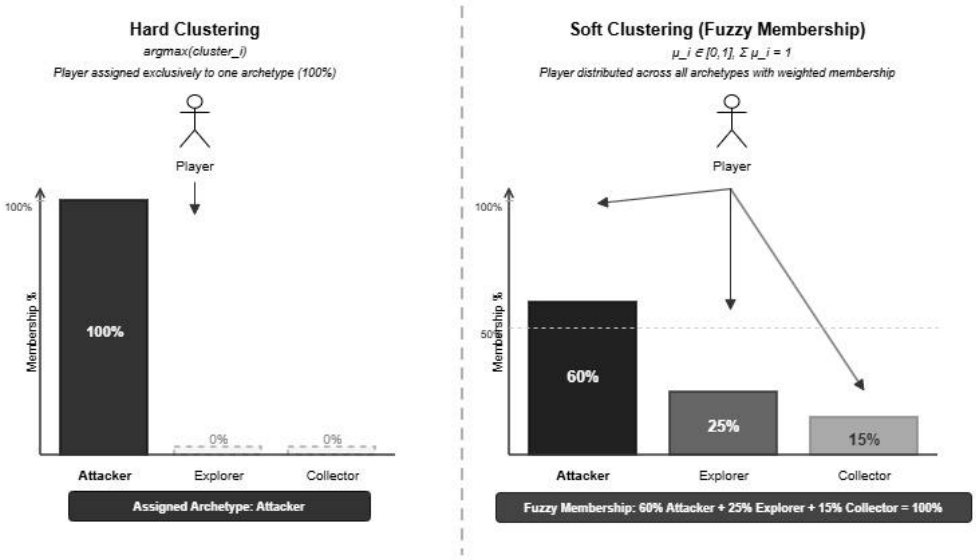


Figure 3: Hard Clustering vs Soft (Fuzzy) Membership Modelling (Self Composed)

Soft membership allows proportional representation across archetypes, reducing abrupt adaptation triggered by rigid classification.

2.4.5. Neuro-Fuzzy and Hybrid Reasoning Systems

Neuro-fuzzy systems combine rule-based reasoning with adaptive learning. Unlike purely rule based DDA or black-box neural networks, these systems encode interpretable IF–THEN rules while allowing parameters to be updated through data-driven optimisation (Mandryk and Atkins, 2007; Lara-Alvarez et al., 2021).

In adaptive game environments, fuzzy controllers have been used to regulate difficulty by mapping player state variables such as emotional indicators, score, or performance deviation to future challenge parameters (Lara-Alvarez et al., 2021). Empirical studies report that fuzzy control systems reduce persistence of negative affective states compared to linear threshold-based DDA mechanisms. The primary advantage lies in smooth parameter transitions. Rather than triggering abrupt difficulty jumps when a threshold is crossed, fuzzy membership functions apply gradual weighted adjustments (Mandryk and Atkins, 2007).

A standard fuzzy inference rule follows the structure:

$$\text{IF } x \text{ is } A \text{ AND } y \text{ is } B \text{ THEN } z = f(x, y)$$

Here, $x$and $y$represent behavioural indicators (e.g., performance error, engagement level), while $z$denotes the adapted difficulty parameter. This structure enhances interpretability because designers can inspect and validate rule logic.

However, deep neuro-fuzzy architecture introduces computational complexity. Sequential fuzzy neural models may become bottlenecked when processing large-scale gameplay telemetry, and some implementations require specialised hardware acceleration to maintain real-time responsiveness (Wheat et al., 2016; Talpur et al., 2022). To mitigate runtime overhead, surrogate models have been proposed to approximate complex reasoning surfaces using faster neural approximations (Schwalbe and Finzel, 2021). Nevertheless, most reviewed systems do not explicitly integrate calibration isolation or temporal delta modelling within their fuzzy reasoning layers.

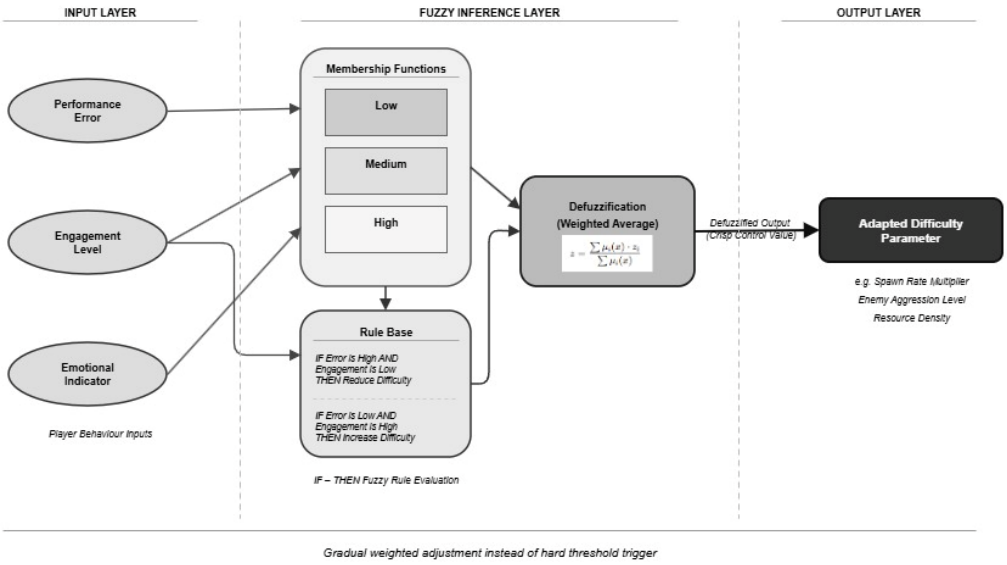


Figure 4: Conceptual Architecture of a Neuro-Fuzzy Adaptive Controller (Self Composed)

2.4.6. Summary of existing approaches and positioning of AURA

Below table synthesises the key limitations identified across the reviewed approaches and maps them to the architectural decisions motivating the AURA framework

Approach	Cold-start handling	Soft membership	Interpretable	Runtime deployable	Stable bounds
Rule-based / threshold DDA	None (static rules)	No	Partial	No (engine-only)	Yes (manual)
RL-based adaptive PCG	Warm-up episodes needed	No (hard states)	No	Limited (latency)	No (unstable rewards)
Telemetry-based modelling	Baseline session required	Partial	Partial	Varies	Partial
Hard K-Means clustering	None	No	No	Yes	No
ANFIS / Fuzzy inference	None formalised	Yes (fuzzy sets)	Yes	Slow at runtime	Partial
AURA (proposed)	Phase 1 calibration study	Yes (IDW-based)	Yes (MLP surrogate)	Yes (<200ms)	Yes ([0.6,1.4])

Table 4: Summary of existing approaches and AURA positioning

2.5. Synthetic telemetry dataset and data strategy

Adaptive gameplay systems rely on behavioural datasets, typically derived from game telemetry capturing player actions, interactions, and performance metrics (Lopes, Fachada and Fonseca, 2025). While some studies incorporate subjective profiling data, such as surveys, telemetry remains the primary source for objective behavioural analysis (Yu, 2024). However, publicly available datasets are limited and often lack contextual detail or consistency across different game environments, reducing their suitability for behavioural modelling (Pfau et al., 2020).

To address this, many studies **adopt controlled data collection** to ensure consistency and establish unbiased baseline behaviour prior to adaptation (Bawa and Aponso, 2025; Souza, Berretta and Carvalho, 2025). Accordingly, this study adopted the telemetry collection method using a purpose-built prototype aligned with the AURA behavioral scheme, ensuring consistent game conditions and reliable baseline calibration.

2.6. Benchmarking and evaluation

Evaluating adaptive gameplay systems requires systematic benchmarking to assess the effectiveness of behavioural modelling and adaptation mechanisms. This is typically performed using machine learning performance metrics alongside controlled gameplay evaluations, enabling validation of modelling accuracy and improvements over non-adaptive baseline systems (Pianini and Kalogeraki, 2023).

2.6.1. Evaluation metrics

Machine learning models used in behavioural modelling are evaluated using both classification and regression metrics. Classification metrics measure how well behavioural patterns are identified, while regression metrics measure how accurately predicted outputs match observed values.

Commonly used metrics include **accuracy, precision, recall, and F1-score** for behavioural classification tasks mean squared error (MSE) and root mean squared error (RMSE) for measuring prediction error magnitude. Mean absolute error (MAE) for calculating average prediction deviation and the **coefficient of determination (R^2)** for measuring how well the model explains variance in behavioural data (Pianini and Kalogeraki, 2023; Yu, 2024). These metrics allow researchers to evaluate **prediction accuracy** and determine how reliably the adaptive model responds to player behaviour.

2.6.2. Benchmarking approaches

Adaptive gameplay systems are typically benchmarked by comparing adaptive configurations with non-adaptive gameplay conditions. Controlled experiments or A/B testing are frequently used to observe differences in gameplay performance and player engagement (Kyiv, Ukraine and Sulyma, 2025). Evaluation often considers gameplay indicators such as completion time, success rates, interaction frequency, and player progression.

However, benchmarking adaptive systems presents several challenges. Player behaviour varies significantly across individuals, and subjective evaluation methods may introduce response bias. In addition, machine learning models may produce slightly different outputs across repeated

training runs due to stochastic learning processes, making strict benchmarking comparisons difficult (Yannakakis and Togelius, 2018).

Metric	Category	Purpose
Accuracy	Classification	Measures proportion of correct predictions
Precision / Recall	Classification	Evaluates behavioural category identification
F1-score	Classification	Balances precision and recall
MSE / RMSE	Regression	Measures prediction error magnitude
MAE	Regression	Measures average prediction deviation
R^2	Regression	Indicates variance explained by the model

Table 5: Common evaluation metrics used in adaptive systems

2.7. Chapter summary

This chapter synthesised advancements and limitations in adaptive PCG, telemetry-driven modelling, behavioural clustering, and neuro-fuzzy systems. While existing approaches improve engagement and stability, they often lack integration of real-time adaptability, interpretability, and calibration-first design. The review identified a key gap: the absence of a unified framework combining offline neuro-fuzzy reasoning with a lightweight surrogate while preserving player agency and system stability. The PAD (Pleasure, Arousal, Dominance) model was identified as the foundational psychological framework establishing why distinct behavioural archetypes represent meaningfully different adaptation targets, grounding the AURA archetype design in established affective theory (Mehrabian and Russell, 1974; Bran and Vaidis, 2020).

CHAPTER 03: METHODOLOGY

3.1. Chapter overview

Presents the methodological framework for designing, developing, and evaluating AURA, a three-phase calibration-first, telemetry-driven behavioural modelling and neuro-fuzzy surrogate system for real-time procedural environment adaptation via a decoupled backend pipeline.

3.2. Research methodology

This study is structured through Saunders' Research Onion model (Saunders, Lewis and Thornhill, 2003), which provides a layered framework for defining the core philosophical and procedural.

Layer	**Choice**	**Justification**
Philosophy	Pragmatism	The research integrates both quantitative telemetry-based model metrics and qualitative player engagement survey data to address the shared nature of knowledge and test predefined hypotheses about adaptive behavioural modelling.
Approach	Deductive	As the study tests predefined hypotheses derived from flow theory, K-Means clustering, and adaptive gameplay literature against empirically collected telemetry evidence.
Methodological Choice	Mixed Methods	Mixed-methods design integrates quantitative telemetry data (MSE, MAE, R^2, FPS profiling) with qualitative survey-based player engagement responses to provide a comprehensive evaluation of the AURA framework.
Strategy	Experimental / Simulation-Based	Controlled experiments are conducted within an Unreal Engine 5.7 prototype comparing adaptive and non-adaptive configurations to establish causal relationships between the framework and gameplay outcomes.
Time Horizon	Cross-Sectional with Iterative	Data collection occurs at fixed points during controlled gameplay sessions rather than continuously over an extended longitudinal period.

	Prototyping	
Data Collection	Primary (Telemetry + Surveys), Secondary (Literature + Docs)	Gameplay telemetry is captured directly from the prototype. Player feedback is gathered via structured surveys. Secondary sources provide theoretical and technical grounding.

Table 6: Research Methodology

Ethical Considerations: All participant data was anonymized at collection. Informed consent was obtained, and the study complied with GDPR and institutional ethics guidelines.

Hypothesis: A calibration-first, telemetry-driven adaptive framework integrating K-Means soft membership clustering, a canonical ANFIS-inspired adaptation surface whose reasoning is approximated by an MLP surrogate inference service can deliver stable, interpretable, and bounded real-time environment adaptation within a single-player interactive game environment, while mitigating cold-start bias, preserving player agency, and maintaining backend round-trip latency below 200 ms on free-tier cloud infrastructure.

A pragmatist stance was adopted to integrate quantitative model metrics (R^2, MAE, silhouette score) with qualitative feedback. A deductive, mixed-methods approach enables validation of hypotheses derived from flow theory and clustering literature using telemetry data and user evaluation. An experimental simulation-based strategy supports controlled assessment, while a cross-sectional design aligns with project constraints.

3.3. Development methodology

This section outlines the development process of the AURA system, including requirements, design, programming paradigm, testing, and solution methodology.

3.3.1. Requirement elicitation methodology

Requirements were gathered through surveys, interviews, observations, and cognitive walkthroughs.

3.3.2. Design methodology

Object-Oriented Analysis and Design Methodology (OOADM) as selected due to its alignment with Unreal Engine's object-based architecture and its support for modular, scalable system design. It enables key components such as telemetry collection, behavioural modelling, surrogate inference, and PCG control to be structured as independent, testable modules.

3.3.3. Programming paradigm

Object-oriented programming (OOP) was adopted as the primary paradigm across all AURA pipeline components. OOP enables encapsulation and modularity, allowing the behavioral modelling layer (Python), the backend inference service (TypeScript/Node.js), and the game engine interface (Unreal Engine 5.7 Blueprint) to function as independently maintainable units.

3.3.4. Testing methodology

Testing occurred at multiple levels to ensure functionality and stability.

Testing Level	Purpose	Tool / Technique
Unit Testing	Validate individual Python pipeline modules and UE5 game modules in isolation.	Vitest (Node.js services)
Integration Testing	Ensuring correct data flow and communication between the K-Means clustering, MLP surrogate, Next.js backend, and UE5 systems.	Simulated telemetry sessions
System Testing	Evaluate the complete adaptive pipeline end-to-end under controlled gameplay conditions.	Automated test maps
User Testing	Assess player experience via Game Experience Questionnaire (GEQ) and Self-Assessment Manikin (SAM).	Controlled pilot study

Table 7: Testing methodology

3.3.5. Solution methodology

The AURA framework follows a structured pipeline that transforms player behavioral data into

real-time environment adaptations. The methodology consists of several stages that ensure reliable behavioral modelling and efficient deployment.

1. **Calibration Data Collection:** Baseline telemetry was collected with adaptation disabled to capture unbiased player behaviour.
2. **Data Preprocessing:** Invalid sessions were removed, and data was filtered and normalized for consistent modelling.
3. **Calibration Integrity Validation:** Dataset quality was verified through coverage, consistency, and missingness checks before modelling.
4. **Behavioural Profiling:** K-Means clustering with inverse-distance soft membership was used to represent player archetypes.
5. **ANFIS-Inspired Adaptation Surface Design:** A canonical ANFIS-inspired formulation combines soft memberships, temporal deltas, and bounded outputs to define interpretable adaptation behaviour.
6. **Surrogate Model Creation:** A neural surrogate is trained on labelled data from the adaptation surface to enable efficient real-time inference.
7. **Model Validation:** The surrogate model was evaluated using regression metrics such as MSE, MAE, and R^2 to ensure accurate prediction of PCG parameter adjustments.
8. **Real-Time Environment Adaptation:** The backend applies bounded parameter updates based on incoming telemetry during gameplay.

3.4. Project management methodology

Kanban was selected for its simplicity, visual workflow, and flexibility, making it suitable for a single-developer project with evolving priorities. It supports continuous task tracking across data collection, modelling, backend development, and integration, enabling iterative refinement and adaptation to emerging technical challenges.

3.4.1. Project Scope

The project scope is as defined in *Chapter 01*, *Section 1.12.1* and *Section 1.12.2*, covering the full three-phase AURA pipeline, calibration evaluation, behavioural modelling, and backend-mediated inference. No revisions to the in-scope boundary were required during the development phase.

3.4.2. Schedule

3.4.2.1. Gantt Chart

Refer *Appendix C* for the Gantt chart.

3.4.2.2. Deliverables

- **Initial Project Idea Selection**: 20th August 2025
- **Initial Project Proposal Submission**: 10th October 2025
- **Project Proposal & Requirements Specification Submission**: 06th November 2025
- **Interim Progress Demonstration**: 30th January 2025
- **Final Project Report, Code and Demo**: 30th March 2025
- **Final Project Viva**: April 2025 to May 2025

Figure 5: Deliverables (Self Composed)

3.4.3. Resource requirements

3.4.4. Data Resources

Dataset / Source	Description	Type
Gameplay Telemetry Logs	Data captured every 30 seconds	Primary
Google Forms Survey	Player feedback on adaptive gameplay	Primary
Literature & Documentation	Technical specifications and prior studies	Secondary

Table 8: Data Resources

3.4.5. Skills Resources

Skill Area	Description
Machine Learning (Python)	Model development for the clustering and surrogate pipeline
Unreal Engine Development	Blueprint & C++ integration
Fuzzy Logic Design	ANFIS rule and membership function construction
Statistical Analysis	Telemetry evaluation and survey interpretation
Technical Writing	Thesis composition and publication

Table 9: Skills Resources

3.4.6. Risk and mitigation strategy

Key project risks and mitigation strategies are summarised below. In below table **E1 = Severity (1–5)**, **E2 = Frequency (1–5)**

Risk	**E1**	**E2**	**Mitigation Plan**
Limited training data diversity impacting model generalization	5	3	Simulate additional player sessions, increase sample size through augmentation, employ cross-validation to detect overfitting early.
Unreal Engine integration failure disrupting the telemetry-backend pipeline	4	2	Test incremental builds, sandbox VaRest plugin configurations, maintain version-controlled backups after each successful integration milestone.
MLP surrogate model overfitting or producing unstable multiplier output	5	4	Apply regularization, use early stopping during training, validate surrogate output against ANFIS ground-truth on held-out session data.
Hardware or critical data loss	5	2	Maintain cloud backups (GitHub and Google Drive) and enforce daily version control commits.
Participant dropouts during user testing	3	3	Recruit 30% additional participants at the outset, maintain flexible session scheduling.
Privacy or ethics breach involving participant telemetry data	5	1	Anonymize all data at the point of collection obtain explicit informed consent, adhere to GDPR and institutional ethics guidelines.

Table 10: Risk and Mitigation Strategy

3.5. Chapter summary

This chapter outlines the methodological framework of AURA, including research design, development process, and project management. A pragmatist, mixed-methods approach was applied within a structured pipeline covering calibration, modelling, validation, and deployment.

CHAPTER 04: SOFTWARE REQUIREMENT SPECIFICATION

4.1. Chapter overview

This chapter presents the Software Requirements Specification (SRS) for AURA, including a system overview, stakeholder analysis, requirement elicitation methods, and derived functional and non-functional requirements. It concludes with system context, use case diagrams, and structured requirement tables using MoSCoW prioritisation.

4.2. Rich picture Diagram

The rich picture diagram presented in Figure 6: Rich Picture Diagram (Self Composed) provides an informal, holistic representation of the AURA system environment, illustrating the key actors, processes, and interactions that shape the requirement.

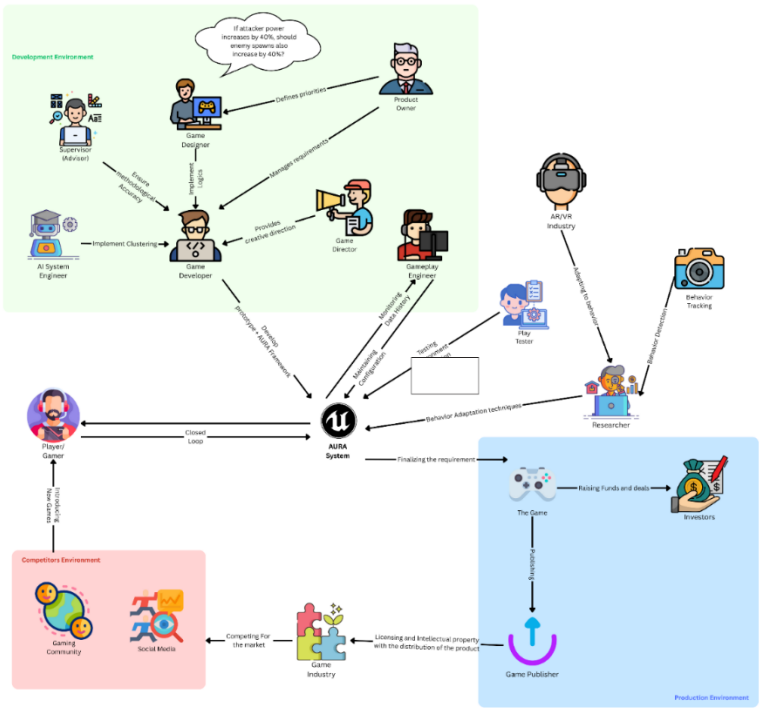


Figure 6: Rich Picture Diagram (Self Composed)

The diagram identifies the player as the primary actor whose gameplay behaviour drives the adaptive pipeline, with telemetry flowing through the backend inference service before returning as bounded PCG parameter updates to the Unreal Engine environment. Key concerns depicted

include latency constraints on the backend communication channel, calibration phase isolation to prevent contamination of early behavioural data, and the developer's role in monitoring and configuring adaptive parameters. The rich picture also highlights the broader research context, including the involvement of the academic supervisor and the ethical obligations around player data anonymisation and consent.

4.3. Stakeholder Analysis

A stakeholder analysis was conducted to identify all individuals and groups with an interest in, or influence over, the design and deployment of the AURA framework. Understanding the diverse needs and perspectives of these stakeholders was essential to deriving a comprehensive and accurate set of system requirements.

4.3.1. Stakeholder Onion Model

The Onion Model represents organises stakeholders according to their proximity to the system's core operation and their relative degree of influence over system outcomes, from direct operational users at the centre to external community stakeholders at the periphery.

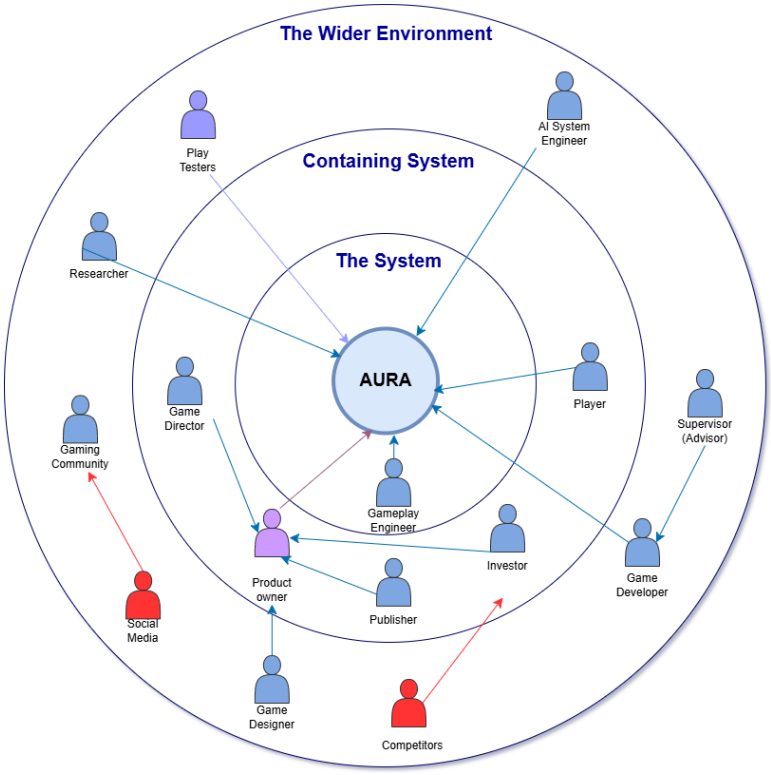


Figure 7: Onion Model (Self Composed)

The model confirms that the player occupies the innermost operational layer as the primary source of behavioural telemetry, whilst the AI/ML engineer and gameplay engineer reside in the immediate development layer responsible for system construction and training. Supervisory and managerial stakeholders occupy outer advisory and governance layers.

4.3.2. Stakeholder Viewpoints

The following table summarises the role and interests of each identified stakeholder group with respect to the AURA system.

Stakeholder	Role	Benefits / Role Description
Player	End-user	The direct user of the AURA system. Gameplay behaviour activates adaptive responses including dynamic environment scaling, difficulty adjustment, and procedural item spawning. The player's telemetry constitutes the primary data source driving the adaptive pipeline.
Game and AI Engineer	Developer	Designs and integrates adaptive gameplay logic, AI modules, and engine-side Blueprint components. Responsible for telemetry capture, VaRest HTTP integration, and ensuring that adaptive features operate within Unreal Engine performance constraints.
Researcher	Analyst / Contributor	Gathers behavioural and performance data from evaluation sessions, interprets model outputs, and provides insights that guide system refinement and adaptive parameter calibration across all three development phases.
Supervisor	Advisor / Mentor	Oversee the research and technical development of the AURA project, ensuring methodological rigour, academic quality, and alignment with institutional submission requirements.
Play Tester	Tester	Engages with development builds to assess how adaptive elements respond during gameplay. Provides qualitative and quantitative feedback on perceived fairness, challenge continuity, and

		engagement quality.
Product Owner	Manager / Coordinator	Represents project goals and priorities, balancing stakeholder expectations and ensuring that adaptive system features meet both design intent and measurable performance targets.
Game Designer	Designer	Defines gameplay flow and challenge balance. Collaborates with the engineering layer to ensure that adaptive outputs enhance player experience without disrupting the core design logic of the game environment.

Table 11: Stakeholder Viewpoints

4.4. Selection of Requirement Elicitation Methodologies

Requirement elicitation combined **multiple qualitative and quantitative techniques** to ensure a comprehensive understanding of user expectations, technical constraints, and system objectives.

1. **Literature Review** - Conducted to obtain theoretical background, identify research gaps related to adaptive PCG and neuro-fuzzy systems, and validate core architectural decisions including K-Means soft clustering, a canonical ANFIS-inspired adaptation surface whose reasoning is approximated by an MLP surrogate inference service.
2. **Survey (Google Forms)** - A questionnaire distributed to 55 participants aged 18–30 to collect opinions and expectations regarding adaptive gameplay, perceived fairness, and environmental modification.
3. **Expert Interviews** - Interviews were conducted with domain and technical experts to gather qualitative insights on existing adaptive systems and identify limitations relevant to the proposed framework. Expert feedback informed the behavioural modelling requirements of the AURA system.
4. **Prototyping** - An iterative prototyping process validated design assumptions and identified technical constraints across successive subsystem builds, including telemetry capture, backend integration, and fuzzy inference control.
5. **Observation** - Used to observe player interactions and system responses during gameplay sessions under static and adaptive configurations, validated using the Self-Assessment Manikin (SAM) scale.

Each method addressed different information needs and was triangulated to form a reliable requirement set.

4.5. Discussion of Findings

This section interprets the results of each elicitation method and derives requirement insights relevant to the Neuro-Fuzzy PCG system.

4.5.1. Findings from Literature Review

Finding	Citation
Procedural Content Generation (PCG) methods have evolved from static, rule-based systems to dynamic frameworks that attempt to integrate adaptivity and personalization, yet most lack real-time responsiveness and player-centric feedback integration.	(Hendrikx et al., 2013)
Data-driven PCG approaches have introduced creative diversity through learning-based generation but continue to suffer from low interpretability, making them unsuitable for transparent adaptive control in live gameplay.	(Summerville et al., 2018)
Emotion-aware and affective adaptation systems improved engagement but failed to integrate with procedural generation pipelines, revealing the need for unified behaviour-based adaptation.	(Bontchev, 2016)
Reinforcement learning-based PCG demonstrated potential for real-time content control. However, its high computational cost and lack of explainability make it impractical for scalable adaptive design.	(Gisslen et al., 2021)
Deep Neuro-Fuzzy Inference Systems proved effective in adaptive environmental control but required manual tuning and retraining, emphasizing the necessity for self-adjusting fuzzy reasoning models.	(Talpur et al., 2022)
Neuro-symbolic frameworks combining logical reasoning with neural learning have increased interpretability but remain limited by synchronization and real-time latency issues.	(Verheyen et al., 2023)

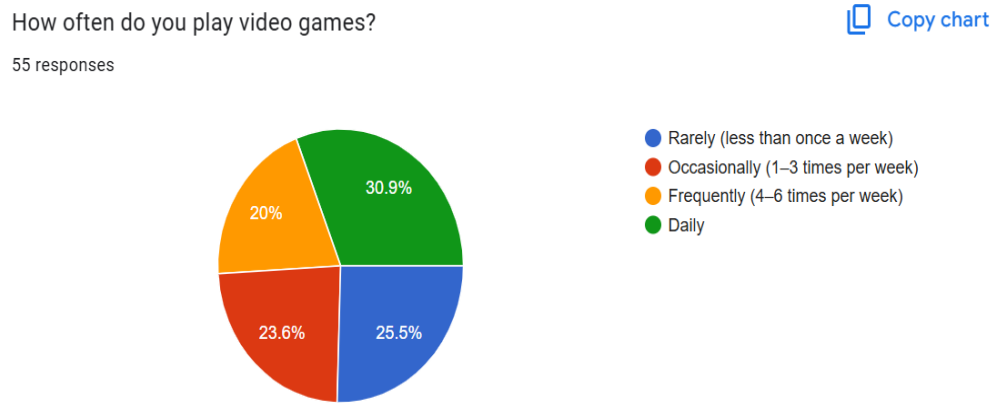
Contemporary evaluations of adaptive PCG highlight the absence of standardized benchmarks and consistent testing criteria across game engines, creating challenges in comparing model performance and responsiveness.	(Kalafatis et al., 2025)
The literature consistently identifies a core research gap, achieving a balance between adaptivity, explainability, and computational efficiency which directly informs the rationale for the proposed Neuro-Fuzzy adaptive PCG framework.	(Summerville et al., 2018; Talpur et al., 2022; Verheyen et al., 2023)

Table 12: Findings from Literature Review

4.5.2. Findings from Survey

An online survey of 55 participants aged 18–30 gathered feedback on adaptive gameplay, environment changes, and system fairness. The questions explored perceptions of adaptive AI and environmental adjustments in games to inform the Neuro-Fuzzy PCG framework's development. Results for Q1 are shown below, with the remaining questions in *Appendix D*.

Question 1	How often do you play video games?
Aim of the Question	To identify the gaming frequency and determine the reliability of participants as active players.
Findings and Conclusion	The majority reported playing either daily or a few times per week, confirming that most participants are active gamers who can provide relevant insights on adaptive mechanics.



4.5.3. Findings from Expert Interviews

Interviews were conducted with domain and technical experts to gather qualitative insights on existing adaptive systems and validate the proposed behavioural modelling approach. The findings were analysed using thematic analysis to identify key themes. Interview transcripts and supporting evidence are available in *Appendix E*.

4.5.4. Findings from Prototyping

The prototyping process followed an iterative design approach to establish a functioning adaptive framework integrating K-Means clustering and ANFIS within Unreal Engine. Each prototype iteration incrementally validated one subsystem of the pipeline, including player telemetry capture, behavioural window accumulation, and HTTP dispatch to the external inference service. During initial iterations, one of the main challenges was establishing reliable communication between Unreal Engine and the backend service, which was resolved by optimising VaRest HTTP request handling and implementing buffered telemetry payloads. Network latency and inconsistent sampling intervals initially disrupted synchronization, which was mitigated by optimizing socket communication and implementing buffered data transmission. Another significant issue was tuning ANFIS membership functions to respond consistently to variable player behaviors. Through successive refinement cycles and simulated test sessions, the prototype achieved stable real-time adaptation with bounded parameter updates and consistent environmental responsiveness.

4.5.5. Findings from Observation

To evaluate the responsiveness and experiential quality of the developed prototype, controlled observation sessions were conducted using player-based simulations. Each session monitored behavioral metrics such as reaction time, exploration rate, and engagement duration, alongside adaptive system responses. Participants were observed during controlled gameplay sessions to identify reactions to parameter change frequency, perceived fairness, and response continuity to measure differences in perceived fairness, engagement continuity, and flow stability. Observations identified that players responded negatively to abrupt parameter changes during gameplay, informing the requirement for smooth, forward-only procedural modulation (FR07, FR08). The need for a stability threshold distinguishing minor fluctuations from significant playstyle shifts

was also derived from observed player responses to oscillatory adaptation.

4.6. Summary of Findings

Each finding has been mapped against the method through which it was identified or validated during the research process. (E1: Literature Review, E2: Survey, E3: Expert Interviews, E4: Prototyping, E5: Observation)

ID	Finding	E1	E2	E3	E4	E5
1	APCG systems require integration of machine learning and fuzzy reasoning to achieve both real-time performance and interpretability.	✓		✓	✓	
2	Player engagement depends heavily on smooth, continuous difficulty balancing rather than static difficulty levels.		✓	✓		✓
3	Behavioral clustering through K-Means provides an effective mechanism for categorizing player types for adaptive reasoning.	✓			✓	
4	Real-time adaptation must remain subtle and non-intrusive to preserve immersion and fairness in gameplay.		✓	✓		✓
5	The communication pipeline between Unreal Engine and the adaptive module must be optimized to avoid latency in data exchange.			✓	✓	
6	ANFIS-based fuzzy rules should dynamically evolve according to clustered behaviour to maintain long-term engagement balance.	✓			✓	
7	Adaptive environment elements, such as enemy density and item distribution, significantly enhance player satisfaction when controlled smoothly.		✓	✓	✓	✓
8	Observational results confirmed that the Neuro-Fuzzy system produced more stable and fair adaptation responses compared to static configurations.					✓
9	The adaptive controller must be modular and engine-agnostic to ensure scalability across different game contexts.	✓			✓	

10	Feedback from player observations validated that transparent adaptation logic improved trust and engagement levels.		✓			✓

Table 13: Summary of Findings

4.7. Context Diagram

The Context Diagram presents an overall view of the Neuro-Fuzzy PCG system boundary, its external entities, and the data flowing among them.

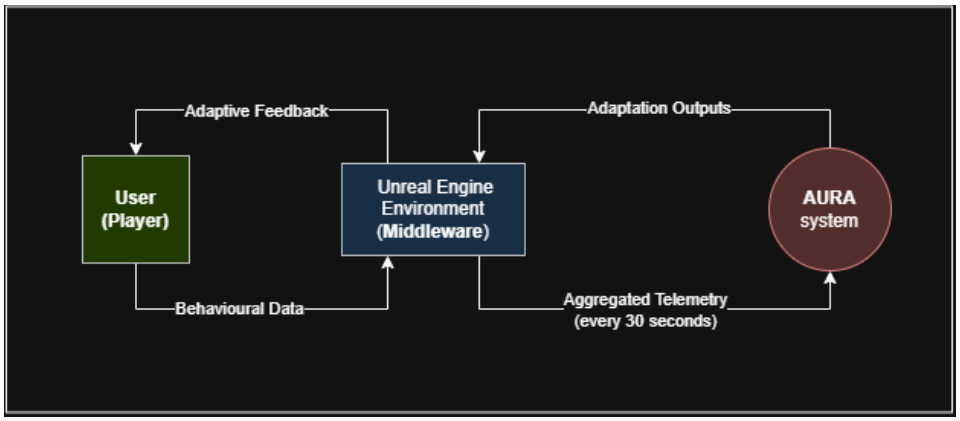


Figure 8: System Context Diagram (Self Composed)

As shown in Figure 8: System Context Diagram (Self Composed), the system boundary separates the Unreal Engine game client and player interface from the external **Node.js backend inference service** and the Next.js analytics dashboard, with telemetry payloads flowing outward and **bounded PCG multiplier values** returning inward through **HTTP endpoints**.

4.8. Use Case Diagram

The Use Case Diagram depicts all user and system interactions within the adaptive gameplay environment.

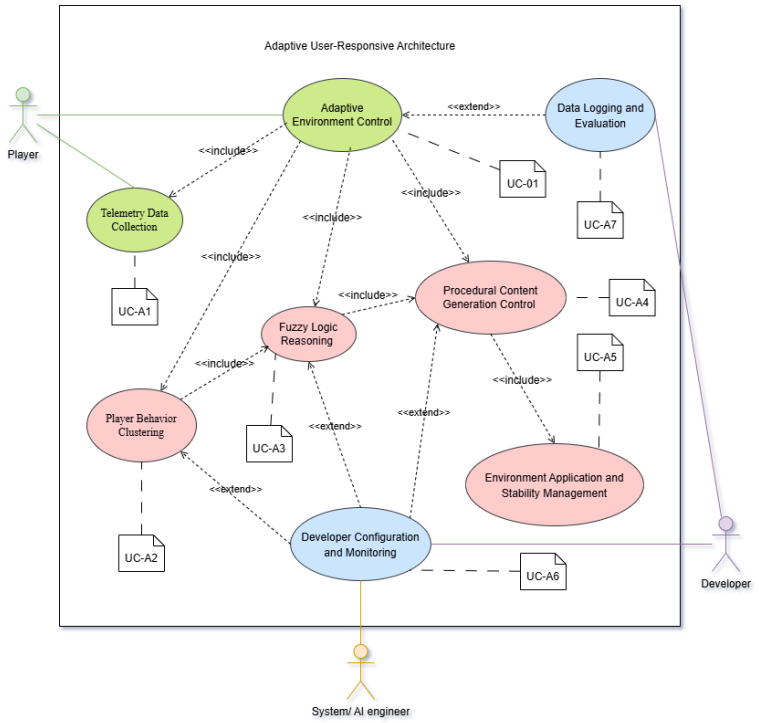


Figure 9: Use case Diagram (Self Composed)

As shown in Figure 9: Use case Diagram, the Player is the primary actor initiating the adaptive loop through gameplay actions, while the Developer actor interacts with the system through configuration and monitoring sub-cases. The diagram confirms seven distinct use cases (UC-A1 through UC-A7) that collectively implement the **closed-loop adaptive pipeline** described in the chapter overview.

4.9. Use Case Descriptions

Use case description for the use case UC-01 is given below. The other use case descriptions can be found in *Appendix F*.

Use Case ID	**UC-01**
Use Case Name	Adaptive Environment Control
Primary Actors	Player
Supporting Actors	System (AI Engine), Developer

Description	Top-level use case describing the full closed-loop adaptation: local telemetry capture, K-Means clustering, ANFIS fuzzy inference, mapping of fuzzy outputs to PCG configuration, and application of environment updates in Unreal Engine. Developers may monitor and adjust configurations.
Preconditions	Game running telemetry, clustering and backend inference service initialized developer UI available if in debug mode.
Postconditions	Environment parameters adjusted (enemy density, item spawn, stamina) according to inferred player behavior while maintaining stability and acceptable latency. Session logs generated.
Normal Flow	UC-A1: Telemetry Data Collection captures gameplay metrics and request adapted parameters through API endpoints. UC-A2: Player Behavior Clustering identifies behavioral type. UC-A3: MLP Surrogate Inference produces a bounded adaptation multiplier via the backend inference service. UC-A4: Receive response for API request and then PCG Control applies those values to the environment. UC-A5: Stability Management ensures smooth transitions. UC-A6 and UC-A7 handle developer configuration and session logging
Alternative Flow	If any module fails (telemetry, clustering, or ANFIS), the system reverts to the last stable configuration and continues gameplay seamlessly.

Table 14: UC-01 Use Case Descriptions

4.10. Requirements

The functional and non-functional requirements of the AURA system were derived from the research objectives and the synthesised findings of the elicitation phase. The requirements are expressed at the system interface level, describing what the system must do and the quality constraints it must satisfy, without prescribing implementation mechanisms. **MoSCoW prioritisation (Must Have, Should Have, Could Have, Won't Have)** is applied to each requirement to reflect its relative importance within the scope of this project and to guide **phased implementation decisions**.

4.10.1. Functional Requirements

FR ID	Functional Requirement Description	Priority Level	Linked Use Case(s)
FR01	Capture player telemetry at fixed intervals.	M	UC-A1, UC-01
FR02	Preprocess and normalise telemetry data.	M	UC-A1, UC-A2, UC-01
FR03	Classify behaviour into archetypes using soft clustering.	M	UC-A2, UC-01
FR04	Retain last valid classification on incomplete data.	M	UC-A2, UC-01
FR05	Generate adaptation values via backend inference.	M	UC-A3, UC-01
FR06	Provide visualisation for inference outputs.	S	UC-A3, UC-A6, UC-01
FR07	Update PCG configuration variable values based on backend adaptation outputs.	M	UC-A4, UC-01
FR08	Apply updates within latency constraints for smooth gameplay.	M	UC-A4, UC-A5, UC-01
FR09	Monitor performance and revert on threshold violations.	S	UC-A5, UC-01
FR10	Provide configurable developer interface.	S	UC-A6, UC-01
FR11	Record telemetry and adaptation data for reporting.	M	UC-A7, UC-01
FR12	Support import/export of configuration templates.	C	UC-A6, UC-01

Table 15: Functional Requirements

4.10.2. Non-Functional Requirements

NFR	Requirement	Description	Priority
NFR1	**Performance**	Real-time operation with stable FPS and latency ≤ 200 ms.	**M**
NFR2	**Quality of Adaptation**	Adaptations must maintain gameplay balance and contextual relevance.	**M**
NFR3	**Efficiency**	Backend inference ≤ 4 ms and total latency ≤ 200 ms.	**S**

NFR4	**Usability**	Interface must be clear, intuitive, and support error recovery.	**S**
NFR5	**Scalability**	Supports new behaviours and parameters without architectural changes.	**S**
NFR6	**Reliability**	Ensures stable operation with fallback on failure.	**M**
NFR7	**Maintainability**	Modular design enabling easy maintenance and debugging	**S**
NFR8	**Security**	Uses anonymised telemetry and no PII exposure or direct database access.	**S**

Table 16: Non-Functional Requirements

4.11. Chapter Summary

This chapter presented the Software Requirements Specification for the AURA framework, including stakeholder analysis, requirement elicitation, and the definition of functional and non-functional requirements. The system requirements were structured and prioritised, with key use cases identified. The next chapter addresses the social, legal, ethical, and professional considerations of the system.

CHAPTER 05: SOCIAL, LEGAL, ETHICAL AND PROFESSIONAL ISSUES (SLEP)

5.1. Chapter Overview

This chapter outlines the social, legal, ethical, and professional considerations of the AURA framework, focusing on responsible telemetry use and fair real-time adaptation.

5.2. Breakdown of SLEP Issues

Below table summarises the identified issues and mitigation strategies.

SOCIAL	LEGAL
Only gameplay behaviour is collected, with no personal data. Adaptation is bounded (±40%) to ensure fairness. Users provide consent and can withdraw.	All tools comply with relevant licences. Data handling follows UK GDPR, ensuring anonymisation and no storage of identifiable data.
ETHICAL	**PROFESSIONAL**
The study is transparent, non-deceptive, and uses anonymised data. Participant rights are maintained throughout.	The project follows the BCS Code of Conduct, ensuring integrity, accurate reporting, and proper version-controlled development.

Table 17: Breakdown of Social, Legal, Ethical, and Professional Issues

5.3. Chapter Summary

SLEP considerations were addressed through compliant data handling, ethical design, and adherence to professional standards with the alignment with the BCS Code of Conduct.

CHAPTER 06: DESIGN

6.1. Chapter Overview

This chapter presents the system design of the AURA framework, defining its architecture, components, and adaptive pipeline. Design goals and key diagrams are introduced, alongside core algorithms governing system behaviour, with supporting interface wireframes provided in the appendix.

6.2. Design Goals

The desired design goals to be achieved by the system are specified in the table below.

Design Goal	Description
Performance	Inference logic is decoupled from the game client and hosted on an external service, reducing in-engine computational load. Adaptation outputs are constrained within a fixed multiplier range, preventing extreme procedural modulation.
Stability	Adaptation outputs are smoothed over a configurable time window. A stability threshold filters minor fluctuations, ensuring only significant playstyle shifts trigger parameter updates.
Accuracy	Behavioural classification uses soft continuous membership across three archetypes informed by temporal momentum, rather than hard discrete assignment, improving representational fidelity.
Modularity	The adaptive pipeline separates calibration, telemetry processing, behavioural modelling, and surrogate inference into independent components with well-defined interfaces.
Scalability	The inference pipeline is game-engine-agnostic, communicates via standard HTTP REST, and is parameterisable across different procedural content domains.

Table 18: AURA Design Goals

6.3. System Architecture

The AURA framework follows a distributed three-tier architecture, separating the game presentation environment, the backend inference logic, and the persistent model artefacts across distinct tiers with well-defined communication boundaries.

6.3.1. Architectural Diagram

Figure 10: Distributed three-tier runtime architecture of the proposed AURA framework (Self Composed). illustrates the distributed three-tier runtime architecture of the AURA framework.

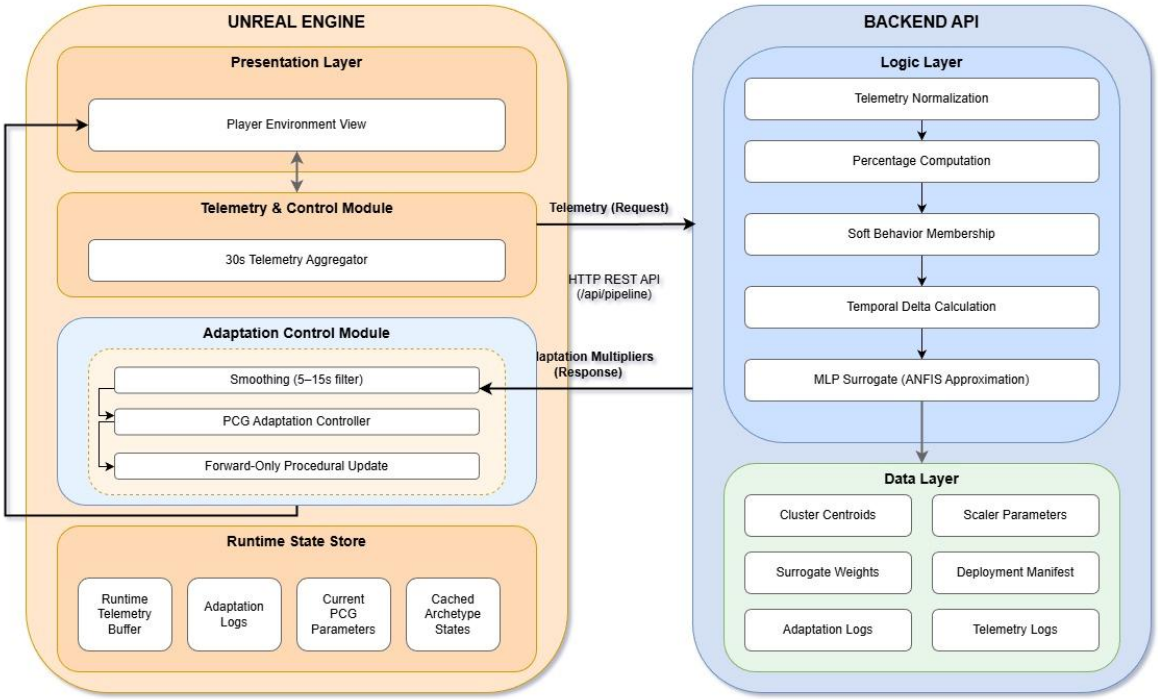


Figure 10: Distributed three-tier runtime architecture of the proposed AURA framework (Self Composed).

6.3.2. Discussion of tiers of the architecture

Presentation tier (Game Client/ Dashboard)

- **Telemetry accumulator:** Collects gameplay signals into 30-second feature windows.
- **HTTP dispatch agent:** Sends telemetry to the backend and receives adaptation results.
- **Adaptation applicator:** Applies player updates immediately and defers PCG changes.
- **Degraded-mode controller:** Uses neutral defaults when the backend is unavailable.

Logic tier (Backend Inference Service)

- **Normalisation component:** Scales telemetry using calibration parameters.
- **Activity scoring component:** Computes behavioural intensities.
- **Soft membership component:** Derives archetype weights and temporal changes.
- **Surrogate inference component:** Generates the adaptation multiplier.
- **Calibration component:** Bounds and finalises the multiplier output.

Data tier (Persistent Artefacts)

- **Calibration artefacts:** Baselines, scaling parameters, and cluster centroids.
- **Model artefacts:** Trained weights and reference values.
- **Telemetry logs:** Recorded feature windows, memberships, and multipliers.

Overall Architecture Summary

The AURA architecture operates as a three-tier system where the client collects behaviour, the backend generates adaptive responses, and the data tier maintains required artefacts. This separation ensures modularity, scalability, and real-time adaptation.

6.4. Detailed Design

6.4.1. Selection of the Design Paradigm

The AURA backend adopts the Object-Oriented Analysis and Design Methodology (OOADM), consistent with *Chapter 03*. This approach supports modular decomposition into components with defined responsibilities, enabling independent testing and facilitating future extension without requiring architectural changes.

6.4.2. Component Diagram

Figure 11: AURA System - Component Diagram (Self Composed) presents the component diagram of the AURA system, illustrating the three deployed execution environments and their communication interfaces. The game client, backend inference service, and dashboard inference

pipeline communicate via standard HTTP, with unidirectional telemetry flow from client to backend and adaptation results returned to the client.

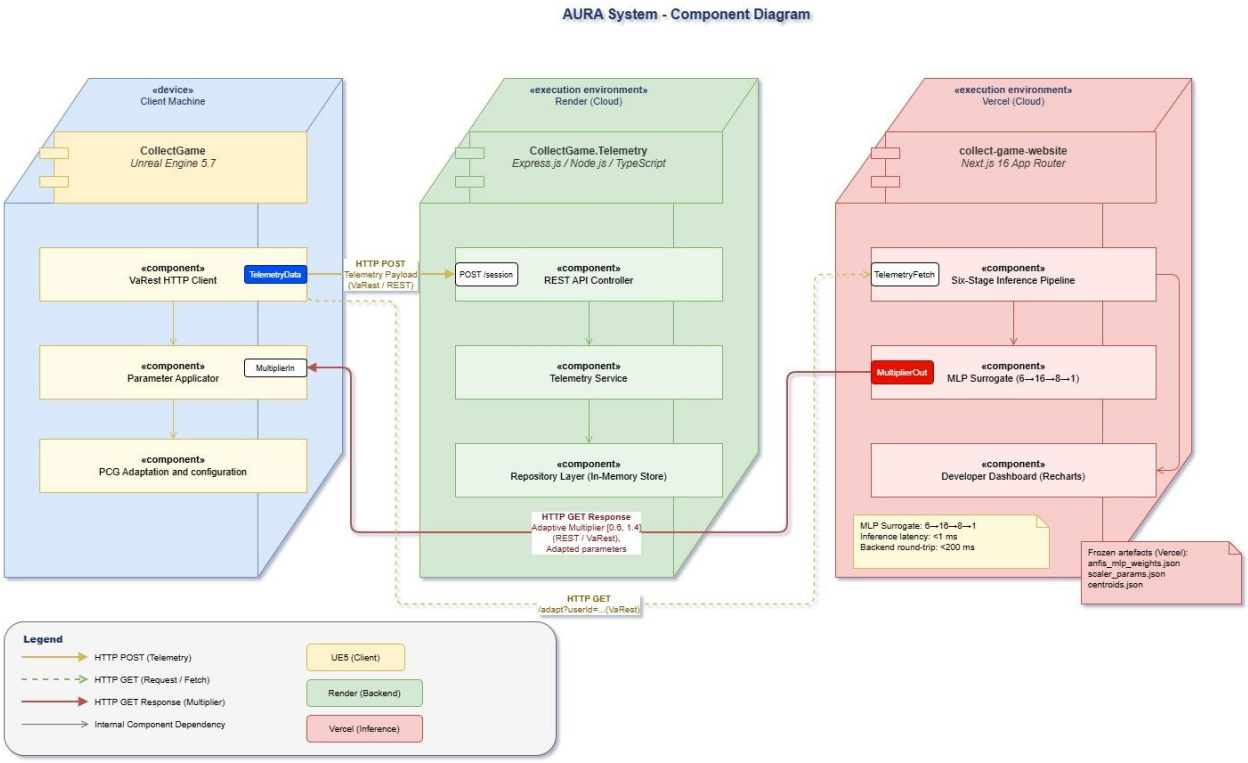


Figure 11: AURA System - Component Diagram (Self Composed)

6.4.3. Class-Level Design (Backend Core)

The backend inference pipeline is composed of five principal components, each with single-responsibility boundaries. The TelemetryNormaliser scales incoming telemetry features against the Mode 1 calibration baselines. The ActivityScorer computes per-archetype activity intensities from the normalised input. The SoftMembershipEngine applies inverse-distance weighting over the K-Means cluster centroids to produce continuous archetype proportions. The SurrogateInferenceModel executes the trained MLP forward pass to generate the raw multiplier output. The MultiplierCalibrator applies neutral-centred offset adjustment and bounds. The full class diagram with attribute definitions is presented in *Appendix G.1*.

6.4.4. Runtime Sequence Diagram

Figure 12: Runtime interaction sequence of the AURA adaptive control loop (Self Composed) *i*llustrates the runtime interaction between the Unreal Engine client and the backend inference

service within a telemetry–inference–adaptation cycle. The sequence reflects the request–response communication and forward-only adaptation applied during gameplay.

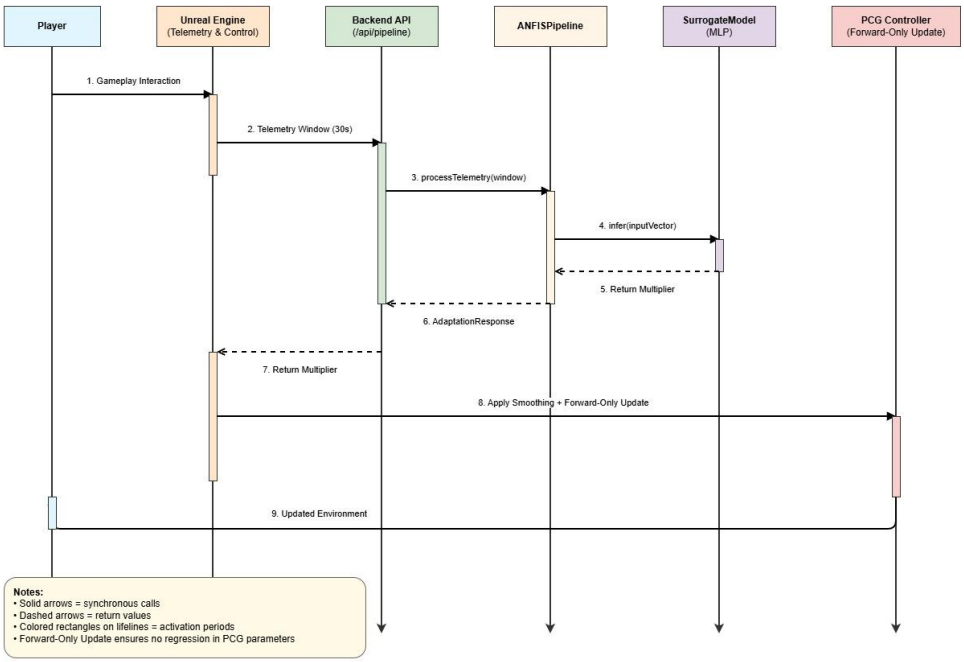


Figure 12: Runtime interaction sequence of the AURA adaptive control loop (Self Composed)

6.4.5. Backend Inference Activity Flow

Figure 13: Backend Inference Activity Flow (Self Composed) presents the backend inference pipeline executed for each telemetry window. The process sequentially transforms input data into an adaptation multiplier, with calibration and bounding ensuring the output remains within defined limits. The full picture is in *Appendix G.4*.

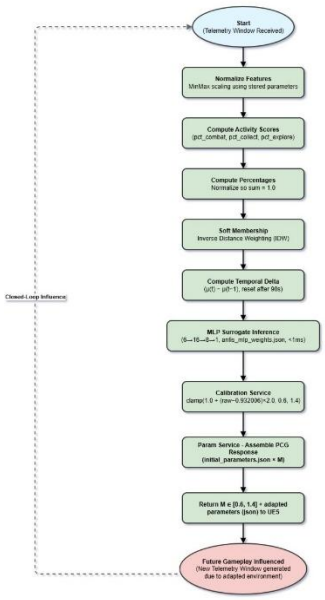


Figure 13: Backend Inference Activity Flow (Self Composed)

6.4.6. Unreal Adaptation Control Flow

The Unreal-side adaptation control activity flow, illustrating multiplier application, temporal smoothing, stability threshold evaluation, and degraded-mode fallback within the game client, is presented in *Appendix G.2*.

6.4.7. System Process Activity Diagram

The end-to-end adaptive control loop, from session initialisation through health-check handshake to per window telemetry inference adaptation cycles, is presented in *Appendix G.3* as *Figure 32*: AURA - End-to-End System Process Activity Diagram (Adaptive Control Loop) (Self Composed). The diagram illustrates the degraded-mode fallback behaviour when the backend is unavailable, and the repeating 30-second interval structure that governs all adaptive execution

6.5. Algorithm Design

This section formally defines the four computational algorithms underpinning the AURA adaptive pipeline, each presented with its functional purpose, mathematical formulation, and structured pseudocode.

6.5.1. Calibration Baseline Selection Algorithm

The calibration algorithm selects the game mode configuration that produces the most statistically neutral behavioural baseline by minimising a composite NeutralityScore across three weighted criteria:

$$NeutralityScore = 0.3 \times z(MeanSparsity) + 0.2 \times z(MeanStdDev) + 0.5 \times z(DeathRate)$$

MeanSparsity(m)	mean proportion of zero-valued telemetry features across calibration sessions for mode m, measuring behavioural expressiveness
DeathRate(m)	proportion of sessions in which the participant exceeded the death threshold for mode m, measuring frustration pressure
MeanStdDev(m)	mean per-feature standard deviation across calibration sessions for mode m, measuring behavioural volatility
w_1, w_2, w_3	Z-score normalised weighting coefficients applied post-normalisation: w_1 (sparsity) = 0.3, w_2 (std dev) = 0.2, w_3 (death rate) = 0.5. Each criterion is z-score normalised across modes before aggregation. The death-rate criterion receives the dominant weight (0.5) because a mode with materially higher lethality cannot serve as a behaviourally neutral baseline regardless of sparsity differences. Robustness analysis confirming Mode 1 selection is presented in *Section 8.5.2* and *Appendix N*

The mode yielding the minimum NeutralityScore is retained as the permanent calibration baseline for all subsequent modelling phases, ensuring that adaptive inference is anchored to uncontaminated behavioural data.

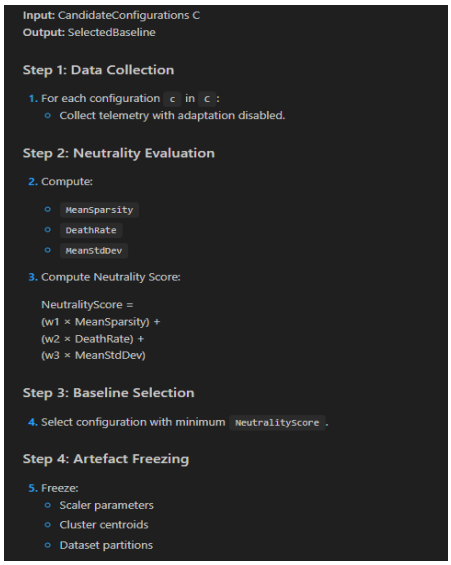


Figure 14: Calibration Baseline Selection Algorithm (Self Composed)

6.5.2. Behavioural Modelling Algorithm

The AURA behavioural archetypes of Combat, Collection, and Exploration are grounded in the PAD emotional model (Mehrabian and Russell, 1974; Bran and Vaidis, 2020). Each archetype represents a distinct emotional profile. Combat behaviour aligns with high Arousal due to challenge and threat, and high Dominance through perceived control (Bontchev, 2016; Khoshnoud, Alvarez Igarzábal and Wittmann, 2020). Collection behaviour reflects high Pleasure driven by reward acquisition, with moderate Arousal (Bontchev, 2016). Exploration behaviour maintains high Pleasure through curiosity, with lower Dominance as the environment remains unconquered. This mapping is a theoretical contribution of AURA, as no prior work directly links these archetypes to PAD dimensions in game contexts.

This grounding informs the interpretation of behavioural data. The soft membership vector [μC, μO, μE] acts as a proxy for PAD states, representing the player's position across affective profiles using telemetry alone (Yu, 2024). The temporal delta vector [ΔC, ΔO, ΔE] captures changes over time, reflecting emotional trajectory rather than static state, consistent with affect-driven adaptation principles (Yannakakis and Togelius, 2011)

The behavioural modelling algorithm transforms a raw eleven-feature telemetry window into a six-dimensional behavioural state vector through four sequential operations.

(i) Feature normalisation

$$X' = \frac{X - min}{max - min}$$

X	Raw telemetry feature value for the current window
min, max	per-feature minimum and maximum values derived from the Phase 1 calibration dataset

(ii) Proportional activity scoring

$$P_k = \frac{S_k}{\sum S_j}$$

S_k	raw activity intensity score for archetype k ∈ {Combat, Collection, Exploration}, computed as the means of its designated telemetry feature group
ΣS_j	sum of all three archetype activity scores, normalising P_k to a unit-simplex

(iii) Inverse-distance weighting (soft membership)

$$d_k = | P - C_k |$$

$$\mu_k = \frac{1/(d_k + \epsilon)}{\sum_j 1/(d_j + \epsilon)}$$

d_k	Euclidean distance between the proportional activity vector P and the K-Means centroid C_k for archetype k
μ_k	continuous soft membership weight for archetype k and the partition-of-unity constraint $\Sigma \mu_k = 1$ is enforced
ε	numerical stabilisation term (10^{-6}) preventing division by zero when P coincides with a centroid

(iv) Temporal delta computation

$$\Delta_k(t) = \mu_k(t) - \mu_k(t-1)$$

$\Delta_k(t)$	window-to-window directional shift in soft membership weight for archetype k, capturing behavioural momentum

The resulting six-dimensional state vector [μC, μO, μE, ΔC, ΔO, ΔE] is passed directly to the surrogate inference component.

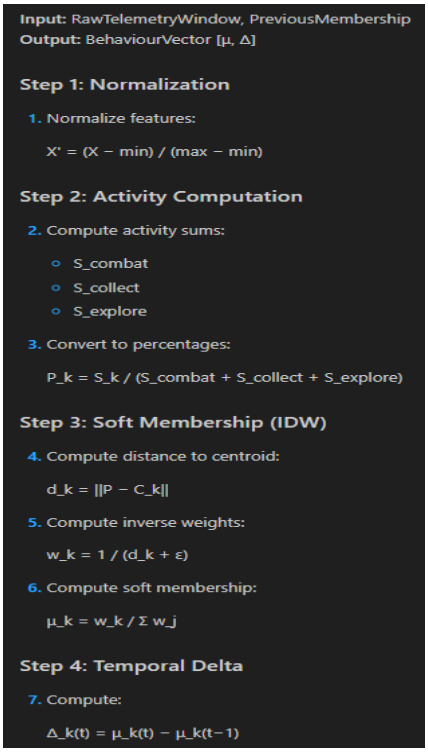


Figure 15: Behavioural Modelling Algorithm (Self Composed)

6.5.3. Surrogate Inference Algorithm

The surrogate inference algorithm approximates the ANFIS reasoning surface trained offline during Phase 2, mapping the six-dimensional behavioural state vector to a bounded scalar adaptation multiplier through a two-hidden-layer perceptron:

(i) Input vector

$$X = [\mu c, \mu l, \mu e, \Delta c, \Delta l, \Delta e]$$

(ii) Forward propagation

$$h_1 = \sigma(W_1 X + b_1)$$
$$h_2 = \sigma(W_2 h_1 + b_2)$$
$$M\_raw = W_3 h_2 + b_3$$

W_1, W_2, W_3	trained weight matrices for layers of dimensionality 6→16, 16→8, and 8→1 respectively
b_1, b_2, b_3	bias vectors for each layer

σ	ReLU activation function applied elementwise at each hidden layer. the output layer uses a linear activation

(iii) Neutral-centred calibration and output bounding

$$M\_final = clamp(1.0 + (M\_raw - M\_neutral) \times 2.0, M\_min, M\_max)$$

M_neutral	the MLP output produced for a behaviourally neutral input, computed from the Mode 1 calibration baseline (M_neutral = 0.932006)
M_min, M_max	adaptation bounds set to 0.6 and 1.4 respectively, constraining all procedural modulation within a ±40% range of the neutral state

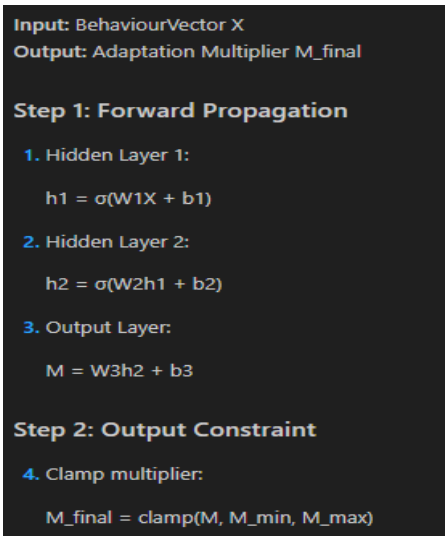


Figure 16: Surrogate Inference Algorithm (Self Composed)

6.5.4. Adaptation Control Algorithm

The adaptation control algorithm applies the bounded multiplier to procedural parameters while suppressing oscillatory responses and enforcing a forward-only modification constraint.

(i) Temporal smoothing

$$M\_smooth(t) = \alpha \cdot M\_new + (1 - \alpha) \cdot M\_prev$$

α	exponential smoothing coefficient set to 0.2, weighing recent inference output against the previous window's value to suppress transient noise
M_new	bounded multiplier output from the surrogate inference component for the current telemetry window

M_prev	smoothed multiplier value applied during the preceding window

(ii) Stability threshold

$$|\mathrm{M\_smooth(t)} - \mathrm{M\_prev}| < \theta$$

θ	minimum multiplier displacement required to trigger a parameter update. changes below this threshold are suppressed, preventing micro-oscillation around the current adaptive state

(iii) Procedural parameter scaling

$$V\_new = V\_base \cdot M\_smooth(t)$$

V_base	the Mode 1 calibration baseline value for the target procedural parameter (enemy density, stamina regeneration, item spawn rate, etc.)
V_new	the adapted parameter value applied at the next generation cycle

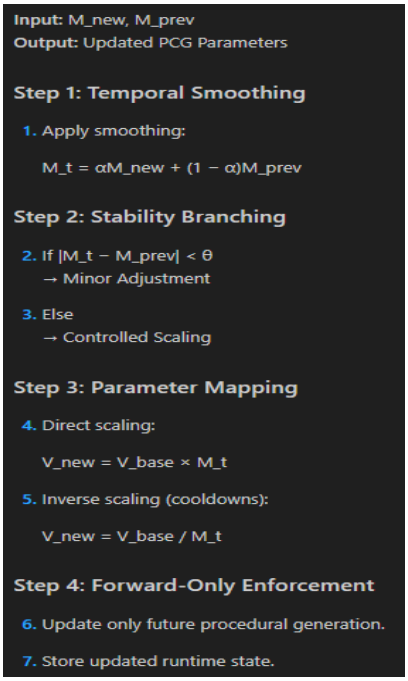


Figure 17: Unreal Adaptation Control Algorithm (Self Composed)

The forward-only constraint is enforced in the parameter application step: V_new affects only content generated after the current window boundary, ensuring that previously created environment elements remain unchanged across all adaptation cycles.

6.6. UI Design

The AURA Low-fidelity wireframes are provided in *Appendix H.1* and *Appendix H.2* respectively, with final implemented interfaces presented in *Chapter 07*.

6.7. Chapter Summary

This chapter presented the design of the AURA framework, including its three-tier architecture, component responsibilities, and closed-loop adaptive pipeline. Core algorithms were specified, and key design artefacts such as diagrams and UI wireframes were introduced, with additional details provided in the appendix.

CHAPTER 07: IMPLEMENTATION

7.1. Chapter Overview

This chapter presents the implementation of the AURA framework, translating the previously defined system design into a functional architecture. It operationalises the calibration-first methodology, telemetry pipeline, behavioural clustering, and neuro-fuzzy surrogate layer. The chapter covers technology selection, core module implementation, user interface integration, and key challenges, concluding with a summary demonstrating system readiness for evaluation.

7.2. Technology Selection

Technology selection was governed by three overarching requirements derived from the Software Requirement Specification: real-time responsiveness, research reproducibility, and modular separation of the calibration, training, and inference workloads.

7.2.1. Technology Stack

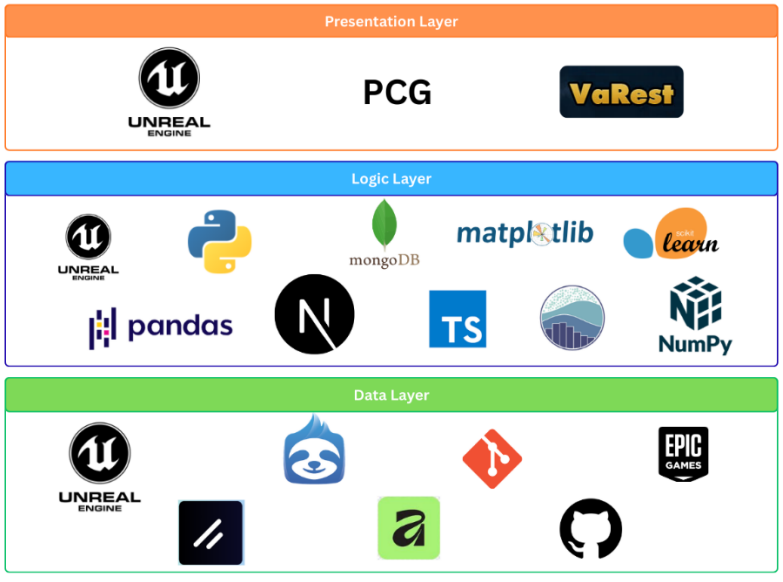


Figure 18: Technology Stack (Self Composed)

7.2.2. Programming Languages

Programming Language	Justification
Python 3.12+	Used for calibration, clustering, and model training with strong data science support.
TypeScript	Ensures type-safe backend inference and pipeline reliability.
Blueprint (UE)	Enables in-engine telemetry handling and adaptation without recompilation.

Table 19: Selection of Programming Languages

7.2.3. Development Framework

Framework	Purpose	Justification
Next.js (App Router)	Backend & Dashboard	Unified API and UI deployment with efficient server-side execution.
Jupyter Notebook	Research Pipeline	Supports iterative analysis and model development with visual feedback.

Table 20: Selection of Development Frameworks

7.2.4. Libraries / Toolkits

Category	Library	Justification
Data Manipulation	Pandas	Efficient telemetry data processing.
	NumPy	Core numerical computation support
	Scikit-learn	Clustering and feature scaling.
Machine Learning Visualisation	Matplotlib	Basic plotting for analysis.
	Seaborn	Statistical visualisation.
	Tailwind CSS	Rapid interface styling.
	Shadcn UI (Radix)	Prebuilt accessible UI components.
Backend UI	VaRest (Unreal Plugin)	Unreal-to-backend HTTP communication.
	Custom TypeScript MLP	Fast runtime inference.

Table 21: Selection of Libraries and Toolkits

7.2.5. Integrated Development Environments (IDEs)

IDE	Tier	Justification

VS Code	Full stack	Lightweight with strong TypeScript support.
Jupyter	Research	Interactive analysis and visualisation.
Unreal Engine	Game	Blueprint and PCG development.
PyCharm	Debugging	Advanced Python debugging.

Table 22: Selection of Integrated Development Environments

7.2.6. Summary of Technology Selection

Component	**Technology Selection**
Languages	Python, TypeScript, Blueprint
Frameworks	Next.js, Unreal Engine, Jupyter
Data	Pandas, NumPy, Matplotlib, Seaborn
ML	Scikit-learn, Custom MLP
Backend	Next.js, Tailwind, Shadcn UI
Engine	VaRest, PCG
IDEs	VS Code, Jupyter, Unreal Editor, PyCharm
Version Control	Git, GitHub
Package Management	npm, pip

Table 23: Summary of Technology Selection

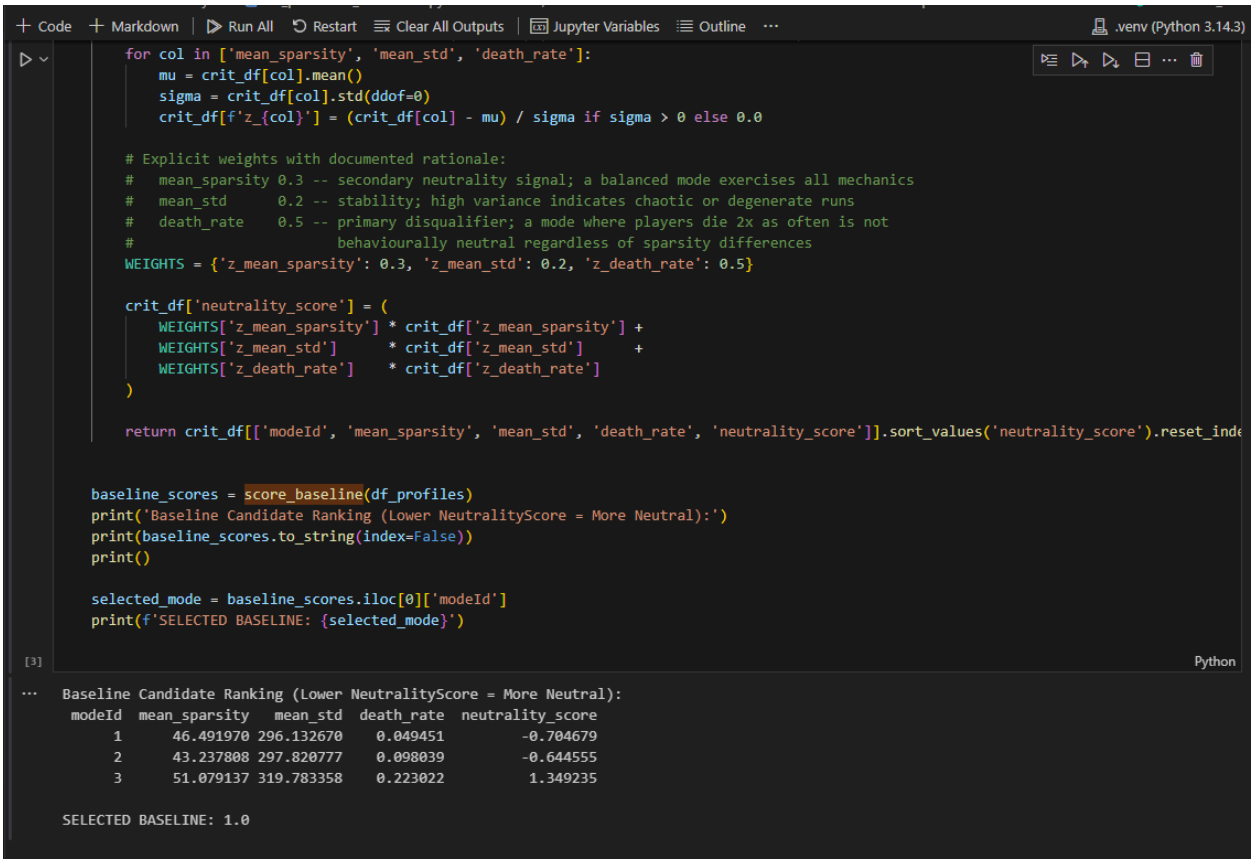
7.3. Core Functionalities Implementation

The AURA framework was implemented across three coordinated components: the calibration pipeline, the backend inference service, and the Unreal Engine client. The implementation adheres to strict phase separation, where calibration is performed offline, inference is handled exclusively within the backend pipeline, and the game client is responsible for telemetry generation and adaptation application. Detailed implementation evidence is provided in *Appendix J*.

7.3.1. Calibration Pipeline

The calibration pipeline establishes a neutral baseline configuration prior to adaptive execution. Telemetry collected across predefined gameplay modes was evaluated using behavioural sparsity, variation, and death-rate characteristics, which were z-score normalised and aggregated using named weights (sparsity: 0.3, standard deviation: 0.2, death rate: 0.5) to compute the

NeutralityScore.



```
+ Code + Markdown | Run All Restart Clear All Outputs | Jupyter Variables Outline ...  .venv (Python 3.14.3)

    for col in ['mean_sparsity', 'mean_std', 'death_rate']:
        mu = crit_df[col].mean()
        sigma = crit_df[col].std(ddof=0)
        crit_df[f'z_{col}'] = (crit_df[col] - mu) / sigma if sigma > 0 else 0.0

    # Explicit weights with documented rationale:
    #   mean_sparsity 0.3 -- secondary neutrality signal; a balanced mode exercises all mechanics
    #   mean_std      0.2 -- stability; high variance indicates chaotic or degenerate runs
    #   death_rate    0.5 -- primary disqualifier; a mode where players die 2x as often is not
    #                        behaviourally neutral regardless of sparsity differences
    WEIGHTS = {'z_mean_sparsity': 0.3, 'z_mean_std': 0.2, 'z_death_rate': 0.5}

    crit_df['neutrality_score'] = (
        WEIGHTS['z_mean_sparsity'] * crit_df['z_mean_sparsity'] +
        WEIGHTS['z_mean_std']      * crit_df['z_mean_std']      +
        WEIGHTS['z_death_rate']    * crit_df['z_death_rate']
    )

    return crit_df[['modeId', 'mean_sparsity', 'mean_std', 'death_rate', 'neutrality_score']].sort_values('neutrality_score').reset_inde


baseline_scores = score_baseline(df_profiles)
print('Baseline Candidate Ranking (Lower NeutralityScore = More Neutral):')
print(baseline_scores.to_string(index=False))
print()

selected_mode = baseline_scores.iloc[0]['modeId']
print(f'SELECTED BASELINE: {selected_mode}')

[3]                                                                                    Python

...  Baseline Candidate Ranking (Lower NeutralityScore = More Neutral):
 modeId  mean_sparsity   mean_std  death_rate  neutrality_score
      1      46.491970 296.132670    0.049451         -0.704679
      2      43.237808 297.820777    0.098039         -0.644555
      3      51.079137 319.783358    0.223022          1.349235

    SELECTED BASELINE: 1.0
```

Table 24: NeutralityScore Computation - Phase 1 Calibration Analysis (Self Composed)

As shown in above image, the configuration with the minimum NeutralityScore was selected as the baseline (Mode 1). This baseline was retained as a fixed reference for all subsequent modelling and runtime stages. All derived calibration artefacts were frozen to ensure consistency during adaptive execution.

7.3.2. Behavioural Modelling and Surrogate Training

Gameplay telemetry is transformed into three behavioural dimensions (Combat, Collection, Exploration) using grouped feature averages. Inverse-distance weighting (IDW) is applied to derive continuous soft membership values across archetypes, and temporal deltas are computed between consecutive windows to capture behavioural transitions *(Figure 19*: IDW Soft Membership Computation (Self Composed)).

```python
# Inverse-distance soft membership (add small epsilon to avoid /0)
inv_d = 1.0 / (dists + 1e-9)
soft  = inv_d / inv_d.sum(axis=1, keepdims=True)

row_sums  = soft.sum(axis=1)
max_dev   = np.abs(row_sums - 1.0).max()
passed    = (max_dev < 1e-6)
report(3, 'Soft membership row sums = 1', passed,
       f'max deviation = {max_dev:.2e}')
assert passed, f'Step 3 failed: max deviation {max_dev:.2e} >= 1e-6'
```

Figure 19: IDW Soft Membership Computation (Self Composed)

The adaptive reasoning surface is realised as a canonical ANFIS-inspired formula derived from Takagi-Sugeno fuzzy design principles, applied to the 3,240-window calibration dataset. This formula serves as the label generator (column anfis_teacher_output in 6_anfis_teacher_dataset.csv), producing target multipliers for training the MLP surrogate using six input features [μc, μo, μe, Δc, Δo, Δe]. The derivation is provided in *Appendix N*. This separates interpretable fuzzy modelling (offline) from low-latency inference (runtime), satisfying both explainability and performance requirements. Full configuration is provided in *Appendix J.5*.

The IDW formulation ensures stable soft membership computation, while the neutral-centred calibration function maintains a balanced output range [0.6, 1.4] with 1.0 as the neutral point. Scaling parameters derived from calibration are preserved as deployment artefacts for consistent runtime behaviour. Further implementation details are provided in *Appendix J*.

7.3.3. Backend Inference Service

The backend inference service is implemented as a decoupled runtime pipeline within a Next.js application. Precomputed calibration and modelling artefacts are loaded into memory at service startup to minimise runtime overhead.

Each telemetry request is processed through sequential stages: feature normalisation, activity scoring, soft membership derivation, temporal delta computation, surrogate inference, and output bounding. Telemetry ingestion (POST /api/unreal/telemetry on the Node.js backend) is architecturally separate from runtime inference (POST /api/pipeline/adapt), ingestion stores gameplay telemetry while the adapt endpoint executes the full pipeline and returns bounded PCG parameters

```
export async function POST(request: NextRequest) {
  try {
    // 1. Wake up the Brain
    const pipeline = getPipeline();

    // 2. Read the Message
    const body = await request.json();
    // NEW SCHEMA: userId is at root, telemetry contains the features directly
    const { userId, telemetry, reset } = body;

    console.error('\n🔵 [API] Incoming Pipeline Request:');
    console.error(JSON.stringify({ userId, telemetry, reset }, null, 2));

    // 3. Check the Rules (Validation)
    const validationError = validatePayload(telemetry, userId);
    if (validationError) {
      console.error('❌ [API] Validation Failed:', validationError);
      return NextResponse.json({ error: validationError }, { status: 400 });
    }

    // 4. Reset Memory (Optional)
    if (reset) {
      pipeline.reset();
    }

    // 5. Think! (Execution)
    // Construct internal TelemetryWindow from the new flattened API schema
    const internalTelemetry: TelemetryWindow = {
        userId: userId,
        timestamp: new Date().toISOString(), // Generate server-side timestamp
        features: telemetry, // The 'telemetry' in body IS the features object now
        duration: 30 // Default to trained window duration
    };

    const result = pipeline.process(internalTelemetry);

    console.error('🟢 [API] Outgoing Pipeline Response:');
    console.error(JSON.stringify(result, null, 2));

    // 6. Reply (Response)
    return NextResponse.json(result);

  } catch (error) {
    console.error('Runtime Error:', error);
    return NextResponse.json(
      {
        error: error instanceof Error ? error.message : 'Unknown Internal Error',
      },
      { status: 500 }
    );
  }
}
```

Figure 20: Backend inference pipeline endpoint (Self Composed)

As shown in *Figure 20*: Backend inference pipeline endpoint (Self Composed), the pipeline processes incoming telemetry through a deterministic sequence and returns a bounded multiplier together with adapted parameters derived from the calibration baseline. This ensures stable adaptation behaviour while maintaining real-time responsiveness. Detailed pipeline implementation is provided in *Appendix J*.

7.3.4. Unreal Engine Integration

The Unreal Engine integration is implemented through four coordinated Blueprint components forming a continuous telemetry–adaptation pipeline, with supporting evidence in *Appendix J*

(*Figure 45*: Phase 1 Mode Selection Interface - CollectGame (self-composed) - *Figure 48*: Response Handling and PCG Parameter Application Blueprint (Self Composed)). At session start, an anonymised mode selection interface is presented to eliminate calibration bias (*Figure 45*: Phase 1 Mode Selection Interface - CollectGame (self-composed)).

Gameplay telemetry is aggregated over fixed 30-second windows using the **BPC_AURA telemetry timer** (*Figure 46*: 30-Second Telemetry Window Timer Blueprint (Self Composed)*)*. At each interval, aggregated signals are dispatched asynchronously via the VaRest HTTP subsystem, constructed within **GI_Memory**, ensuring non-blocking communication with the backend inference service (*Figure 47*: VaRest HTTP POST - Telemetry Dispatch Blueprint (Self Composed)*)*.

Upon receiving a response, the system passes the adaptation multiplier and behavioural archetype memberships, which are processed through Blueprint functions and forwarded to the **BP_AURAVariation parameter applicator**. The *Update Values with AURA Features* function decomposes the feature structure into domain-specific structs (**S_PCGEnemy, S_PCGCollectibles**, and **S_PlayerCore**) before applying updates across gameplay systems (*Figure 48*: Response Handling and PCG Parameter Application Blueprint (Self Composed)). In failure scenarios, the system reverts to the Mode 1 neutral baseline to maintain stability.

A total of eleven PCG parameters are adapted at each 30-second boundary across four categories. Combat parameters control encounter intensity, exploration parameters regulate traversal efficiency, and collection parameters adjust resource distribution, each scaling with their respective behavioural memberships. Global player parameters scale uniformly with the multiplier, ensuring consistent difficulty adjustment.

All parameters are initialised from the Phase 1 neutral baseline (*Appendix J.1, Table 50*: PCG Mode Configuration Defaults - Three Child Blueprint Classes) and applied using a forward-only update strategy, where player attributes update immediately and PCG changes take effect in subsequent generation cycles.

At the implementation level, procedural generation is handled through **BP_EnemiesSpawner** and **BP_CollectibleSpawner,** which encapsulate **PCG_ScatterEnemies** and **PCG_ScatterCollectibles** graphs. These spawners receive updated parameter structs and regenerate content on timed cycles. Player attributes are updated via **BPC_DamageSystem** and **BPC_StaminaSystem**.

To ensure smooth transitions (FR09), bounded, forward-only, temporally smoothed adaptation is used on PCG configurations for next generated environment adaptation for a smooth transition of the game making it easier for player to adopt to the newly creating environment while preserving responsiveness. Runtime adaptation and integration behaviour are evidenced in *Appendix J.9* (from *Figure 51*: Telemetry aggregation and timed dispatch using BPC_AURA. *to Figure 64*: AURA In-Game PCG Parameter Adaptation - Base and Current Values (Self Composed)*)*.

7.4. User interface implementation

The system includes a gameplay interface and a developer-facing dashboard. The gameplay interface supports participant interaction, while the dashboard provides monitoring of behavioural distributions and adaptation outputs.

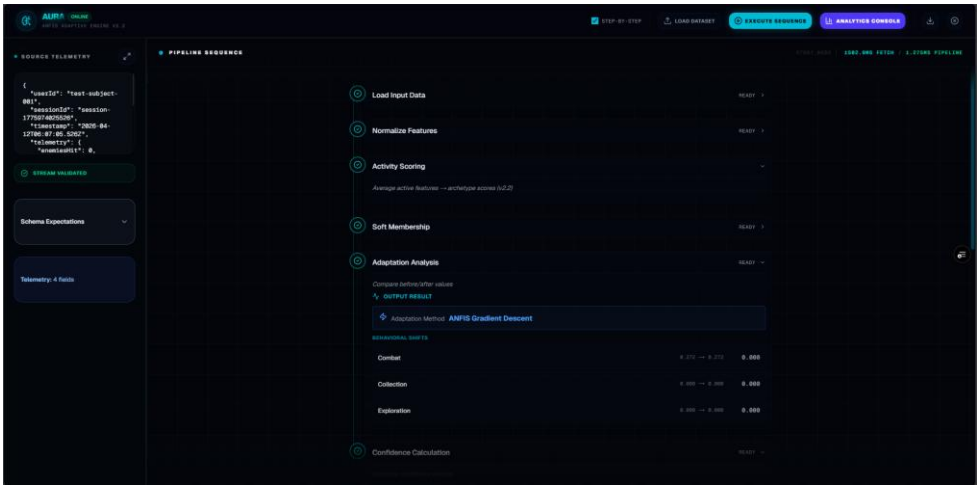


Figure 21: Developer analytics dashboard (Self Composed)

As shown in *Figure 21*: Developer analytics dashboard (Self Composed), the dashboard visualises archetype memberships and multiplier progression while remaining external to the participant experience, ensuring unbiased evaluation conditions. Detailed dashboard and gameplay screens implementation are provided in *Appendix I*.

7.5. Challenges and Solutions

Implementation challenges included inference latency, behavioural signal imbalance, backend cold-start delays, and numerical instability in membership computation. These were addressed through surrogate modelling, refined feature grouping, startup handling strategies, and epsilon

stabilisation. Detailed descriptions of each challenge and its resolution are provided in *Appendix J*.

7.6. Chapter Summary

This chapter presented the implementation of the AURA framework across its core components, including calibration, behavioural modelling, backend inference, and Unreal Engine integration. The system transforms gameplay telemetry into bounded adaptive responses through a decoupled pipeline. Detailed implementation artefacts and supporting evidence are provided in *Appendix J*. *Chapter 8* presents the testing and evaluation of the system.

CHAPTER 08: TESTING

8.1. Chapter Overview

This chapter presents testing procedures used to validate the correctness, performance, and robustness of AURA. Testing covers model evaluation, functional verification against Chapter 4 requirements, and non-functional performance assessment. Additional backend unit and API integration testing are included. The objectives are to validate the calibration baseline, clustering quality, surrogate accuracy, performance constraints, and identify limitations.

8.2. Testing Criteria

Testing criteria were defined across four categories to ensure coverage of both AI/ML components and the deployed system, as shown in Table 25: AURA Testing Criteria.

Testing Type	Scope	Evaluation Approach
AI/ML Model Testing	Evaluation of calibration baseline selection, K-Means clustering quality, and MLP surrogate accuracy against pre-defined quantitative thresholds on a held-out test set.	Quantitative metrics (R^2, MAE, MSE), bootstrap confidence interval, sensitivity analysis.
Functional Testing	Black-box verification of all 12 functional requirements from Chapter 4.	Test cases mapped to FR identifiers. Automated Vitest unit tests and API integration tests. Full test case table in *Appendix K.4*.
Non-Functional Testing	Assessment of all eight non-functional requirements from Chapter 4, covering performance, adaptation quality, efficiency, usability,	CI latency probes, Vitest edge-case tests, code structure review, manual access verification.

	scalability, reliability, maintainability, and security.	
Additional Testing	Automated unit test suits across two deployed backend services and a GitHub Actions CI integration workflow.	Vitest runner (62 tests across 13 files). GitHub Actions curl-based HTTP pipeline.

Table 25: AURA Testing Criteria

8.3. Model Testing

Model testing evaluated three components: calibration baseline selection, K-Means clustering, and MLP surrogate performance, each against predefined thresholds from *Chapter 03*.

8.3.1. Evaluation Metrics

The MLP surrogate predicts a continuous multiplier (M) and is evaluated using:

1. **R^2 (Coefficient of Determination)**: Measures the proportion of variance in the target multiplier explained by the model. An R^2 of 1.0 indicates a perfect fit. 0.0 indicates the model performs no better than predicting the mean. The pre-defined acceptance threshold was $R^2 \geq 0.90$.
2. **MAE (Mean Absolute Error)**: The average absolute deviation between predicted and actual multiplier values. Expressed as a percentage of the output range [0.6, 1.4] (range = 0.8) to provide domain-normalised interpretation. Threshold: < 2% of output range (i.e., < 0.016).
3. **MSE (Mean Squared Error)**: The average squared deviation between predictions and actual values. More sensitive to outliers than MAE used as a secondary diagnostic metric.
4. **Silhouette Score (Clustering)**: For the K-Means clustering component, the silhouette score measures how similar each window is to its own cluster relative to other clusters. Values range from −1 to +1 higher is better. The project-defined acceptable threshold was ≥ 0.3.

8.3.2. Phase 1 Calibration Testing

A within-subjects study (N=7) across three modes computed the NeutralityScore using z-score normalised criteria (mean sparsity, mean standard deviation, death rate) aggregated with named weights (0.3, 0.2, 0.5 respectively). The death-rate criterion receives the dominant weight because a mode with materially higher lethality cannot serve as a behaviourally neutral baseline. Mode 1 achieved the lowest composite score and was selected as the calibration baseline (Table 26: Calibration Mode NeutralityScore Comparison).

Mode ID	Mean Sparsity	Mean Std Dev	Death Rate	NeutralityScore
Mode 1 - Selected	46.49	296.13	0.0495	**−0.7047**
Mode 2	43.24	297.82	0.0980	−0.6446
Mode 3	51.08	319.78	0.2230	+1.3492

Table 26: Calibration Mode NeutralityScore Comparison

NeutralityScore is a z-score composite (weights: sparsity 0.3, std 0.2, death_rate 0.5). Lower = more neutral. Mode 1 selected. The baseline selection was assessed for directional robustness: Mode 1 retains the lowest composite score even under increased emphasis on the death-rate criterion, confirming it as the most conservative and neutral candidate across plausible weighting variations

8.3.3. Phase 2 Clustering Evaluation

K-Means clustering was applied to 3,240 telemetry windows from 45 sessions, using ten-dimensional per-window behavioural features (not aggregated summaries), ensuring fine-grained modelling. A grid search (108 configurations) identified K=3 with MinMaxScaler as optimal, achieving a silhouette score of 0.3752 (>0.3 threshold). The resulting clusters were labelled Combat, Collection, and Exploration, as shown in Figure 22: Archetype Distribution Across 3,240 Telemetry Windows (Self Composed). The search evaluated $K \in \{2–5\}$ across three scalers. Although K=2 produced a higher silhouette (0.4217), it merged Collection and Exploration, reducing semantic distinction. K=4 and K=5 yielded lower scores (0.3241, 0.2988) and unstable centroids. K=3 was selected as the best balance between clustering quality and interpretability.

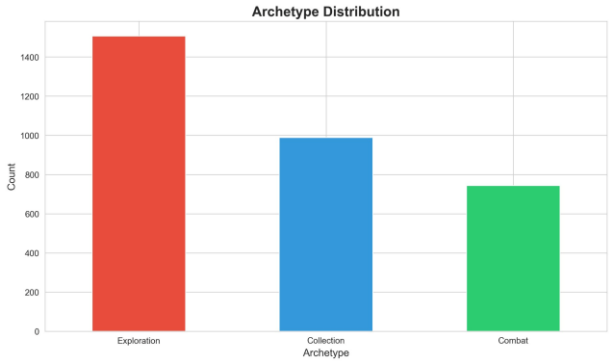


Figure 22: Archetype Distribution Across 3,240 Telemetry Windows (Self Composed)

Exploration dominates (46.5%), followed by Collection (30.5%) and Combat (23.0%), reflecting traversal-heavy gameplay. The silhouette score (0.3752) is below ideal separation but acceptable given overlapping behavioural signals and the use of soft membership. Full heatmap results are provided in *Appendix K.6*.

8.3.4. MLP Surrogate Evaluation

The MLP surrogate was trained on 3,240 windows using an 80/20 split with targets generated from the ANFIS-inspired surface (anfis_teacher_output). It approximates the canonical neuro-fuzzy behaviour with strong generalisation, achieving test $R^2 = 0.9264$ and MAE = 0.0127 (Table 27: MLP Surrogate Training and Test Performance Metrics).

Metric	Training Set	Test Set
R^2 (Coefficient of Determination)	0.8600	**0.9264 ✓ (threshold ≥ 0.90)**
MAE (Mean Absolute Error)	0.013926	**0.012705 ✓ (1.6% of range)**
MSE (Mean Squared Error)	0.000773	0.000385
MLP Neutral Output (mlp_neutral)	-	0.932006

Table 27: MLP Surrogate Training and Test Performance Metrics

The higher test R^2 is attributed to more mid-range targets, while training includes more extreme values that are harder to approximate. Low MSE and bootstrap analysis ($\sigma = 0.0229$) confirm no overfitting.

Figure 23: MLP Surrogate - Actual vs Predicted (R^2=0.9264) and Residual Distribution (Self Composed) presents the Actual vs Predicted scatter plot alongside the residual distribution

histogram generated on the held-out test set. The distribution of target multiplier values across the full dataset is presented in *Appendix K.9*.

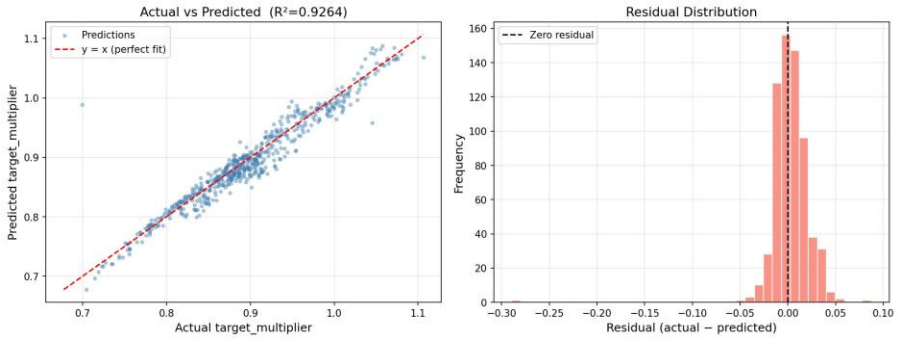


Figure 23: MLP Surrogate - Actual vs Predicted (R²=0.9264) and Residual Distribution (Self Composed)

Predictions align closely with the ideal fit, with residuals centred near zero. Both R^2 and MAE thresholds are satisfied, validating the surrogate for runtime deployment. Detailed residual plots are provided in *Appendix K.8*.

8.4. Benchmarking

Benchmarking compares the MLP surrogate with a naïve baseline and the ANFIS-inspired reference surface. The baseline has no explanatory power ($R^2 = 0$), while ANFIS represents a non-deployable upper bound. The MLP achieves $R^2 = 0.9264$, validating it as the runtime model.

Method	**Test R^2**	**Test MAE**	**Notes**
Naïve Baseline (mean predictor)	0.0000	~0.075	Predicts dataset mean (0.902) for all inputs. No explanatory power.
ANFIS Reference (label generator)	1.0000 (by construction)	0.0000	Generates training labels. Not deployable at runtime.
MLP Surrogate (AURA)	0.9264 ✓	0.0127 ✓	Exceeds $R^2 \geq 0.90$ threshold. Deployed as the runtime inference engine at sub-millisecond latency.

Table 28: Benchmarking - MLP Surrogate vs Baseline Methods

The surrogate achieves Test $R^2 = 0.9264$, substantially outperforming the baseline and approximating 92.64% of the ANFIS-defined surface at sub-millisecond latency. Direct ANFIS inference (27 rules) requires 50–200 ms, making the surrogate a practical alternative consistent

with ANFIS distillation (Wang and Mendel, 1992). Alternative models were considered analytically, with full derivation provided in Appendix N and empirical comparison designated as future work. The absence of an external non-adaptive baseline comparison (i.e., AURA-adapted gameplay versus static parameter gameplay) is acknowledged as the primary limitation of the evaluation design and is designated as the highest-priority future experiment

8.5. Further Evaluations

Two further evaluations were conducted to assess the statistical reliability and feature-level interpretability of the MLP surrogate.

8.5.1. Bootstrap Confidence Interval Analysis

A bootstrap confidence interval was computed over 100 resampling iterations on the held-out test set. The analysis yielded a mean R^2 of 0.9293, a standard deviation of 0.0229, and a 95% confidence interval of [0.874, 0.954]. The stored Test R^2 of 0.9264 lies within this interval, confirming statistical reliability. Full bootstrap results are presented in *Appendix K.1*.

8.5.2. Sensitivity Analysis

A univariate sensitivity analysis quantified the marginal impact of each of the six input features on the predicted multiplier M. Figure 24: Sensitivity Analysis - Impact of Each Input Feature on Predicted Multiplier M (Self Composed) presents the sensitivity curves.

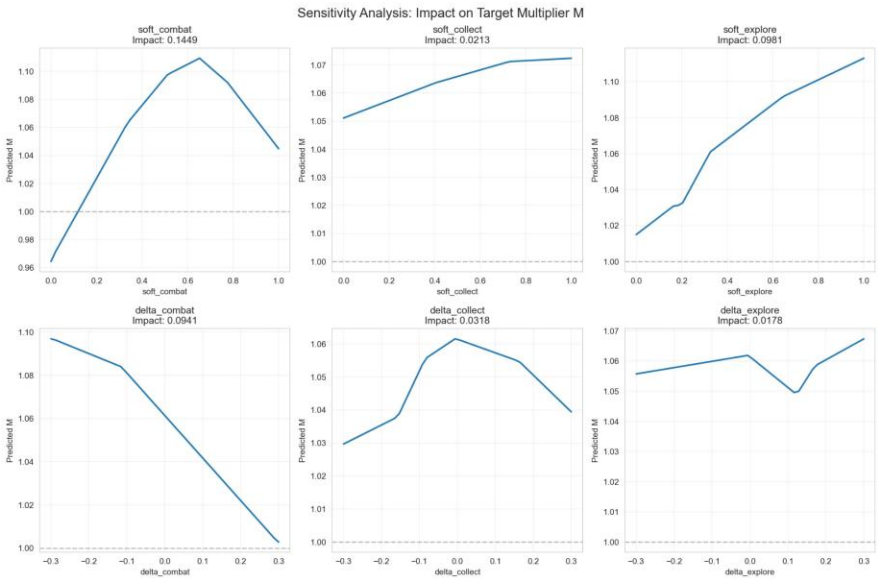


Figure 24: Sensitivity Analysis - Impact of Each Input Feature on Predicted Multiplier M (Self Composed)

Sensitivity analysis shows soft_combat as the dominant feature, followed by soft_explore and delta_combat, with soft memberships providing stable context and delta features capturing behavioural transitions. The three delta features contribute a combined impact of 0.1462, exceeding soft_explore (0.0981), supporting the 30-second windowed design.

Ablation confirms this effect: removing delta features reduces performance from $R^2 = 0.9726$ to 0.5891 (−38.35 pp), demonstrating that temporal signals are essential for accurate adaptation. A complementary comparison shows IDW soft membership reduces transition magnitude by 34.3% (0.6559 vs 0.9988), validating smoother, stable adaptation over hard clustering. Full results are provided in *Appendix K*.

The full ranked sensitivity scores are presented in *Appendix K.2*. Temporal delta distributions across the full dataset are presented Appendix K.7.

8.5.3. Player Experience Instrument Scope Decision

The research methodology (Chapter 3) identified the Game Experience Questionnaire (GEQ) and Self-Assessment Manikin (SAM) as candidate instruments for user testing. Following the completion of the calibration study (Phase 1, N=7) and the confirmation that the primary research question centred on architectural validity and surrogate model performance rather than player affect measurement, the deployment scope of these instruments was adjusted. The two-participant observational focus group conducted in Phase 3 (*Section 9.6.4*) was treated as pilot behavioural

evidence rather than a statistically powered user study. Administering GEQ and SAM instruments to a two-participant sample would not have produced results of sufficient validity to report, and doing so would have misrepresented the statistical power available. A controlled comparative user study employing validated GEQ and SAM instruments against both AURA-adaptive and non-adaptive gameplay conditions is designated as the primary direction for future empirical work.

8.6. Results Discussion

All three model components met their predefined thresholds. Mode 1 was selected as the neutral calibration baseline. The clustering result achieved a silhouette score of 0.3752, which is acceptable given the overlapping nature of continuous behavioural signals. The MLP surrogate demonstrated strong reliability, with a bootstrap-confirmed 95% confidence interval of [0.874, 0.954]. Sensitivity analysis shows that combat-related features have the strongest influence on the model output, aligning with the adaptive design rationale.

Direct ablation experiments were conducted for the two most critical design choices and are reported in *Section 8.5.2*: the delta ablation (R^2 lift +0.3835) confirms that temporal features are load bearing, and the hard-versus-soft comparison (34.3% transition magnitude reduction) validates the IDW soft membership design. For the 30-second window parameter, no ablation was executed, this was justified by prior empirical evidence (Bevilacqua, Engström and Backlund, 2019; Kowalik and Kapusta, 2025) and is indirectly supported by the sensitivity analysis in *Section 8.5.2*. These design decisions are further supported by the sensitivity evidence in *Section 8.5.2*, which confirms that delta features contribute a combined marginal impact of 0.1462, exceeding the standalone contribution of soft_explore (0.0981).

8.7. Functional Testing

Functional testing verified all 12 requirements using black-box methods, automated tests, and Unreal validation. The complete test case table is presented in *Appendix K.4*.

FR Coverage	Total FRs	Passed	Failed	Pass Rate	Status
FR-01 to FR-09, FR-11 (Must-Have / Should-Have)	10	10	0	100%	Fully Implemented
FR-10, FR-12	2	0	2	0%	Deferred

(Could-Have - formally deferred)					(future work)

Table 29: Functional Testing Summary - Must/Should-Have Requirements vs. Deferred Could-Have Items

8.8. Non-Functional Testing

Non-functional testing was conducted against all eight non-functional requirements defined in *Chapter 04*.

NFR	Requirement	Criteria	Test Method & Result	Outcome
NFR1	Performance	API warm-start round-trip < 200 ms	5× CI warm-run probes against /api/pipeline. All returned < 200 ms.	Passed
NFR2	Adaptation Quality	All output M ∈ [0.6, 1.4]	Vitest: all-zero, neutral, extreme combat edge cases. All three passed.	Passed
NFR3	Efficiency	Pipeline < 4 ms per call. MLP < 1 ms	MLP forward pass timed at sub-millisecond. Stateless per-request design.	Passed
NFR4	Usability	Developer dashboard accessible and clear	Manual HTTPS verification at Vercel deployment. All panels were rendered correctly.	Partially Passed
NFR5	Scalability	Supports new clusters/rules without retraining	Stateless inference. Frozen JSON artefacts allow model updates without code changes.	Passed
NFR6	Reliability	Stable operation and fallback on failure	Session reset and fallback were verified. Safe delta handling prevents invalid values, and audit fields confirm bounded, stable adaptation.	Passed
NFR7	Maintainability	Modular, documented codebase	Three-repository structure. MVC pattern. 62 automated tests provide regression safety.	Passed

NFR8	Security	Anonymised telemetry, input validation	HTTP 400 on missing userId/telemetry/empty body. No PII in payloads.	Partially Passed

Table 30: Non-Functional Testing Results - All NFR1–NFR8

NFR4 (Usability) is partially satisfied: the developer dashboard is fully implemented, but the in-engine configuration panel is scoped to Phase 4. NFR8 (Security) is partially satisfied: input validation and anonymisation are confirmed, but endpoint encryption remains a future enhancement.

8.9. Additional Testing

In addition to the testing types defined in *Section 8.2*, two supplementary automated testing procedures were implemented to provide regression-protected verification of the backend pipeline.

8.9.1. Backend Unit Testing

The collect-game-website inference pipeline was validated using Vitest v4.0.18 with 19-unit tests across 6 files, as shown in Figure 25: collect-game-website - 19 tests passed, 6 suites (Self Composed). The CollectGame.Telemetry backend was validated using Vitest v1.x with 43-unit tests across 7 files, as shown in Figure 26: CollectGame.Telemetry backend - 43 tests passing (Self Composed). The complete test suite breakdown is presented in *Appendix K.5*.

(i) collect-game-website inference

The collect-game-website inference pipeline was validated using Vitest v4.0.18 comprising 19-unit tests across 6 test files. Figure 25: collect-game-website - 19 tests passed, 6 suites (Self Composed) presents the Vitest terminal output confirming all 19 tests passed in 296 ms.

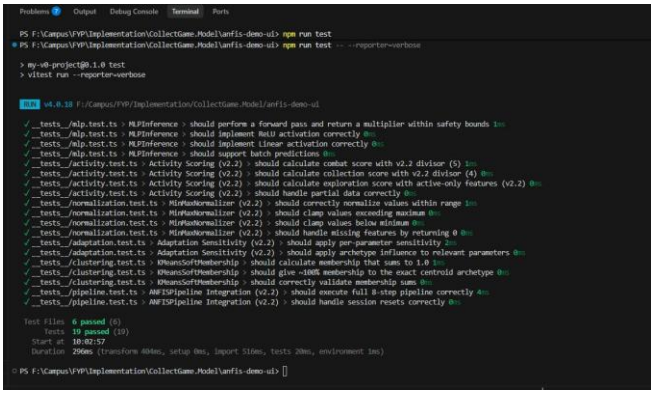


Figure 25: collect-game-website - 19 tests passed, 6 suites (Self Composed)

(ii) CollectGame.Telemetry backend

The CollectGame.Telemetry backend was validated using Vitest v1.x comprising 43-unit tests across 7 test files covering calibration controller logic, telemetry processing, notification service, user management, value object integrity, timezone handling, and end-to-end pipeline integration. Figure 26: CollectGame.Telemetry backend - 43 tests passing (Self Composed) presents the Telemetry backend test output.

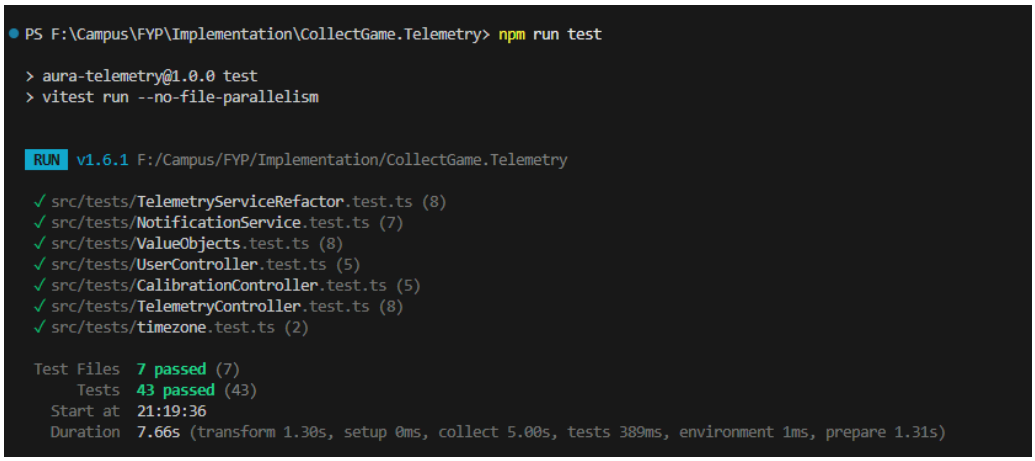


Figure 26: CollectGame.Telemetry backend - 43 tests passing (Self Composed)

8.9.2. API Integration Testing

A GitHub Actions CI workflow executes five HTTP test steps against the live /api/pipeline endpoint on every push to the main branch: valid payload (HTTP 200), missing userId (HTTP 400), missing telemetry (HTTP 400), empty body (HTTP 400), and five warm-run latency probes (all < 200 ms). Figure 27: GitHub Actions CI Pipeline - All Jobs Passed (Build, Unit Tests, 5 API Tests) (Self Composed) confirms all jobs passed. The full CI verification evidence is presented in

Appendix K.10.

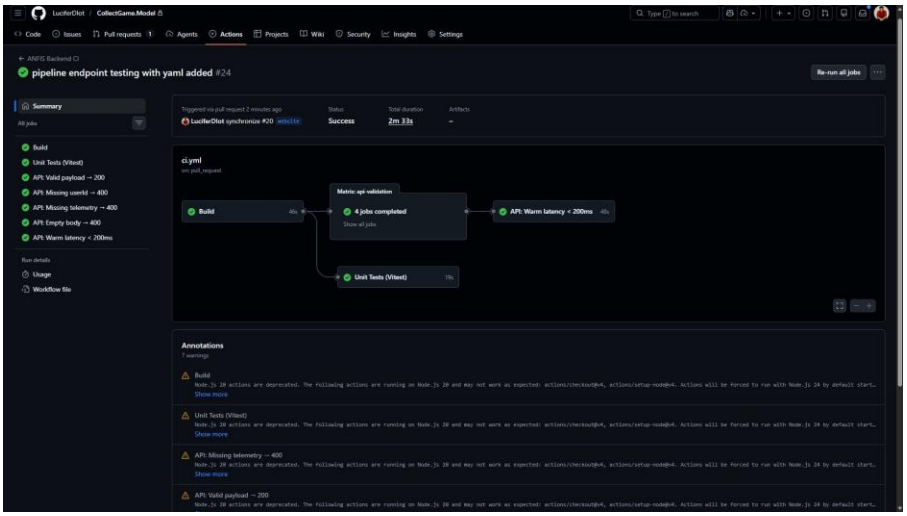


Figure 27: GitHub Actions CI Pipeline - All Jobs Passed (Build, Unit Tests, 5 API Tests) (Self Composed)

All three CI jobs (Build, Unit Tests, and API Tests) passed. The combined 62-test automated suite provides continuous regression protection against future code changes.

8.10. Limitations of the Testing Process

While the testing procedures comprehensively validated all implemented components, several constraints were encountered. Table 31: Limitations of the Testing Process presents the identified limitations alongside future improvement strategies.

Category	Limitation	Future Improvement
PCG Adaptation Scope	PCG parameter updates apply exclusively to future spawn cycles. The perceptual effect of procedural modulation on active gameplay has not been evaluated in a controlled adaptive-versus-static comparison.	Conduct a controlled A/B study comparing AURA-adapted gameplay against a non-adaptive baseline using validated engagement metrics (GEQ, SAM).
Dataset Scale	The model was trained on 3,240 windows from 45 participants in a single game environment. Generalisation of player populations with divergent behavioural patterns not represented	Expand participant pool, conduct cross-session longitudinal studies, and test surrogate generalisation across

	in CollectGame is untested.	alternative game environments.
Cold-Start Latency	Render free-tier enters a sleep state after periods of inactivity, producing cold-start round-trip times that exceed the NFR-01 threshold of 200 ms. This is an environmental rather than architectural constraint.	Migrate to a persistent-process deployment tier or alternative platform (e.g., Railway, Fly.io) to eliminate cold-start delays from the operational profile.
Unit Test Coverage Breadth	The combined suite (62 tests across 13 files: 19 in collect-game-website, 43 in CollectGame.Telemetry) covers core pipeline correctness. Edge cases for concurrent session handling, sub-field telemetry validation, and extreme multiplier combinations are not yet covered.	Expand to property-based tests for edge case inputs, concurrent session stress tests, and expanded schema validation assertions.

Table 31: Limitations of the Testing Process

8.11. Chapter Summary

This chapter evaluated the AURA framework, confirming model performance, behavioural clustering validity, and surrogate accuracy. Benchmarking and sensitivity analysis demonstrated effective adaptive behaviour. Functional testing achieved a 100% pass rate for all prioritised requirements (FR01–FR09, FR11), while FR10 and FR12 were planned deferrals. Non-functional testing confirmed overall system performance, with only minor limitations noted.

CHAPTER 09: CRITICAL EVALUATION

9.1. Chapter Overview

This chapter presents the critical evaluation of the AURA framework through structured assessment by eight industry evaluators and two end-user participants, concluding with a formal review of the functional and non-functional requirements defined in *Chapter 04*. Self-evaluation, expert feedback, and end-user observations collectively validate the research concept, solution architecture, and implementation quality of the framework.

9.2. Evaluation Methodology and Approach

Both qualitative and quantitative methods were used to evaluate the AURA framework. Qualitative feedback was obtained from domain and technical experts, supported by end-user observations to assess usability and practical effectiveness. Feedback was collected through structured discussions and written responses based on shared evaluation materials. Thematic analysis was applied to identify recurring patterns, guided by the predefined evaluation criteria to ensure consistent assessment.

9.3. Evaluation Criteria

Six evaluation criteria were defined to ensure systematic and comprehensive assessment of all dimensions of the AURA framework. The complete criteria and their corresponding objectives are presented in Table 56: Evaluation Criteria in *Appendix L.1*.

9.4. Self-Evaluation

Self-evaluation was conducted by the author against the criteria defined in *Section 9.3*. Table 32: Self-Evaluation (Partial) presents the first two criteria inline and the detailed evaluation criteria of the author's self-evaluation is presented in *Appendix L.2*.

Criterion	Author's Self-Evaluation
Concept and novelty	AURA introduces a calibration-first, interpretable adaptive pipeline that bridges reactive DDA and opaque learning-based systems. Its neutral

	baseline and soft behavioural membership provide a clear and structured contribution to adaptive game design.
Scope, depth, and complexity	The project spans three fully implemented phases, including calibration, behavioural modelling, and backend deployment. The integration of data analysis, the project implements three phases (calibration, behavioural modelling, and backend deployment) integrating data analysis, machine learning, and real-time systems. This demonstrates substantial technical depth beyond typical undergraduate scope.
Design and architectural decisions	The three-phase architecture separates concerns: calibration defines the baseline, Phase 2 builds the model, and Phase 3 handles inference without retraining. Bounded multipliers, forward-only adaptation, and temporal smoothing ensure stability. The primary limitation is the absence of a controlled adaptive-versus-static gameplay comparison, meaning the perceptual impact of procedural parameter modulation has not been empirically separated from the baseline condition. This is designated as the primary future experimental direction
Implementation of quality and technical efficiency	The MLP surrogate achieved high accuracy ($R^2 = 0.9264$, MAE = 0.0127) with strong validation. Backend latency remains below 200 ms, supported by automated testing and CI. A limitation is the absence of a formal controlled A/B evaluation separating the perceptual impact of parameter modulation from the baseline condition.
Adaptive stability and behavioural alignment	Temporal smoothing and bounded adaptation prevent instability, while soft membership ensures balanced behavioural influence. The calibrated neutral baseline ensures stable, unbiased adaptation around a multiplier of 1.0.
Limitations and future improvement directions	Key limitations include small calibration sample size, restricted Unreal integration scope, backend cold-start delays, absence of a formal adaptive-versus-static perceptual comparison, and limited test coverage. These define clear directions for future improvement rather than weakening the contribution.

Table 32: Self-Evaluation (Partial)

9.5. Selection of Evaluators

Eight Eight domain and technical evaluators were recruited, supplemented by one additional domain specialist, bringing the evaluation panel to 8+ participants. Domain experts (DE-01 to DE-05, and additional specialist) held active professional backgrounds in game development, adaptive game systems, and interactive software design. Technical experts (TE-01 to TE-04 specialised in Unreal Engine development, software architecture, and machine learning deployment. This composition reflects the dual research-and-engineering nature of the AURA framework. All evaluators were active professionals at the time of evaluation. The complete evaluator roster is provided in *Appendix L.3*. Selection of Evaluators, Table 58: Selection of Evaluators.

9.6. Evaluation Results

9.6.1. Expert Opinion

The AURA framework was evaluated through expert feedback targeting the criteria established in *Section 9.3*. Thematic analysis was conducted across all eight evaluator responses to identify recurring themes. The table below presents the consolidated thematic analysis findings and complete per-evaluator findings are presented in *Appendix L.4*.

Criteria	Category	Theme	Conclusion
Concept & novelty	Innovation and relevance	Calibration-First design	Experts recognised the calibration-first approach as a clear departure from conventional systems, with ANFIS-based modelling offering strong research novelty and publication potential.
Scope & complexity	Technical depth	Beyond undergraduate scope	The three-phase pipeline, full deployment, and tested backend were identified as evidence of high engineering rigour beyond typical undergraduate work.
Design decisions	Architecture	Decoupled backend & soft	IDW soft membership was validated for modelling overlapping behaviours, while the

		membership	decoupled backend and bound adaptation were confirmed as appropriate and scalable design choices.
Implement ation quality	Efficiency	Pipeline performance	The surrogate model performance and low latency were commended, with smoother parameter transitions identified as a key improvement area.
Adaptive stability	User experience	Temporal smoothing	Temporal smoothing was confirmed as essential for stable adaptation, with users reporting gradual and natural difficulty adjustments.
Limitations & future work	Expansion	Dataset & generalisation	Dataset size was noted as sufficient for prototyping but requiring expansion, with cross-session adaptation highlighted as a key future enhancement.

Table 33: Evaluation Results - Thematic Analysis

9.6.2. Domain Experts

(i) Concept

All four domain experts affirmed the calibration-first neuro-fuzzy approach as a principled and industry-relevant departure from conventional adaptive systems. The use of ANFIS-guided modelling was independently identified as the academically distinctive feature of the framework, with strong potential for research publication noted across multiple responses.

(ii) Solution

The IDW soft membership design was commended as the correct approach for handling the inherent overlap in continuous player behaviour, with the silhouette score of 0.3752 presented as validation evidence. The bounded multiplier [0.6, 1.4] and forward-only adaptation constraint were validated as industry-standard stability mechanisms. Expanding the calibration cohort to 30–50 diverse participants was recommended for production-directed validation.

9.6.3. Technical Experts

(i) Scope

Technical experts confirmed that the phased pipeline structure demonstrates the system was developed as a framework grounded in prior calibration and systematic modelling decisions, substantially exceeding typical undergraduate research scope and demonstrating genuine engineering rigour.

(ii) Architecture of the Solution

Decoupling ML inference from the game engine was confirmed as the architecturally correct decision, as in-engine inference would consume frame-budget resources impractical for most productions. The staged pipeline was affirmed as providing a level of interpretability absent from fully black-box adaptive systems, and temporal smoothing was validated as a critical stability design choice based on first-hand shipped-game experience.

(iii) Implementation of the Solution

The end-to-end implementation was confirmed as technically feasible and practically game-integration ready. Multiple evaluators independently identified smooth parameter transitions as a quality-of-polish priority. The bounded multiplier and forward-only adaptation were confirmed as correctly addressing the risk of diminishing challenge-mastery satisfaction in unconstrained adaptive systems by only influencing the next generated content, within a controlled range, and only when a meaningful behavioural shift is detected. The complete per-evaluator findings are presented in *Appendix L.4*.

9.6.4. End-user observational sessions

(i) Prototype Features

Two volunteer participants engaged in observed gameplay sessions of approximately 20–25 minutes without prior knowledge of the adaptive pipeline. Given the small sample size, these sessions are treated as observational pilot evidence rather than a statistically representative usability study, the findings are nonetheless informative as indicative user feedback. Both reported adaptation behaviour as natural and gradual, with no perceived difficulty spikes or abrupt environmental shifts during their sessions. The complete focus group findings are presented in *Appendix L.5*.

(ii) Usability

Both participants affirmed that the core gameplay loop was accessible and that adaptive elements did not disrupt their understanding of or control over the game experience. Both independently suggested a player-facing transparency mechanism either an in-game visual cue or post-session adaptation summary corroborating domain expert feedback regarding this area as the clearest direction for future player experience enhancement.

9.7. Limitations of Evaluation

The evaluation encountered several practical limitations. Feedback was gathered asynchronously through written responses and video-call sessions rather than direct in-person observation, and the absence of a standardised questionnaire instrument introduces variability in response focus, partially mitigated through thematic analysis. A formal comparative user study benchmarking AURA-adapted gameplay against a non-adaptive baseline was not conducted, representing the strongest direction for future empirical validation. The focus group was limited to two end-user participants, constraining the statistical generalisability of the usability observations.

The main limitation is that a controlled comparison between adaptive and non-adaptive gameplay was not conducted. This was a deliberate scope decision, focusing on validating the architecture and surrogate model within the available timeframe. A full player study requires ethical approval, larger samples, and validated measures (GEQ/SAM), which were beyond scope. Pilot observations provide only indicative insights, marking player-outcome evaluation as future work.

9.8. Functional and Non-Functional Requirements Implementation

All ten Must-Have functional requirements (FR01–FR08, FR11) were fully implemented across the three delivered phases, including PCG parameter adaptation (FR07) and forward-only parameter application within latency constraints (FR08). FR06 and FR09 were partially implemented with outstanding elements relating to in-engine visualisation and automated reversal. FR10 and FR12 remain future scope as C-priority items. All eight non-functional requirements were implemented or partially implemented. NFR4 (Usability) is partially satisfied as the in-engine developer configuration panel remains a future enhancement. NFR8 (Security) is partially satisfied as endpoint encryption remains a planned improvement. The complete functional requirements

evaluation is provided in *Appendix L.6* and the non-functional requirements evaluation in L.7. Evaluation of Non-Functional Requirements Appendix L.7.

9.9. Chapter Summary

This chapter evaluated the AURA framework through expert review and end-user observation. Evaluators confirmed the calibration-first design, soft membership modelling, decoupled architecture, and bounded adaptation as strong and practical contributions. Users found the adaptation natural and non-intrusive, with recommendations for improved transparency. All Must-Have functional requirements and all core non-functional requirements were successfully implemented.

CHAPTER 10: CONCLUSION

10.1. Chapter Overview

This chapter concludes the AURA research project by reflecting on the achievement of the stated research aim and objectives, documenting the utilisation of academic knowledge and skills, and identifying the problems encountered alongside their mitigations. The chapter further examines deviations from the original project plan, acknowledges the limitations of the research, and proposes directions for future enhancement. Contributions to the research and problem-domain bodies of knowledge are articulated, and the chapter closes with concluding remarks on the overall significance of the AURA framework.

10.2. Achievement of Research Aims and Objectives

10.2.1. Achievement of the Research Aim

The aim of this research was to design, implement, and evaluate a calibration-first, telemetry-driven adaptive control framework capable of delivering stable, interpretable, and real-time procedural environment modulation, seeking to mitigate cold-start bias, model behavioural fluidity through soft clustering and temporal dynamics, and deploy neuro-fuzzy surrogate reasoning within a latency-aware backend architecture while preserving player agency and perceptual continuity. This aim was achieved through the three-phase AURA pipeline, with Phase 1 establishing a neutral calibration baseline (z-score NeutralityScore -0.7047, Mode 1 selected), Phase 2 delivering a K-Means soft membership model and MLP surrogate (R^2=0.9264), and Phase 3 producing a deployed inference service confirming sub-200 ms latency, ten of twelve functional requirements validated within scope, and bounded multiplier output within [0.6, 1.4].

10.2.2. Achievement of Objectives

Thirteen of the fourteen objectives were fully achieved. R13 was completed, with archetype proportions validated as interpretable via the analytics dashboard (Chapter 9). R14 was partially achieved through publication of a dataset and reproducible pipeline on Kaggle, with formal publication planned post-submission. The extended Unreal PCG configuration panel was descoped

as a Could-Have item. Full objective mapping is provided in *Appendix M.1*.

10.3. Utilisation of Knowledge from the Course

The following degree modules were most directly applicable to the AURA project:

- Machine Learning and Data Mining
- Algorithms: Theory, Design and Implementation
- Object Oriented Programming
- Server-Side Web Development
- Mathematics for Computing

The complete justification mapping each module to the specific pipeline components and artefacts it informed is presented in *Appendix M.2*.

10.4. Use of Existing Skills

Existing skills in Python and the scientific computing stack were applied in Phase 1 calibration and Phase 2 modelling for data processing and surrogate training. TypeScript and Node.js supported the Phase 3 backend inference service using established software engineering patterns. Prior experience with React, Next.js, and Git enabled development of the analytics dashboard and structured deployment workflow.

10.5. Use of New Skills

The following technical skills were acquired specifically through the AURA project and were not part of the prior degree curriculum:

- ANFIS and neuro-fuzzy systems: No prior knowledge existed before the project. ANFIS theory covering membership function convergence, hybrid learning rules, and fuzzy surface fitting was independently studied and applied to build the Phase 2 offline training pipeline.
- K-Means soft membership with IDW weighting: Soft membership computation via inverse distance weighting, the partition-of-unity constraint ($\Sigma\mu_k=1.0$), and silhouette-based cluster evaluation were learned beyond the degree curriculum and applied throughout Phase 2 behavioural modelling.

- MLP surrogate design and validation: Grid search across 108 configurations, bootstrap confidence interval validation (95% CI [0.874, 0.954]), and neutral-centred multiplier calibration were learned and applied during Phase 2 to deliver a runtime-compliant surrogate model.
- Unreal Engine 5.7, VaRest, and PCG framework: Blueprint scripting, the VaRest HTTP plugin, Set Timer by Event construction, and the PCG framework were learned in parallel with the project through official UE documentation and community resources.

10.6. Achievement of Learning Outcomes

The achievement of all eight learning outcomes for the BEng Final Year Project module is presented in *Appendix M.3*.

10.7. Problems and Challenges Faced

The following table summarises the primary problems and challenges encountered during the AURA project, together with the mitigation strategy applied in each case.

Problem / Challenge	**Mitigation Strategy**
MLP surrogate inference latency exceeding the sub-200 ms real-time budget	An MLP surrogate (6→16→8→1, ReLU) was selected as the runtime inference engine from the outset. ANFIS was retained solely for offline training to fit the fuzzy surface. The surrogate achieved Test R^2=0.9264 and operates at sub-1 ms per inference call.
Passive signal contamination and archetype scoring bias (v2.0 → v2.1)	The timeOutOfCombat variable was removed from the Exploration archetype formula. Activity scores were converted to per-archetype feature means. The full modelling pipeline was rerun without requiring new data collection.
Cold-start latency on the Render free-tier deployment environment	A BeginPlay HTTP GET health-check request is dispatched at session start to warm the container before the first 30-second telemetry window, eliminating cold-start delay from the active session lifecycle.
IDW floating-point	An epsilon stabilisation term $\varepsilon=1\times10^{-6}$ was added to all centroid

precision drift when centroid distances approach zero	distance computations in both the Python modelling pipeline and the TypeScript backend. Vitest unit tests verify the fix under near-zero distance conditions.
Recruiting a sufficient calibration cohort within the project timeline	Recruitment was broadened to include peers from adjacent degree programmes. A within-subjects randomised-order design was adopted to maximise statistical yield from the available N=7 participants across all three game modes.
Coordinating three parallel repositories with interdependent artefact dependencies	A versioning protocol (v1.0→v2.2.1) was applied across all three repositories. Frozen artefact files (scaler_params.json, anfis_mlp_weights.json, centroids.json) were used as immutable interface contracts between pipeline phases.

Table 34: Problems and Challenges Faced

10.8. Deviations

Two deviations from the original project plan were identified and managed. Archetype labels were revised from informal proficiency categories to behaviour-derived identifiers (Combat, Collection, and Exploration) following the K-Means analysis. The originally proposed A/B control group was replaced by a within-subjects randomised-order calibration study (N=7, three game modes) to provide stronger internal validity within the recruitment constraints.

10.9. Limitations of the Research

The following limitations were identified in the AURA research

- **Calibration dataset scale:** The N=7 cohort is insufficient for statistically stable archetype centroids at production scale results cannot be generalised to a broad player population without a larger study.
- **Formal PCG adaptation evaluation**: While PCG parameter adaptation is implemented and operational across eleven parameters in four categories, a controlled A/B comparison between AURA-adapted and non-adaptive gameplay configurations was not conducted. The perceptual impact of procedural content modulation on player engagement and immersion therefore remains empirically unvalidated.
- **Single game genre**: CollectGame is a single action-collection prototype. Archetypes and

multiplier calibration are specific to its mechanical affordances and may not generalise across genres.

- **No Cross-Session persistence:** Temporal state resets between sessions and longitudinal personalisation across multiple play sessions is not supported in the current architecture.
- **Telemetry Consent Enforcement:** User consent is captured at registration but is not enforced at the telemetry ingestion endpoint. Production deployment should validate consent at the API boundary to ensure full compliance with GDPR Article 7. This is identified as a planned NFR8 security enhancement.

10.10. Future Enhancements

10.10.1. Research Domain

- Formal Feature Ablation Study: Re-execution of the pipeline under three conditions (soft memberships only, soft memberships with deltas, and the full 6-feature pipeline) on an independently collected dataset (minimum N=30 new sessions), building on the current ablation result (R^2 lift +0.3835) with proper independent validation.
- Cross-session personalisation: Accumulate behavioural history across sessions to enable longitudinal adaptive evolution and expand the calibration cohort to 30–50 participants to establish statistically robust archetype baselines at scale.

10.10.2. Problem Domain

- Comparative APCG Outcome Study: A within-subjects A/B experiment ($N \geq 20$) comparing AURA-adaptive and static-parameter gameplay using the GEQ Immersion subscale and SAM valence as primary outcomes. Two 25-minute sessions per participant, 48-hour washout interval.
- Cross-Genre Generalisation and Robustness: Application of the calibration-first pipeline to a second game prototype, combined with adversarial input testing (missing fields, zero-activity windows, session timeout boundaries) to validate clamping and fallback behaviour beyond the CollectGame prototype.

10.11. Achievement of the Contribution to Body of Knowledge

10.11.1. Research Domain Contributions

- Calibration-first methodology: A phase-separation principle isolating baseline construction from live inference. A methodological safeguard against feedback contamination not previously formalised in adaptive PCG literature.
- IDW soft membership for behavioural classification: Continuous archetype proportions derived from K-Means centroids via inverse distance weighting, demonstrating a principled alternative to hard cluster assignment for game behavioural modelling.
- PAD-Grounded Archetype Formalisation: Combat, Collection, and Exploration archetypes are mapped to PAD (Pleasure–Arousal–Dominance) dimensions (Mehrabian and Russell, 1974), establishing a theoretically grounded behavioural ontology for adaptive PCG and a formalisation of these archetypes within the PAD model.
- ANFIS-Inspired Surrogate Design Pattern: Shows that interpretable neuro-fuzzy reasoning can be approximated by an MLP for real-time use, validated by delta ablation (R^2 +0.3835) and 34.3% smoother transitions (see *Appendix N*).

10.11.2. Problem Domain Contributions

- End-to-end deployed adaptive PCG backend: A production-deployed pipeline combining telemetry-driven clustering, neuro-fuzzy surrogate inference, and neutral-centred calibration, accessible via a live API endpoint.
- Sub-200 ms inference on free-tier cloud: Validates the commercial viability of decoupled game AI backends for independent studios without dedicated server infrastructure.
- Automated validation suite and live dashboard: Open-source engineering infrastructure providing regression-protected verification and real-time pipeline observability at https://collect-game-website.vercel.app/.

10.12. Concluding Remarks

The AURA project addressed a clearly defined gap: the absence of a framework that is simultaneously calibration-separated, behaviourally interpretable, real-time capable, and stability-constrained. Through three phase-separated components (calibration baseline construction, soft membership behavioural modelling, and decoupled backend inference), the system demonstrated

that neuro-fuzzy surrogate reasoning and bounded adaptive control can be integrated into a coherent, deployable solution without sacrificing interpretability or performance.

The empirical outcomes directly validate all four research questions: phase separation produced a statistically neutral baseline (z-score NeutralityScore = -0.7047, Mode 1 selected, RQ1), 30-second temporal delta modelling confirmed stable, oscillation-free adaptation (RQ2), IDW soft membership (silhouette=0.3752) captured behavioural fluidity without hard classification artefacts (RQ3) and the MLP surrogate (R^2=0.9264, 95% CI [0.874, 0.954], <200 ms warm-start latency) confirmed that neuro-fuzzy reasoning is deployable under real-time constraints (RQ4). Ten of twelve functional requirements were validated within scope, supported by 62 automated tests under continuous integration.

AURA's calibration-first methodology, IDW soft membership pipeline, and ANFIS-inspired canonical surrogate architecture constitute transferable design principles applicable beyond the CollectGame prototype, establishing a reproducible and extensible foundation for adaptive game AI research.

REFERENCES

Alyaseri, S. (2025). Generative AI in Procedural Content Generation for Computer Games: Key Contributions and Trends. Available from https://doi.org/10.36227/techrxiv.174317787.76161480/v1 [Accessed 4 November 2025].

Bawa, H. and Aponso, A. (2025). Addressing the Cold Start Problem of Recommendation Systems: A Comprehensive Review of Traditional and Emerging Solutions. *2025 10th International Conference on Information and Network Technologies (ICINT)*. 12 March 2025. Melbourne, Australia: IEEE, 41–46. Available from https://doi.org/10.1109/ICINT65528.2025.11030887 [Accessed 12 February 2026].

Bevilacqua, F., Engström, H. and Backlund, P. (2019). Game-Calibrated and User-Tailored Remote Detection of Stress and Boredom in Games. *Sensors*, 19 (13), 2877. Available from https://doi.org/10.3390/s19132877.

Bontchev, B. (2016). Adaptation in Affective Video Games: A Literature Review. *Cybernetics and Information Technologies*, 16 (3), 3–34. Available from https://doi.org/10.1515/cait-2016-0032.

Bran, A. and Vaidis, D.C. (2020). On the Characteristics of the Cognitive Dissonance State: Exploration Within the Pleasure Arousal Dominance Model. *Psychologica Belgica*, 60 (1), 86–102. Available from https://doi.org/10.5334/pb.517.

Caldas, R.D. et al. (2020). A hybrid approach combining control theory and AI for engineering self-adaptive systems. *Proceedings of the IEEE/ACM 15th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*. 29 June 2020. Seoul Republic of Korea: ACM, 9–19. Available from https://doi.org/10.1145/3387939.3391595 [Accessed 7 January 2026].

Chorrojprasert, S. (2025). Exploring the Effects of Dynamic Difficulty Adjustment Functions in Running Exergames on Player Experience and Physical Effort.

Etheredge, M., Lopes, R. and Bidarra, R. (2013). A Generic Method for Classification of Player

Behavior. *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, 9 (3), 2–8. Available from https://doi.org/10.1609/aiide.v9i3.12593.

Fernandes, P.M., Lopes, M. and Prada, R. (2024). Generating Game Levels by Defining Player Experiences. *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, 20 (1), 179–188. Available from https://doi.org/10.1609/aiide.v20i1.31878.

Freiknecht, J. and Effelsberg, W. (2017). A Survey on the Procedural Generation of Virtual Worlds. *Multimodal Technologies and Interaction*, 1 (4), 27. Available from https://doi.org/10.3390/mti1040027.

Gisslen, L. et al. (2021). Adversarial Reinforcement Learning for Procedural Content Generation. *2021 IEEE Conference on Games (CoG)*. 17 August 2021. Copenhagen, Denmark: IEEE, 1–8. Available from https://doi.org/10.1109/CoG52621.2021.9619053 [Accessed 9 November 2025].

Hendrikx, M. et al. (2013). Procedural content generation for games: A survey. *ACM Transactions on Multimedia Computing, Communications, and Applications*, 9 (1), 1–22. Available from https://doi.org/10.1145/2422956.2422957.

Hunicke, R. (2005). The case for dynamic difficulty adjustment in games. *Proceedings of the 2005 ACM SIGCHI International Conference on Advances in computer entertainment technology*. 15 June 2005. Valencia Spain: ACM, 429–433. Available from https://doi.org/10.1145/1178477.1178573 [Accessed 23 February 2026].

Hunicke, R. and Chapman, V. (2004). AI for Dynamic Difficulty Adjustment in Games.

Huo, Y. et al. (2023). Adaptive critic design for nonlinear multi-player zero-sum games with unknown dynamics and control constraints. *Nonlinear Dynamics*, 111 (12), 11671–11683. Available from https://doi.org/10.1007/s11071-023-08419-5.

Kaimara, P., Oikonomou, A. and Deliyannis, I. (2021). Could virtual reality applications pose real risks to children and adolescents? A systematic review of ethical issues and concerns.

Virtual Reality, 26, 697–735. Available from https://doi.org/10.1007/s10055-021-00563-w.

Kalafatis, E. et al. (2025). A modular framework for automated evaluation of procedural content generation in serious games with deep reinforcement learning agents. *IEEE Transactions on Games*, 1–10. Available from https://doi.org/10.1109/TG.2025.3589439.

Kanwal, M., Siddiqui, M.S. and Ali, S.A. (2025). Unveiling Gamer Archetypes through Multi modal feature Correlations and Unsupervised Learning. Available from https://doi.org/10.48550/arXiv.2510.10263 [Accessed 13 February 2026].

Khoshnoud, S., Alvarez Igarzábal, F. and Wittmann, M. (2020). Peripheral-physiological and neural correlates of the flow experience while playing video games: a comprehensive review. *PeerJ*, 8, e10520. Available from https://doi.org/10.7717/peerj.10520.

Kowalik, J. and Kapusta, P. (2025). Orchestrating Player Affect: A Closed-Loop Transformer Architecture for Targeted Emotional Induction in Mobile Games. Available from https://doi.org/10.20944/preprints202512.1305.v1 [Accessed 13 February 2026].

Kyiv, Ukraine and Sulyma, Y. (2025). Dynamic Difficulty Algorithms as a Tool for Enhancing Player Retention: An Empirical Study in a Gaming Environment. *The American Journal of Engineering and Technology*, 07 (06), 115–123. Available from https://doi.org/10.37547/tajet/Volume07Issue06-12.

Lara-Alvarez, C. et al. (2021). Induction of Emotional States in Educational Video Games Through a Fuzzy Control System. *IEEE Transactions on Affective Computing*, 12 (1), 66–77. Available from https://doi.org/10.1109/TAFFC.2018.2840988.

Leong, W.Y. (2025). Autonomous Game Balancing: AI-Driven Dynamic Difficulty Adjustment and Fairness Metrics.

Liang, B., Wang, Y. and Tong, C. (2025). AI Reasoning in Deep Learning Era: From Symbolic AI to Neural–Symbolic AI. Available from https://www.mdpi.com/2227-7390/13/11/1707.

Liu, J. et al. (2021). Deep learning for procedural content generation. *Neural Computing and Applications*, 33 (1), 19–37. Available from https://doi.org/10.1007/s00521-020-05383-8.

Lopes, P., Fachada, N. and Fonseca, M. (2025). Closing the Loop: A Systematic Review of Experience-Driven Game Adaptation. Available from https://doi.org/10.48550/arXiv.2505.01351 [Accessed 12 February 2026].

Maddalon, L. et al. (2024). Exploring Adaptive Virtual Reality Systems Used in Interventions for Children With Autism Spectrum Disorder: Systematic Review. *Journal of Medical Internet Research*, 26. Available from https://doi.org/10.2196/57093.

Mandryk, R.L. and Atkins, M.S. (2007). A fuzzy physiological approach for continuously modeling emotion during interaction with play technologies. *International Journal of Human-Computer Studies*, 65 (4), 329–347. Available from https://doi.org/10.1016/j.ijhcs.2006.11.011.

Mehrabian, A. and Russell, J.A. (1974). *An approach to environmental psychology.* Cambridge, MA, US: The MIT Press. Available from https://psycnet.apa.org/record/1974-22049-000.

Mi, Q. and Gao, T. (2025). Engagement-Oriented Dynamic Difficulty Adjustment. *Applied Sciences*, 15 (10), 5610. Available from https://doi.org/10.3390/app15105610.

Mortazavi, F. (2024). Dynamic difficulty adjustment approaches in video games: a systematic literature review. *Multimedia Tools and Applications*.

Pfau, J. et al. (2020). Dungeons & Replicants: Automated Game Balancing via Deep Player Behavior Modeling. *2020 IEEE Conference on Games (CoG)*. August 2020. Osaka, Japan: IEEE, 431–438. Available from https://doi.org/10.1109/CoG47356.2020.9231958 [Accessed 7 January 2026].

Pianini, D. and Kalogeraki, V. (2023). Foreword: ACSOS 2021 Special Issue. *ACM Transactions on Autonomous and Adaptive Systems*, 18 (3), 1–2. Available from https://doi.org/10.1145/3612929.

Rodríguez, I., Puig, A. and Rodríguez, À. (2022). Towards Adaptive Gamification: A Method

Using Dynamic Player Profile and a Case Study. *Applied Sciences*, 12 (1), 486. Available from https://doi.org/10.3390/app12010486.

Rupp, F. and Eckert, K. (2024). G-PCGRL: Procedural Graph Data Generation via Reinforcement Learning. *2024 IEEE Conference on Games (CoG)*. 5 August 2024. Milan, Italy: IEEE, 1–8. Available from https://doi.org/10.1109/CoG60054.2024.10645633 [Accessed 23 February 2026].

Sakyev, T. (2025). Game Difficulty Balancing: Adaptive Difficulty and Its Effect on Player Experience. Available from https://doi.org/10.20944/preprints202511.2251.v1 [Accessed 13 February 2026].

Schwalbe, G. and Finzel, B. (2021). A comprehensive taxonomy for explainable artificial intelligence: a systematic survey of surveys on methods and concepts. *Data Mining and Knowledge Discovery*, 1–59. Available from https://doi.org/10.1007/s10618-022-00867-8.

Shu, T., Liu, J. and Yannakakis, G.N. (2021). Experience-Driven PCG via Reinforcement Learning: A Super Mario Bros Study. *2021 IEEE Conference on Games (CoG)*. 17 August 2021. Copenhagen, Denmark: IEEE, 1–9. Available from https://doi.org/10.1109/CoG52621.2021.9619124 [Accessed 23 February 2026].

Souza, C.H.R., Berretta, L.O. and Carvalho, S.T. (2025). Player Modeling and Large Language Models: An Interaction-Based Approach.

Summerville, A. et al. (2018). Procedural Content Generation via Machine Learning (PCGML). *IEEE Transactions on Games*, 10 (3), 257–270. Available from https://doi.org/10.1109/TG.2018.2846639.

Talpur, N. et al. (2022). Deep Neuro-Fuzzy System application trends, challenges, and future perspectives: a systematic survey. *Artificial Intelligence Review*, 56, 865–913. Available from https://doi.org/10.1007/s10462-022-10188-3.

Verheyen, L. et al. (2023). Neuro-Symbolic Procedural Semantics for Reasoning-Intensive Visual Dialogue Tasks. In: Gal, K. Nowé, A. Nalepa, G.J. et al. (eds). *Frontiers in Artificial Intelligence and Applications*. IOS Press. Available from

https://doi.org/10.3233/FAIA230544 [Accessed 1 November 2025].

Wang, L.-X. and Mendel, J.M. (1992). Generating fuzzy rules by learning from examples. *IEEE Transactions on Systems, Man, and Cybernetics*, 22 (6), 1414–1427. Available from https://doi.org/10.1109/21.199466.

Wheat, D. et al. (2016). Modeling Perceived Difficulty in Game Levels.

Wilson, R.C. et al. (2019). The Eighty Five Percent Rule for optimal learning. *Nature Communications*, 10 (1), 4646. Available from https://doi.org/10.1038/s41467-019-12552-4.

Xue, S. et al. (2017). Dynamic Difficulty Adjustment for Maximized Engagement in Digital Games. *Proceedings of the 26th International Conference on World Wide Web Companion - WWW '17 Companion*. 2017. Perth, Australia: ACM Press, 465–471. Available from https://doi.org/10.1145/3041021.3054170 [Accessed 7 February 2026].

Yannakakis, G.N. and Togelius, J. (2011). Experience-Driven Procedural Content Generation. *IEEE Transactions on Affective Computing*, 2 (3), 147–161. Available from https://doi.org/10.1109/T-AFFC.2011.6.

Yannakakis, G.N. and Togelius, J. (2018). Artificial Intelligence and Games. *Artificial Intelligence and Games*. Cham: Springer International Publishing, 91–150. Available from https://doi.org/10.1007/978-3-319-63519-4_3 [Accessed 1 November 2025].

Yu, X. (2024). Adaptive Map Generation Using Player Behavioral Data.

Zhang, Y. and Goh, W.-B. (2021). Personalized task difficulty adaptation based on reinforcement learning. *User Modeling and User-Adapted Interaction*, 31 (4), 753–784. Available from https://doi.org/10.1007/s11257-021-09292-w.

Zhang, Z. et al. (2025). Predicting Quality of Video Gaming Experience Using Global-Scale Telemetry Data and Federated Learning. Available from https://doi.org/10.48550/arXiv.2412.08950 [Accessed 7 February 2026].

BIBLIOGRAPHY

The following works informed the broader research context of this dissertation and are recommended for further reading on adaptive game systems, neuro-fuzzy AI, and telemetry-driven player modelling:

Leong, W.Y. (2025). Autonomous Game Balancing: AI-Driven Dynamic Difficulty Adjustment and Fairness Metrics.

Yannakakis, G.N. and Togelius, J. (2018). Artificial Intelligence and Games. *Artificial Intelligence and Games*. Cham: Springer International Publishing, 91–150. Available from https://doi.org/10.1007/978-3-319-63519-4_3 [Accessed 1 November 2025].

Summerville, A. et al. (2018). Procedural Content Generation via Machine Learning (PCGML). *IEEE Transactions on Games*, 10 (3), 257–270. Available from https://doi.org/10.1109/TG.2018.2846639.

Talpur, N. et al. (2022). Deep Neuro-Fuzzy System application trends, challenges, and future perspectives: a systematic survey. *Artificial Intelligence Review*, 56, 865–913. Available from https://doi.org/10.1007/s10462-022-10188-3.

Mortazavi, F. (2024). Dynamic difficulty adjustment approaches in video games: a systematic literature review. *Multimedia Tools and Applications*.

Xue, S. et al. (2017). Dynamic Difficulty Adjustment for Maximized Engagement in Digital Games. *Proceedings of the 26th International Conference on World Wide Web Companion - WWW '17 Companion*. 2017. Perth, Australia: ACM Press, 465–471. Available from https://doi.org/10.1145/3041021.3054170 [Accessed 7 February 2026].

APPENDICES

Appendix A - Supplementary Research Documentation

A.1. Literature Summary

The table summarises the existing work reviewed in *Section 1.5* of *Chapter 01*, including individual contributions and identified limitations.

Citation	Summary	Limitation	Contribution
Adaptive Map Generation Using Player Behavioral Data (Yu, 2024)	Proposed adaptive map generation using player telemetry to personalise Infinite Mario Bros levels.	Static PCG produces generic content and fails to accommodate diverse player skills and preferences.	Showed that telemetry-driven adaptive PCG improves replayability over static generation and highlights the need for personalisation.
Dynamic difficulty adjustment approaches in video games: a systematic literature review (Mortazavi, 2024)	Conducted a systematic review of 85 DDA papers, categorising rule-based and data-driven approaches.	Rule-based DDA is deterministic, limited in expressiveness, and unreliable in noisy environments. Threshold heuristics lack nuance.	Built a taxonomy of DDA methods and demonstrated the growing shift towards machine-learning-based personalisation.
Deep learning for procedural content generation (Liu et al., 2021)	Surveyed deep learning methods for PCG including GANs, LSTMs, and autoencoders for level and asset generation.	DL-PCG faces data scarcity, functional constraints (unplayable content), and interpretability issues from black-box models.	Formalised the DLPCG field and categorised deep generative approaches for content creation.

Adversarial Reinforcement Learning for Procedural Content Generation (Gisslen et al., 2021)	Introduced Adversarial Reinforcement Learning for PCG using generator and solver agents to co-evolve challenging levels.	Computationally expensive and requires careful reward shaping. May converge to narrow or unplayable solutions.	Showed improved generalisation and prevention of level memorisation through adversarial training.
Dungeons & Replicants: Automated Game Balancing via Deep Player Behavior Modeling. (Pfau et al., 2020)	Applied deep player behaviour modelling to cluster and replicate player behaviour for automated game balancing.	Deep neural clustering acts as a black box, limiting interpretability of behavioural grouping decisions.	Automated playtesting via DL-based replicants, improving balance detection over heuristic approaches.
Induction of Emotional States in Educational Video Games Through a Fuzzy Control System. (Lara-Alvarez et al., 2021)	Developed a fuzzy control system to adjust difficulty and aesthetics based on performance and emotional state in educational games.	Requires expert-designed membership functions and may rely on intrusive sensor inputs.	Demonstrated the effectiveness of fuzzy logic in handling uncertainty and improving emotional engagement.
Personalized task difficulty adaptation based on reinforcement learning. (Zhang and Goh, 2021)	Proposed Bootstrapped Policy Gradient combining offline clustering with online reinforcement learning for real-time difficulty adaptation.	Traditional RL adaptation is slow and pure rule-based systems are fast but inaccurate. Latency remains a constraint.	Showed that hybrid approaches can improve responsiveness and reduce cold-start challenges in short-session environments.

Table 35: Summary of existing work and limitations

A.2. Research Objectives Mapping

The table below presents the full research objectives (R01–R14) defined in *Section 1.11* of *Chapter 01*, with descriptions, learning outcome mappings, and research question alignments.

Objectives	**Research Objective**	**Description**	**LOs Mapped**	**RQs Mapped**
Problem Identification	R01: Identify limitations of immediate adaptive PCG systems	Analyse cold-start bias, instability, and agency violations in current adaptive frameworks.	L02	RQ1
	R02: Define a calibration-first adaptive architecture	Establish theoretical basis for separating calibration from adaptation.	L02	RQ1
Literature Review	R03: Conduct a comprehensive literature review on DDA, APCG, and telemetry-driven modelling	Critically evaluate static PCG, rule-based DDA, RL-PCG, and neuro-fuzzy systems.	L01, L04	RQ1, RQ4
	R04: Investigate soft clustering and neuro-symbolic reasoning models	Identify interpretable modelling strategies for behavioural fluidity.	L01, L04	RQ3, RQ4
Requirement Elicitation	R05: Identify system-level requirements for real-time adaptive control	Define behavioural modelling, latency, smoothing, and bounded adaptation requirements.	L03	RQ2, RQ4
System Design	R06: Design the three-phase adaptive architecture	Develop a structured pipeline ensuring stability and forward-only adaptation.	L01, L03	All RQs
	R07: Design telemetry feature engineering and	Define 30-second windowing, intent-based	L01	RQ2, RQ3

	temporal modelling strategy	features, and delta modelling.		
Implementation	R08: Implement behavioural clustering with soft membership modelling	Develop K-Means centroids with inverse distance weighting.	L01	RQ3
	R09: Implement neuro-fuzzy surrogate reasoning layer	Train ANFIS and approximate via MLP surrogate for runtime efficiency.	L01	RQ4
	R10: Integrate backend-mediated adaptive inference with Unreal Engine	Implement decoupled pipeline via backend communication to ensure latency stability.	L01	RQ2, RQ4
Testing & Evaluation	R11: Evaluate adaptation stability and boundedness	Measure oscillation absence, multiplier constraints, and behavioural smoothness.	L01, L04	RQ2, RQ3
	R12: Evaluate performance impact and latency	Measure FPS stability and backend inference latency.	L01	RQ2
	R13: Evaluate behavioural interpretability	Assess transparency of behavioural proportions and adaptive decisions.	L04	RQ4
Documentation & Dissemination	R14: Document and disseminate findings	Prepare publishable research outputs aligned with neuro-symbolic adaptive PCG.	L04	All RQs

Table 36: Research objectives, descriptions, and mapping

A.3. In-Scope Items

This table lists the complete set of in-scope items defined in *Section 1.12.1* of *Chapter 01*.

In-Scope Items
Design and implementation of a calibration-first adaptive procedural framework
Development of a controlled calibration phase for baseline behavioural stabilisation
Telemetry collection using fixed 30-second windows under non-adaptive conditions
Feature engineering based on behavioural intent metrics: Combat, Collection, and Exploration archetypes
Soft behavioural clustering using K-Means with inverse distance weighting for archetype proportions
Integration of temporal delta modelling to capture behavioural momentum between windows
Offline training of an ANFIS reasoning surface for adaptive control modelling
Development and deployment of an MLP surrogate (6-16-8-1) to approximate MLP surrogate inference at runtime
Design of a backend-mediated adaptive inference pipeline decoupled from the game engine
Real-time procedural parameter modulation using bounded multiplier outputs within [0.6, 1.4]
Forward-only adaptation constraints to preserve player agency and perceptual continuity
Performance evaluation: latency measurement, FPS impact, multiplier boundedness, and oscillation testing

Table 37: In-scope items

A.4. Out-Scope Items

The table below lists the complete set of out-of-scope items defined in *Section 1.12.2* of *Chapter 01*.

Out-of-Scope Items
Direct runtime execution of MLP surrogate inference within the Unreal Engine environment
Emotion recognition using biometric sensors or affective computing devices
Use of reinforcement learning or deep generative adversarial networks for content generation
Multiplayer or competitive game adaptation scenarios

Cross-genre generalisation beyond the implemented single-player prototype
Full-scale AAA production-level optimisation and scalability benchmarking
Integration of large language models for behavioural inference or narrative generation
Formal runtime model verification using symbolic verification frameworks

Table 38: Out-of-scope items

A.5. Hardware Requirements

The table below details the physical hardware configuration used during development and offline model training defined in Section 1.13 of *Chapter 01*.

Category	**Specification**	**Purpose / Justification**
Development Workstation	Intel Core i7 14th gen gen or above, 16 GB RAM minimum, NVIDIA RTX 4060 or above	Primary development, model training, and Unreal Engine 5.7 prototype testing.
Secondary Test Platform	Intel Core i5 12th gen, 16 GB RAM, NVIDIA RTX 3050	Secondary performance and regression testing under lower-spec conditions.
External / Cloud Storage	1 TB HDD + cloud backup (Google Drive / GitHub/ Diversion)	Version control, backup, and sharing of model weights, telemetry logs, and thesis documents.

Table 39: Hardware requirements

A.6. Software Requirements

The table below lists all software tools and frameworks used across the AURA development pipeline, with version information and justification for each selection defined in Section 1.13 of *Chapter 01*.

Software / Tool	**Version**	**Category**	**Justification for Selection**
Python	3.x	Language	Primary language for all data processing, clustering, and model training notebooks, selected for its extensive scientific computing ecosystem (NumPy, Pandas, scikit-

			learn)
Jupyter Notebook	6.x	IDE	Iterative, cell-by-cell execution of the 8-stage model construction pipeline with inline visualisation of clustering and training metrics
scikit-learn	1.x	ML Library	Provides K-Means clustering, MinMaxScaler, and grid search utilities used across Phases 1 and 2 of the AURA pipelines
NumPy / Pandas	Latest	Data Processing	Telemetry window aggregation, feature engineering, and dataset filtering across 3,240 30-second windows derived from 45 participant sessions
Node.js	22 LTS	Runtime	Server-side runtime for the AURA inference backend and LTS version selected for production stability on the Render cloud deployment
TypeScript	5.x	Language	Static typing for the backend codebase reduces runtime errors in the inference pipeline and improves long-term maintainability
Next.js	16.x.x	Frontend Framework	React-based framework for the AURA monitoring dashboard selected for Vercel-native deployment and built-in API routing capabilities
Unreal Engine	5.7	Game Engine	Engine used to develop CollectGame, the telemetry collection and adaptive gameplay environment across all three AURA phases. PCG parameter adaptation is implemented in Phase 3, with the bounded multiplier applied to eleven parameters spanning Combat, Exploration, Collection, and Global categories.
Visual Studio Code	Latest	IDE	Primary editor for TypeScript backend and Next.js dashboard development integrated Git support and ESLint extensions streamlined the development workflow
GitHub	N/A	Version Control	Version control host for all three AURA repositories (CollectGame.CalibrationAnalysis, CollectGame.Model,

			CollectGame.Telemetry) supports branch-based development and change tracking
Render	Free Tier	Cloud Platform	Hosts the Node.js inference backend and selected for zero-cost deployment and automatic HTTPS provisioning, enabling accessible remote evaluation
Vercel	Free Tier	Cloud Platform	Hosts the Next.js monitoring dashboard and selected for native Next.js integration and global edge CDN, ensuring sub-second dashboard load times
Google Forms	N/A	Data Collection	Administered participant consent forms and post-session questionnaires during the Phase 1 calibration study (N=7) and Phase 2 telemetry collection (69 participants registered. 64 with complete telemetry records and 45 sessions retained after applying the ≥20-minute duration filter)
Vitest	1.6.1 / 4.0.18	Testing	Unit testing framework for the Node.js backend and 43 tests across 7 suites validate the full inference pipeline. Two versions used across the Telemetry and ANFIS sub-systems
Microsoft Word 365	365	Documentation	Used to author all thesis chapters and the Software Requirements Specification in compliance with the IIT Final_Template.docx formatting standard

Table 40: Software requirements

Appendix B - Concept Map

Below figure demonstrates the concept map defined in *Section 2.2* of *Chapter 02*.

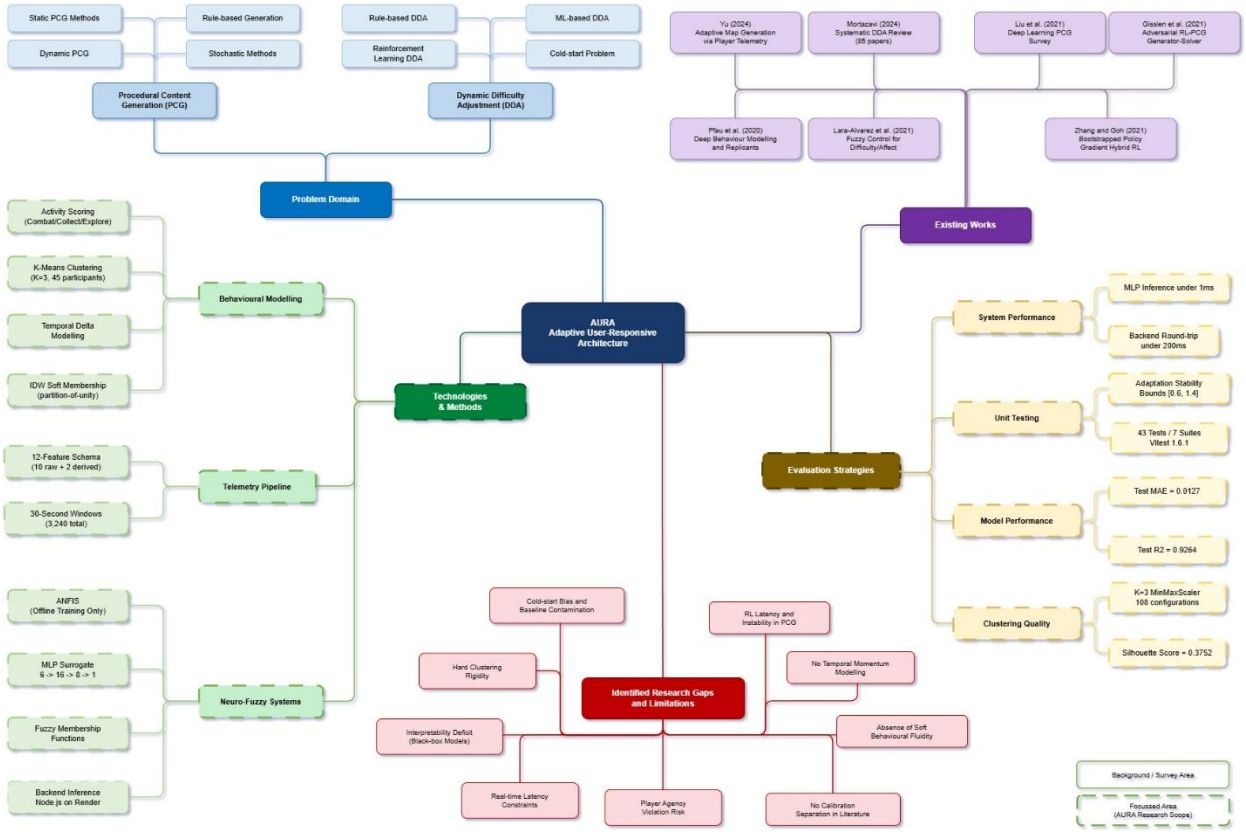


Figure 28: Concept Map (Self Composed)

Appendix C - Gantt Chart

Project schedule as referenced in *Section 3.4.2.1* in *Chapter 03*.

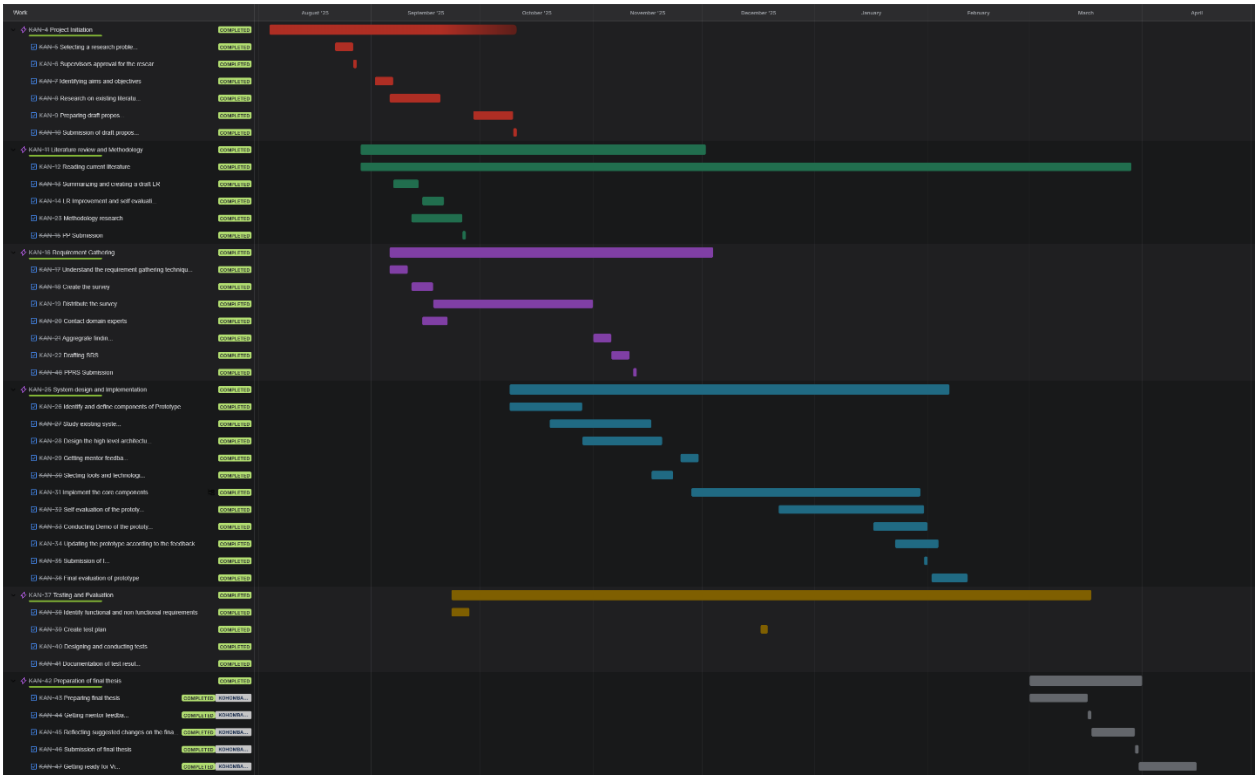
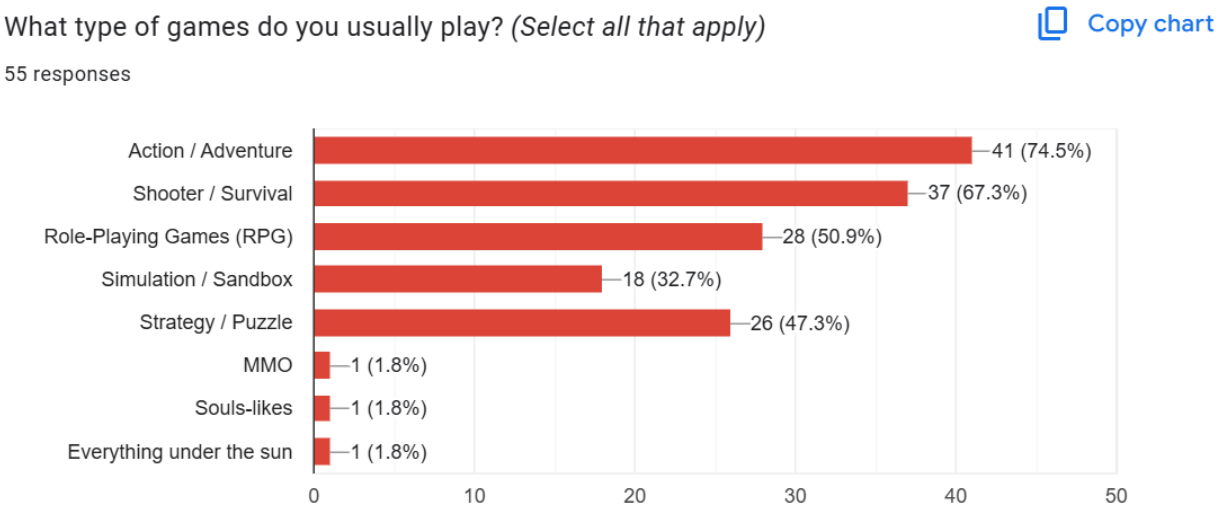


Figure 29: Gantt Chart (Self Composed)

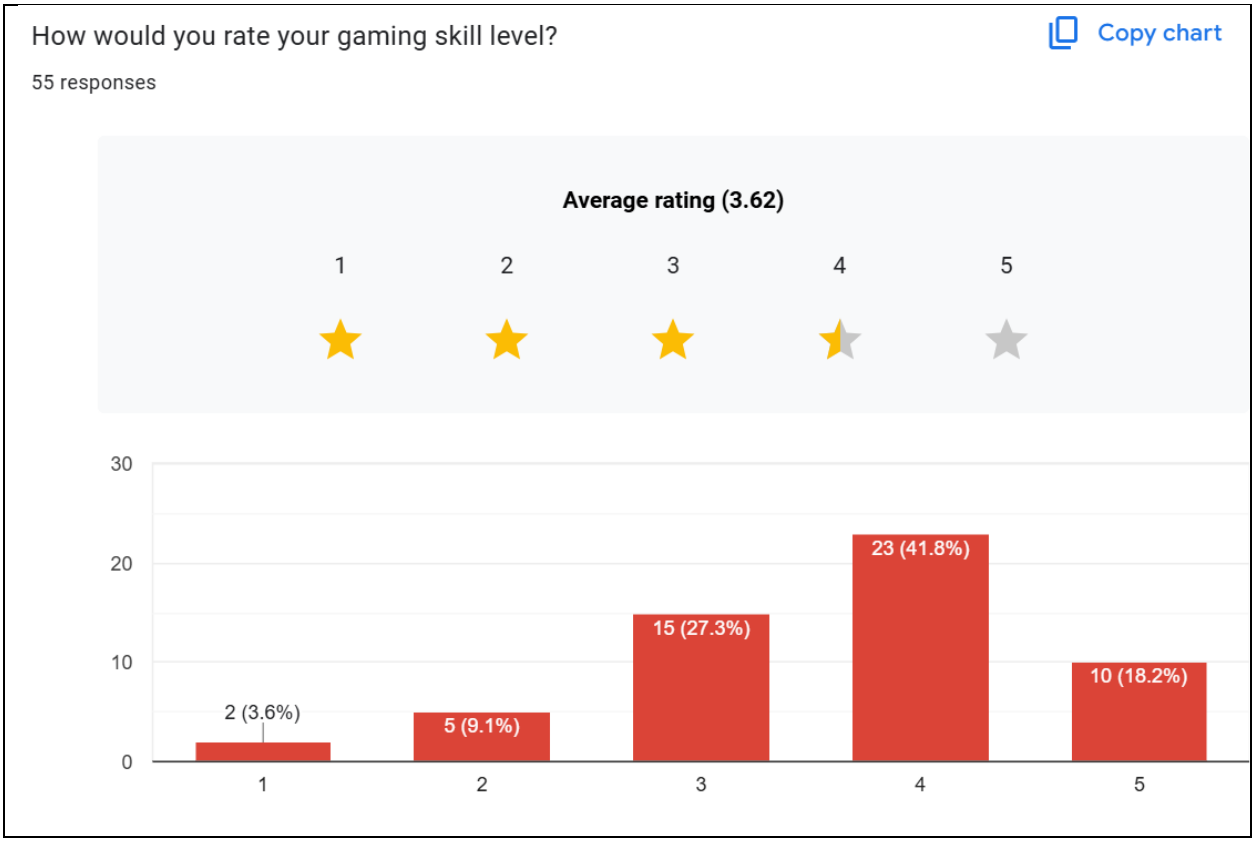
Appendix D - Additional Survey Findings

Supplementary survey results supporting the elicitation findings in *Section 4.5.2* in *Chapter 04*.

Id	2
Question	What type of games do you usually play?
Aim of the Question	To understand which genres, dominate the participants' gameplay patterns and contextualize adaptive design expectations.
Findings and Conclusion	
Action-Adventure and Shooter genres were the most common, followed by Role-Playing and Simulation games, indicating that adaptive systems should emphasize combat balance and exploration pacing.	



Id	3
Question	How would you rate your gaming skill level?
Aim of the Question	To classify players by experience level for behavioural clustering relevance.
Findings and Conclusion	
Most respondents rated their skill as moderate to high, validating the need for adaptive systems that dynamically balance difficulty to maintain engagement for mixed-skill audiences.	

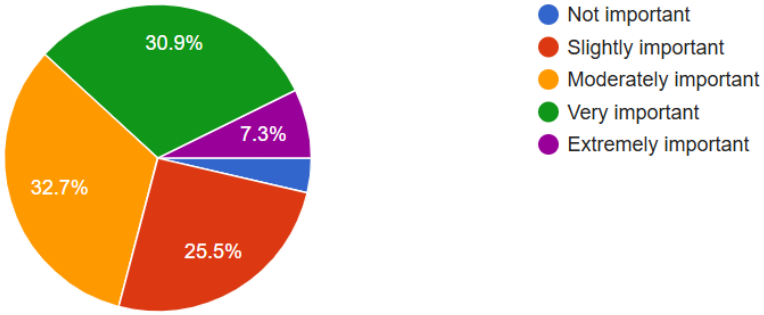


Id	4
Question	How important is it for a game to adjust its difficulty based on your performance?
Aim of the Question	To measure user demand for adaptive gameplay features.
Findings and Conclusion	
Nearly all participants rated adaptivity as important or very important, reinforcing the core objective of implementing real-time Neuro-Fuzzy adaptation in the project.	

How important is it for a game to adjust its difficulty based on your skill or performance?

Copy chart

55 responses

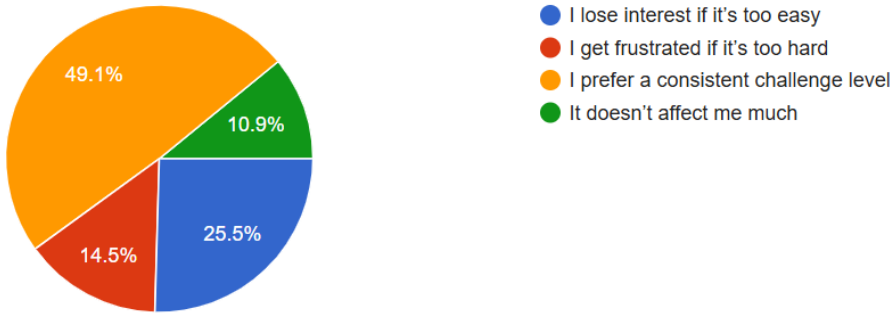


Id	5
Question	When a game becomes too easy or too hard, how does it affect your engagement?
Aim of the Question	To understand how imbalance affects player immersion.
Findings and Conclusion	
Most respondents stated that rapid shifts in difficulty reduce immersion, highlighting the necessity of smooth transitional control in the fuzzy inference stage.	

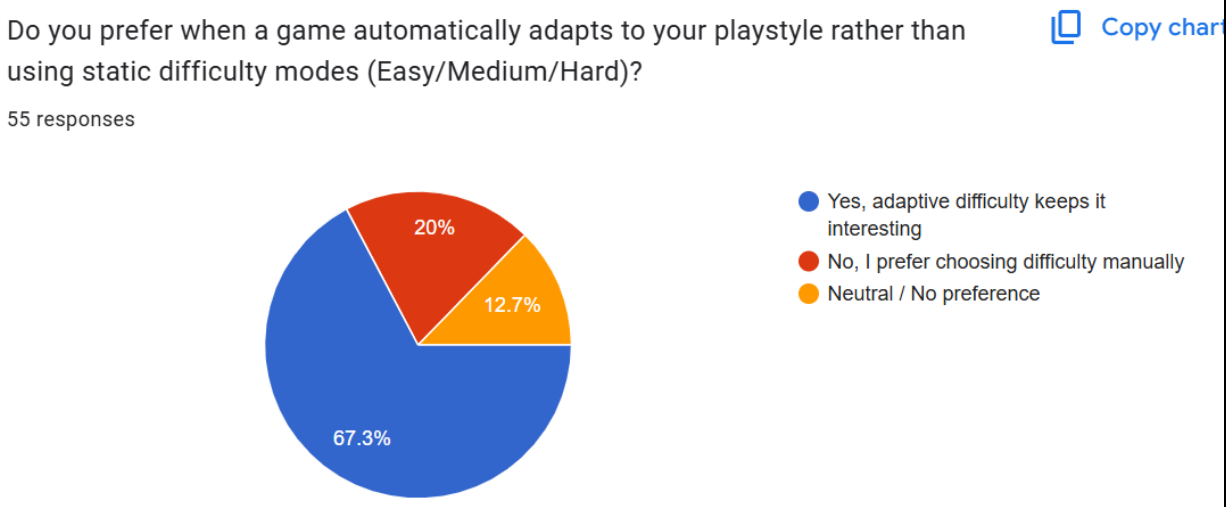
When a game becomes too easy or too hard, how does it affect your engagement?

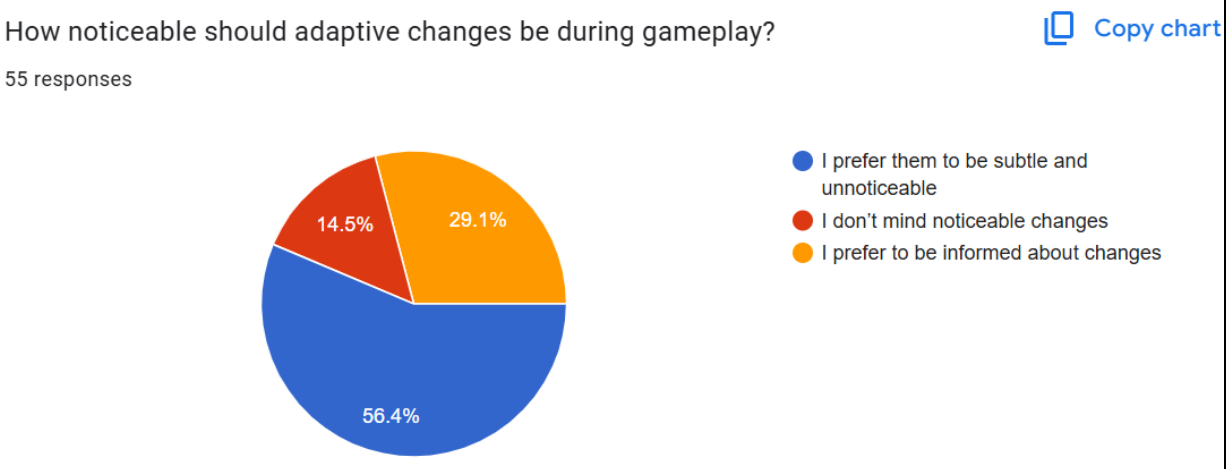
Copy chart

55 responses

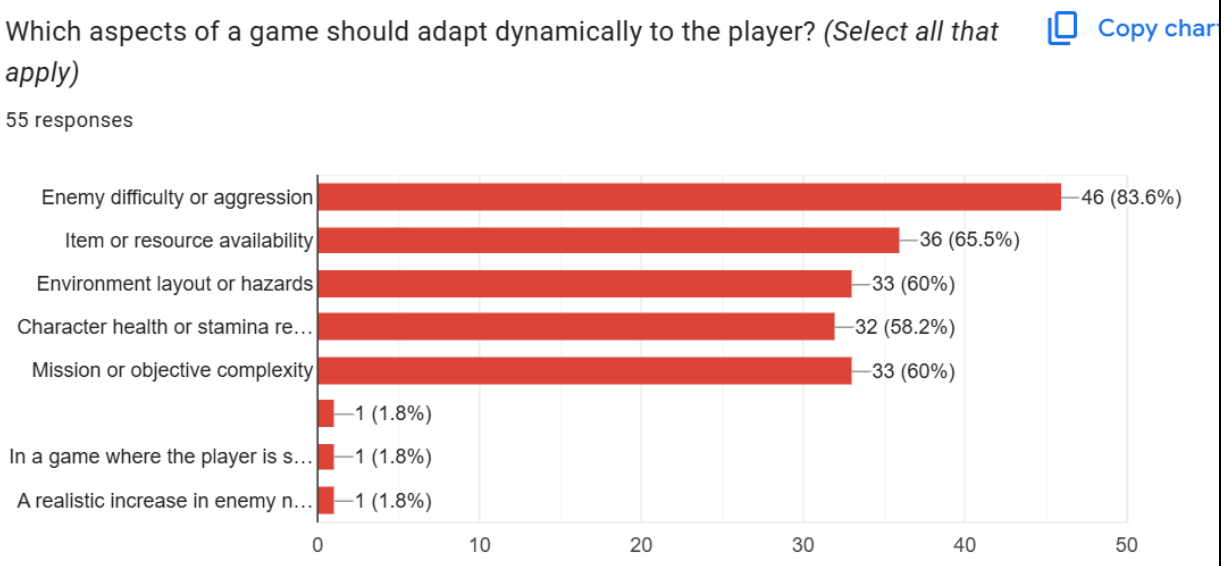


Id	6
Question	Do you prefer when a game automatically adapts to your playstyle

	rather than using static difficulty modes
Aim of the Question	To identify acceptance of automated adaptivity.
Findings and Conclusion	
Over 80 % of respondents preferred automated adjustment, validating the Neuro-Fuzzy self-adapting design. 	

Id	7
Question	How noticeable should adaptive changes be during gameplay?
Aim of the Question	To determine user tolerance for visible adaptation.
Findings and Conclusion	
Most participants preferred subtle adjustments, suggesting that adaptation should occur transparently without breaking immersion. 	

Id	8
Question	Which aspects of a game should adapt dynamically to the player?
Aim of the Question	To identify preferred parameters for dynamic adjustment.
Findings and Conclusion	
Enemy aggression, item availability, and environmental density were the highest priorities, supporting the framework's real-time control over spawn and resource variables.	



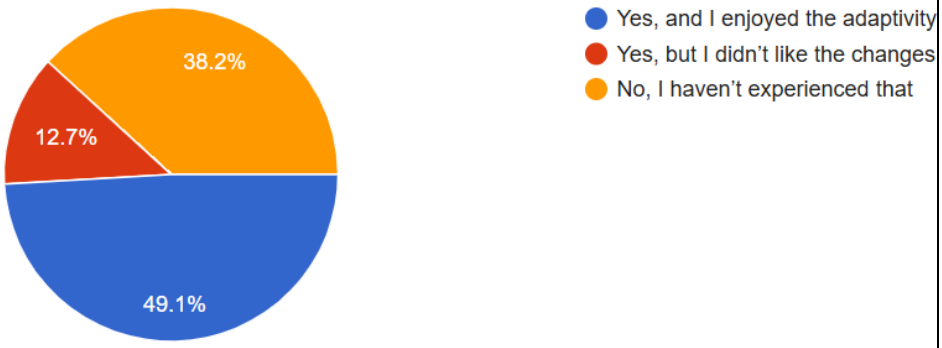
Id	9
Question	Have you played a game that adapted its challenge to your performance?
Aim of the Question	To examine prior exposure to adaptive systems.
Findings and Conclusion	
Over 60 % had experienced adaptive gameplay. Mixed feedback indicated both enjoyment and frustration when systems over-compensated, demonstrating the importance of balanced fuzzy rule calibration.	

Have you played a game that adapted its challenge to your performance?

e.g.,

1. **Resident Evil 4**: Enemy aggression changes based on your skill.
2. **Left 4 Dead**: The AI Director spawns enemies/items dynamically.
3. **F.E.A.R.**: Enemies use advanced, adaptive flanking and cover tactics.
4. **God Hand**: A visible gauge increases enemy difficulty as you perform well.
5. **Racing Games**: Opponent speed 'rubber-bands' to keep races close.

55 responses



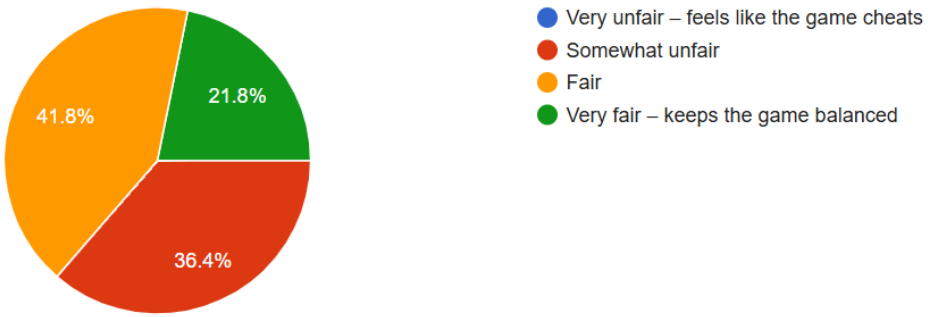
Id	10
Question	How fair do you consider adaptive difficulty systems in games?
Aim of the Question	To assess trust and perceived fairness in adaptive AI.
Findings and Conclusion	

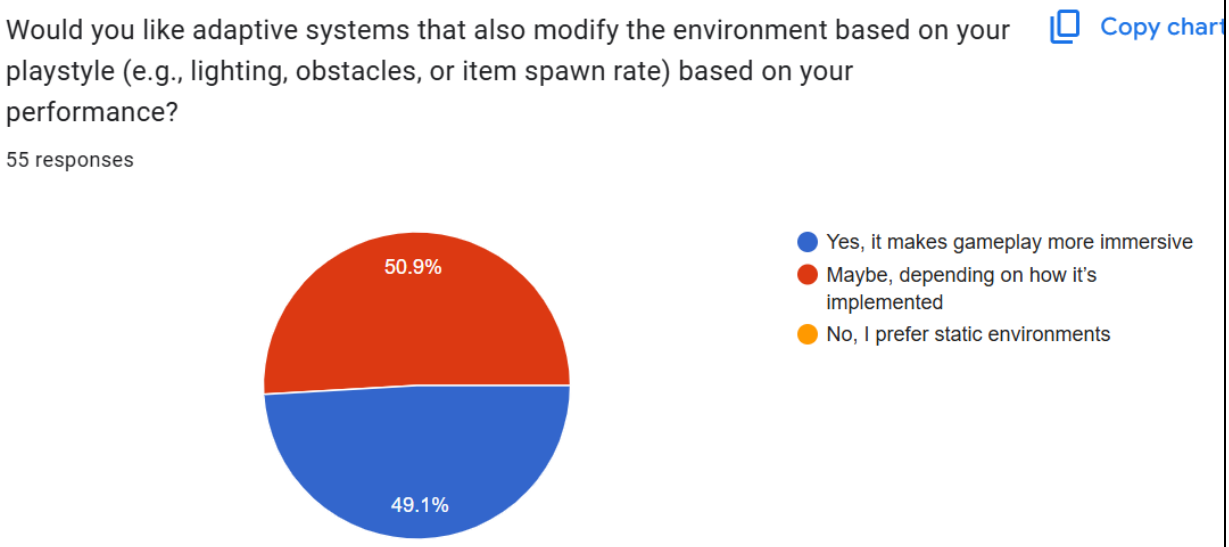
Most participants rated adaptive systems as fair or very fair, provided that adaptations occurred logically and consistently, aligning with explainable-AI design principles.

How fair do you consider adaptive difficulty systems in games?

Copy chart

55 responses



Id	11
Question	Would you like adaptive systems that also modify the environment based on your playstyle (e.g., lighting, obstacles, or item spawn rate)?
Aim of the Question	To evaluate acceptance of environment-based adaptation.
Findings and Conclusion	
A majority expressed strong interest in environmental modifications, validating the extension of Neuro-Fuzzy control to environmental parameters. 	

Id	12
Question	Any additional comments or suggestions about adaptive or intelligent game systems?
Aim of the Question	To collect qualitative insights and limitations from participants.
Findings and Conclusion	
Some respondents mentioned frustration with "rubberbanding" effects or abrupt AI reactions, reinforcing the importance of smooth fuzzy transitions and interpretability in adaptive systems.	

Appendix E - Expert Interview Evidence

This section includes evidence for expert interviews in *Section 4.5.3* of *Chapter 04*.

E.1. Interviewee Details

ID	Name	Designation	Organisation
EI-1	Ravindu Cooray	Head of Studio	RAM Studios Pvt Ltd
EI-2	Hasindu Ramanayake		

Table 41: Expert Interviewee Details

E.2. Interview Summaries

EI-1: Ravindu Cooray highlighted that the majority of commercial **Dynamic Difficulty Adjustment** implementations he had encountered in production responded to **skill-based outcome metrics** such as death count and score differential, rather than modelling the underlying intent of player behaviour. He noted that this approach fails to account for players who engage in off-archetype actions for strategic or survival reasons and expressed that a calibration-first behavioural pipeline of the kind proposed in AURA represented a more principled approach to **personalised adaptation**.

EI-2: Hasindu Ramanayake raised the intent-misclassification problem as a key concern with **hard archetype assignment** systems. He specifically cited the scenario in which a player whose primary motivation is resource collection is compelled to engage enemies for survival, causing the system to incorrectly register them as a combat-dominant archetype for the remainder of the session. He affirmed that **soft membership modelling** across multiple archetypes would more accurately reflect the fluid and contextual nature of real player behaviour.

E.3. Thematic Analysis

Code	Theme	Conclusion
Skill vs. Behaviour	Limitations of existing DDA systems	Existing Dynamic Difficulty Adjustment systems adapt based on player skill metrics such as death rate and score, rather than behavioural intent. Experts confirmed this as a structural gap that AURA directly addresses through

		telemetry-driven behavioural profiling.
Intent Misclassification	Hard archetype classification risk	Experts identified that hard archetype assignment fails when incidental actions contradict player intent for example, a collection-oriented player killing enemies for survival is misclassified as combat-dominant. This finding motivated the requirement for soft membership over rigid archetype classification.

Table 42: Findings from Expert Interviews

Appendix F - Additional Use Case Descriptions

"Extended use case descriptions supporting the primary use case (UC-01) presented in *Section 4.9* in *Chapter 04*.

Field	Content
Use Case ID	UC-A1
Use Case Name	Telemetry Data Collection
Primary Actor	Player
Supporting Actor	System (Telemetry Module)
Description	Collects and aggregates player gameplay metrics at fixed session intervals for submission to the adaptive inference service.
Preconditions	Game session active. telemetry collector initialized.
Postconditions	A normalized telemetry record/buffer is available for clustering.
Normal Flow	System monitors gameplay events during the session. At each interval boundary, the system aggregates accumulated metrics. The system transmits the aggregated behaviour summary to the backend service.
Alternative Flow	If aggregation fails or data is incomplete, the system retains the last valid summary until the next interval.
Priority	Must

Table 43: UC-A1 - Telemetry Data Collection

Field	Content
Use Case ID	UC-A2
Use Case Name	Player Behaviour Clustering
Primary Actor	System (K-Means Module)
Supporting	Telemetry Module

Actor	
Description	Analyses aggregated gameplay metrics to produce a continuous behavioural profile indicating the player's current tendency across Combat, Collection, and Exploration activity types.
Preconditions	Normalized telemetry records available (from UC-A1).
Postconditions	Cluster ID and normalized cluster-centroid metrics stored and forwarded to ANFIS (UC-A3).
Normal Flow	System receives aggregated behaviour metrics. System computes proportional activity scores across the three behavioural dimensions. System forwards the behavioural profile to the adaptive reasoning component.
Alternative Flow	If insufficient data is available, system retains the previous behavioural profile and logs the incident.
Priority	Must

Table 44: UC-A2 - Player Behaviour Clustering (K-Means)

Field	**Content**
Use Case ID	UC-A3
Use Case Name	Adaptive Inference
Primary Actor	System (backend inference service)
Supporting Actor	K-Means Module, Developer (monitoring)
Description	Applies the trained adaptive reasoning model to the current behavioural profile to produce a bounded adaptation multiplier for environment adjustment.
Preconditions	Valid cluster ID and telemetry metrics available (from UC-A2).
Postconditions	The system produces a set of continuous adaptation parameters and sends them to the Procedural Content Generation Controller (UC-A4). The fuzzy rule activations are saved for later explanation and developer review.
Normal Flow	System receives the behavioural profile. The adaptive reasoning model

	processes the profile and produces a bounded adaptation value. The system applies neutral-centring and safety bounds before forwarding the adaptation value to the PCG control component.
Alternative Flow	If inference fails, system uses the last valid adaptation value and logs the failure.
Priority	Must

Table 45: UC-A3 - Fuzzy Logic Reasoning (ANFIS Adaptation)

Field	**Content**
Use Case ID	UC-A4
Use Case Name	Procedural Content Generation Control
Primary Actor	System (PCG Controller)
Supporting Actor	Unreal Engine Environment
Description	Applies the recommended adaptation value to procedural content generation parameters, affecting only future content generation to preserve current gameplay continuity.
Preconditions	Adaptation parameters available from UC-A3.
Postconditions	In-game PCG parameters updated changes take effect with minimal disruption.
Normal Flow	System receives adaptation value. System applies the value progressively to environment generation parameters. Changes take effect at the next procedural generation cycle.
Alternative Flow	If engine call fails or latency high, system reduces adaptation magnitude or reverts to last safe parameters, logs incident.
Priority	Must

Table 46: UC-A4 - Procedural Content Generation Control (PCG Configuration)

Field	**Content**
Use Case ID	UC-A5

Use Case Name	Adaptive Environment Application
Primary Actor	System
Supporting Actor	Unreal Engine runtime
Description	Apply the recommended adaptation parameters to procedural content generation settings, targeting only future content generation to preserve continuity of the current gameplay experience.
Preconditions	Adaptation parameters received from the inference service. Game session active.
Postconditions	Environment generation parameters updated. Changes take effect at the next procedural generation cycle. Previously generated content unchanged.
Normal Flow	System receives adaptation parameters. System applies a progressive transition to the new parameter values. Update parameters influence the next procedural generation cycle. System logs the adaptation event.
Alternative Flow	If the adaptation parameters fall outside safe operating bounds, the system retains the current parameters and continues gameplay without interruption.
Priority	Should

Table 47: UC-A5 - Environment Application and Stability Management

Field	**Content**
Use Case ID	UC-A6
Use Case Name	Analytics Monitoring
Primary Actor	Developer
Supporting Actor	System
Description	The developer accesses the live analytics dashboard to observe the current adaptation state, behavioural profile history, and session pipeline outputs

	during or after a gameplay session.
Preconditions	Developer has access to the analytics dashboard. An active or completed session exists.
Postconditions	Developer has observed the behavioural breakdown, multiplier trend, and pipeline validation status for the session.
Normal Flow	The developer accesses the live analytics dashboard to observe the current adaptation state, behavioural profile history, and session pipeline outputs during or after a gameplay session.
Alternative Flow	If the session being monitored has concluded before the developer connects, the dashboard displays the most recently recorded session state and presents a notification indicating that no active session is currently in progress.
Priority	Should

Table 48: UC-A6 - Developer Configuration and Monitoring

Field	**Content**
Use Case ID	UC-A7
Use Case Name	Session Data Export
Primary Actor	Developer
Supporting Actor	System
Description	Compiles and exports recorded session data including behaviour summaries, adaptation decisions, and pipeline validation records for offline research analysis.
Preconditions	A session has ended or a periodic export has been triggered.
Postconditions	Session data exported and available for offline analysis. Original session records remain intact.
Normal Flow	Upon session end, the system aggregates recorded behaviour summaries, adaptation events, and pipeline outputs. The system exports the data in a structured format. The developer retrieves the export for analysis.

Alternative Flow	If exports fail, the system retains the session data locally until the next successful export attempt.
Priority	Must

Table 49: UC-A7 - Data Logging and Evaluation

Appendix G - Detailed Design

G.1. Class-Level Design of the Backend Inference Pipeline

The figure summarises the existing work reviewed in *Section 6.4.3* of *Chapter 06*. The class-level design diagram presented in *Figure 30*: Backend Inference Pipeline Class-Level Design (Self Composed) illustrates the structural composition and dependency relationships of the backend behavioural inference module.

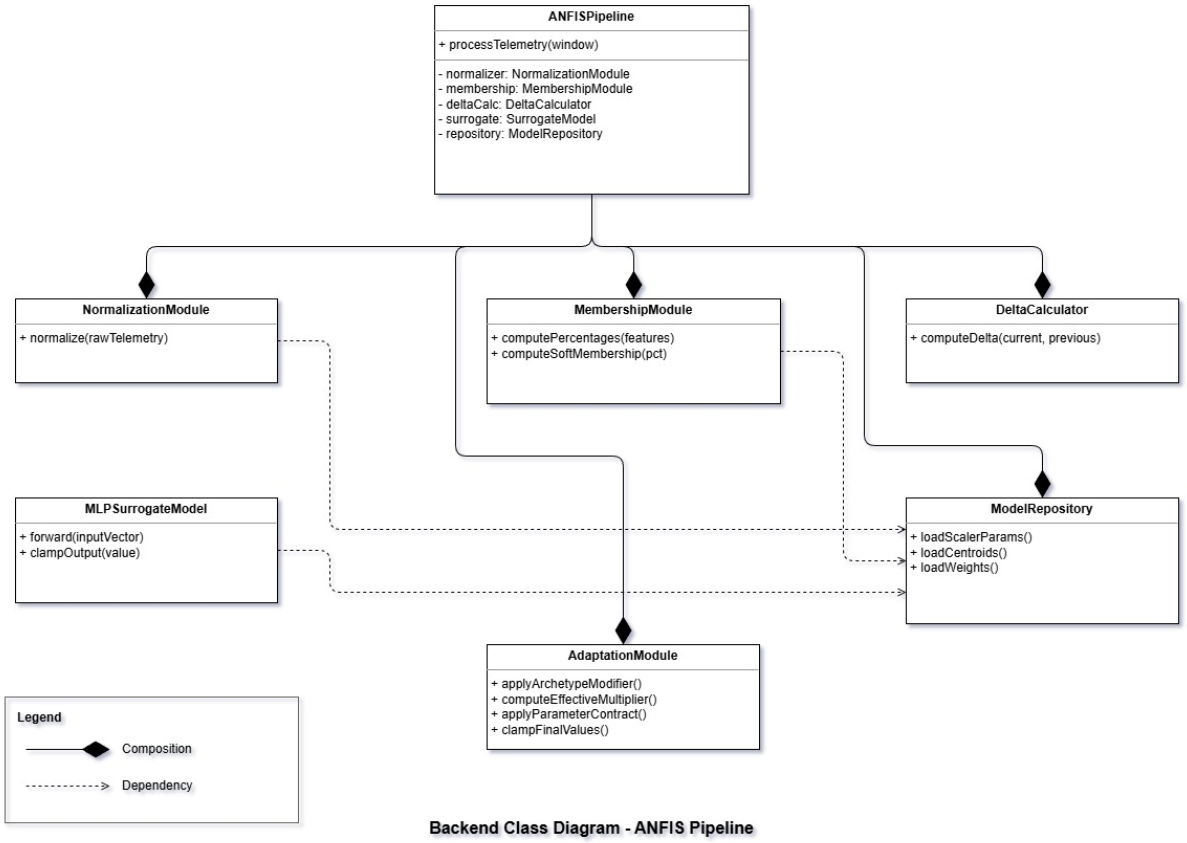


Figure 30: Backend Inference Pipeline Class-Level Design (Self Composed)

As shown in Figure 30: Backend Inference Pipeline Class-Level Design (Self Composed), the class-level design enforces single-responsibility boundaries across all inference components, supporting modularity and testability within the backend service. Each component exposes a clearly defined interface and holds no cross-component state, enabling independent unit testing and future extension of individual pipeline stages.

G.2. Unreal-Side Adaptation Control Activity Diagram

The figure summarises the existing work reviewed in *Section 6.4.6* of *Chapter 06*. The Unreal-side adaptation control activity diagram presented in Figure 31: Unreal-Side Adaptation Control Activity Flow (Self Composed) illustrates the operations applied within the game client upon receipt of a bounded multiplier value from the backend inference service.

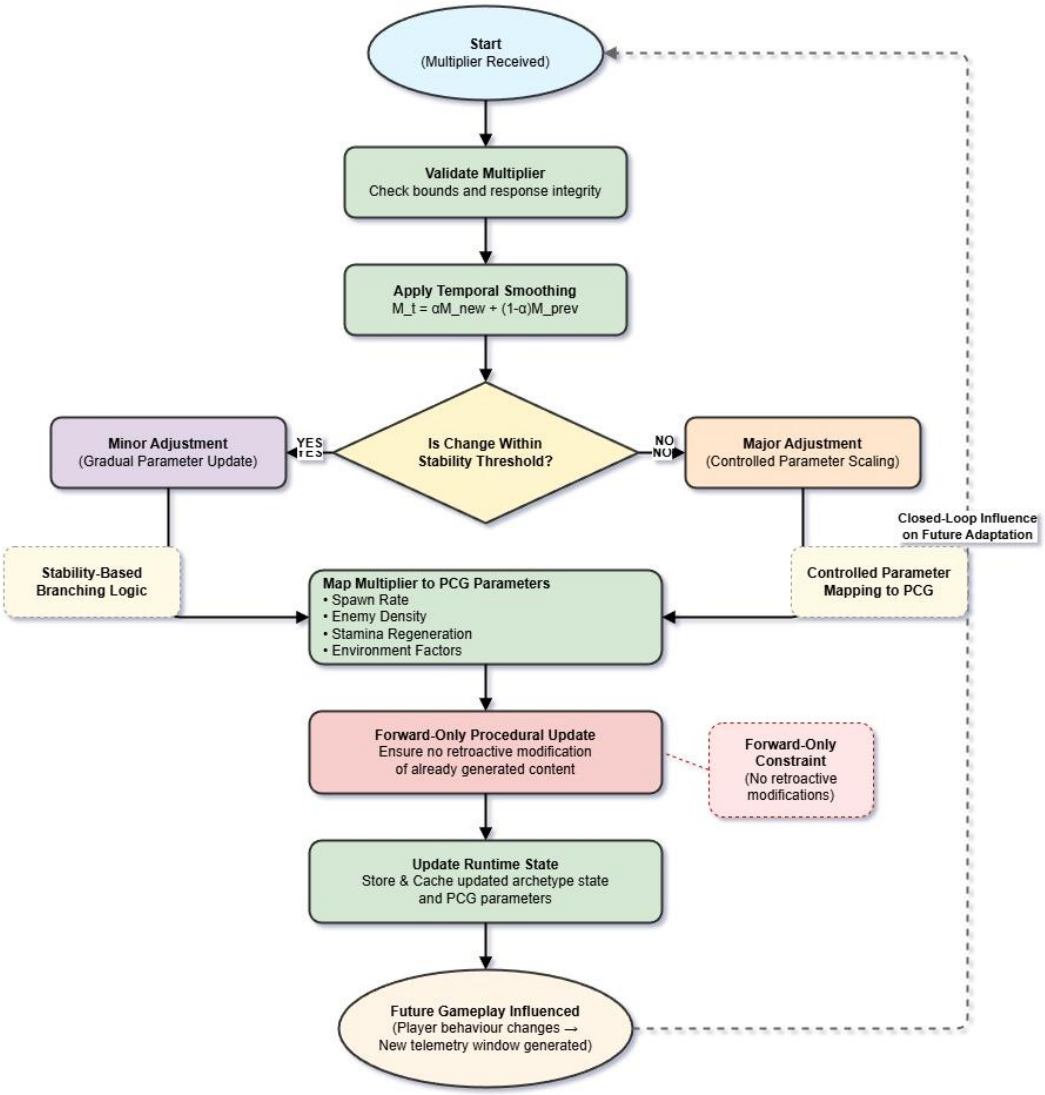


Figure 31: Unreal-Side Adaptation Control Activity Flow (Self Composed)

G.3. AURA end-to-end system process activity diagram

The figure summarises the existing work reviewed in *Section 6.4.7* of *Chapter 06*. The figure

complete activity flow of the AURA adaptive control loop, illustrating the interaction between the Unreal Engine client and the backend inference service.

The process begins with a backend availability check. If the backend is unreachable, the system enters degraded-mode operation and applies a neutral baseline configuration. Upon successful communication, gameplay telemetry is aggregated over fixed time windows and dispatched to the inference service. The returned adaptation multiplier is then applied to player parameters, while PCG updates are deferred to subsequent generation cycles. This loop repeats continuously throughout the session.

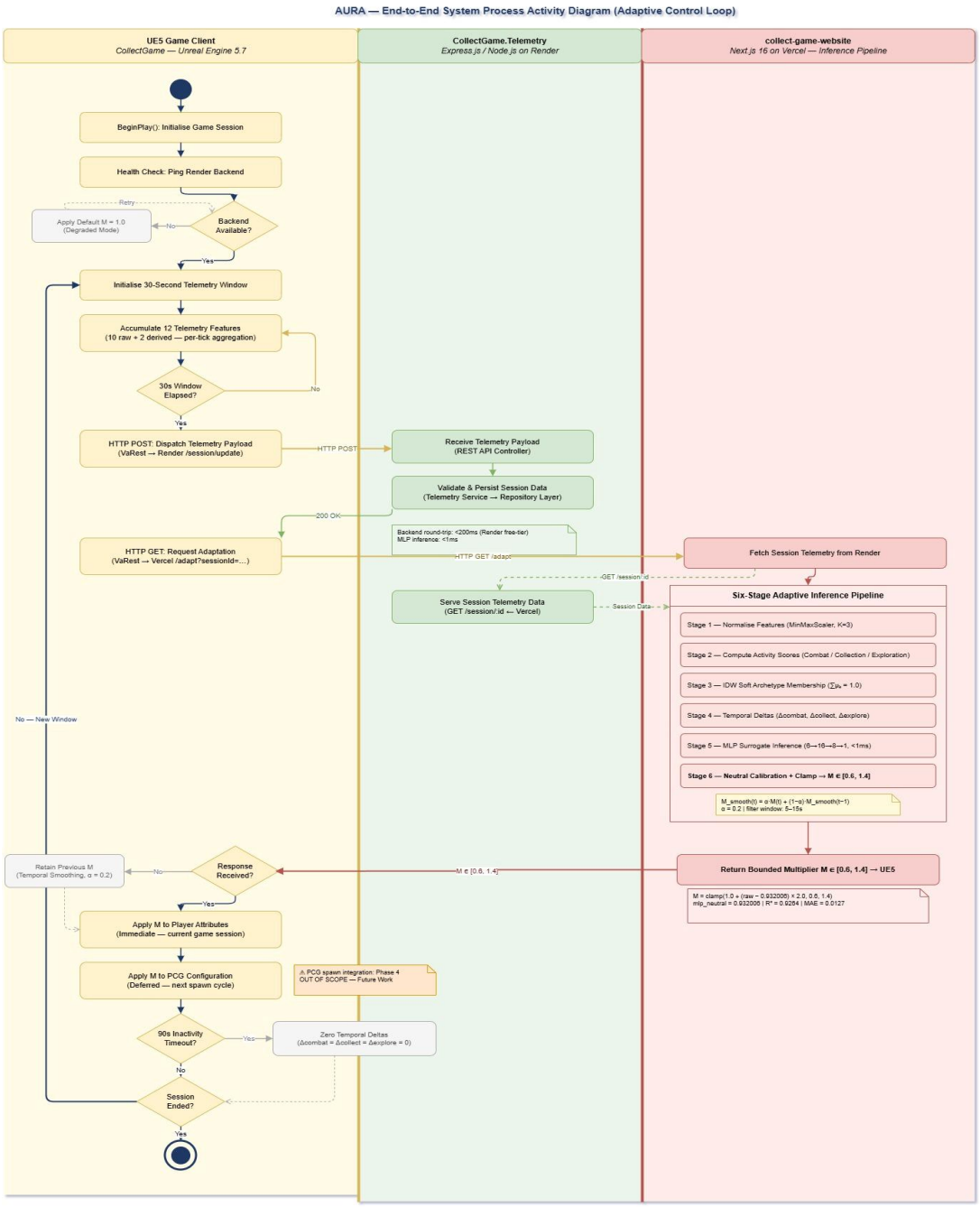
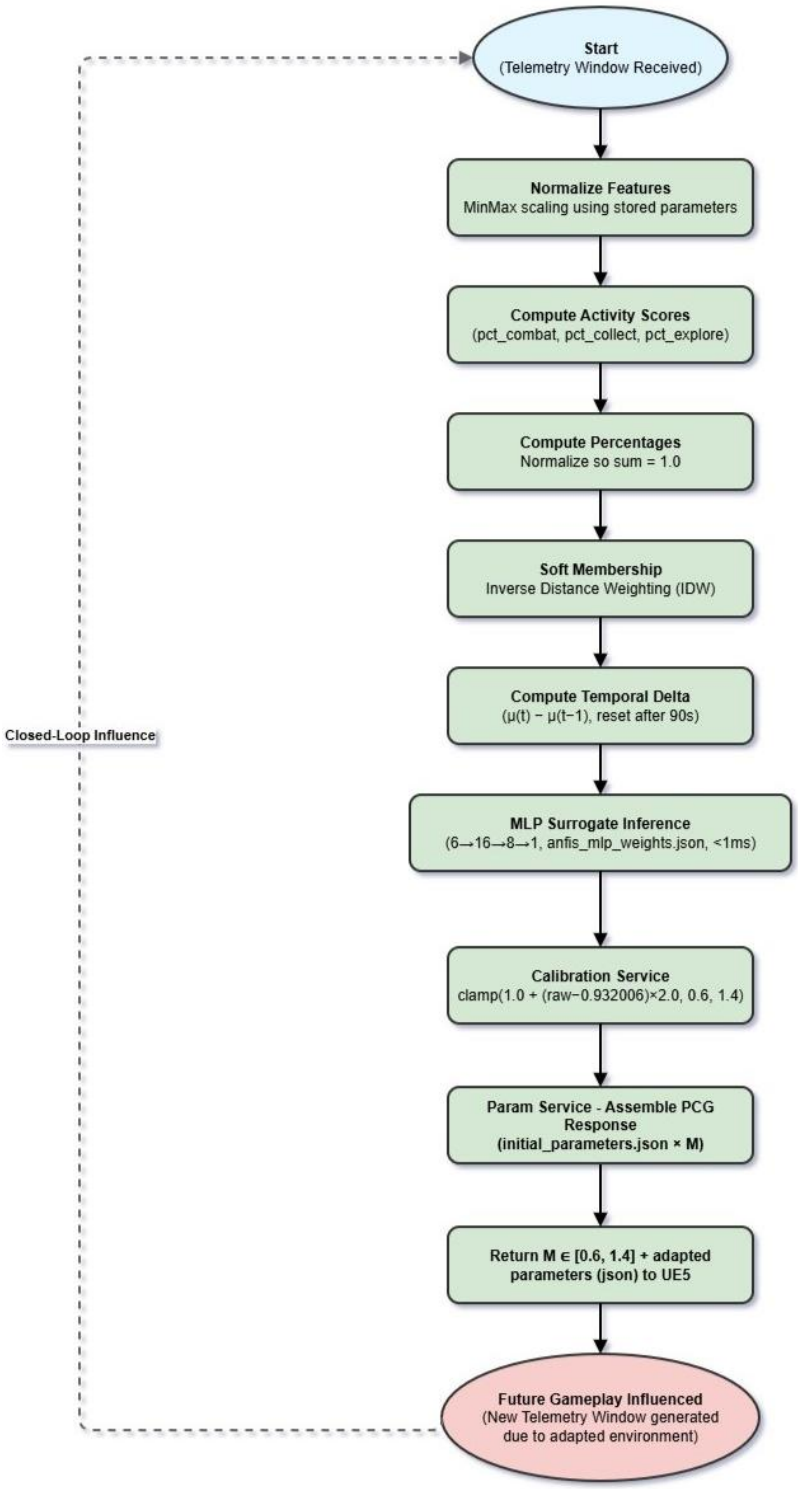


Figure 32: AURA - End-to-End System Process Activity Diagram (Adaptive Control Loop) (Self Composed)

G.4. Backend Inference Activity Flow diagram

The figure is the full image in *Section 6.5.4* of *Chapter 06*


Start
(Telemetry Window Received)
Normalize Features
MinMax scaling using stored parameters
Compute Activity Scores
(pct_combat, pct_collect, pct_explore)
Compute Percentages
Normalize so sum = 1.0
Soft Membership
Inverse Distance Weighting (IDW)
Compute Temporal Delta
(μ(t) − μ(t−1), reset after 90s)
MLP Surrogate Inference
(6→16→8→1, anfis_mlp_weights.json, <1ms)
Calibration Service
clamp(1.0 + (raw−0.932006)×2.0, 0.6, 1.4)
Param Service - Assemble PCG Response
(initial_parameters.json × M)
Return M ∈ [0.6, 1.4] + adapted parameters (json) to UE5
Future Gameplay Influenced
(New Telemetry Window generated due to adapted environment)
Closed-Loop Influence

Appendix H - User Interfaces Design

H.1. UI Design: Developer Analytics Dashboard

Developer analytics dashboard wireframes as referenced in *Section 6.6* in *Chapter 06*. Final implemented interfaces are presented in *Section 7.4 in Chapter 07*. The developer analytics dashboard is designed as a single-page monitoring interface providing research-facing visibility of the AURA adaptive pipeline. The dashboard presents real-time telemetry data, behavioural archetype membership weights across Combat, Collection, and Exploration dimensions, and an adaptation multiplier history chart annotated with the defined adaptation bounds.

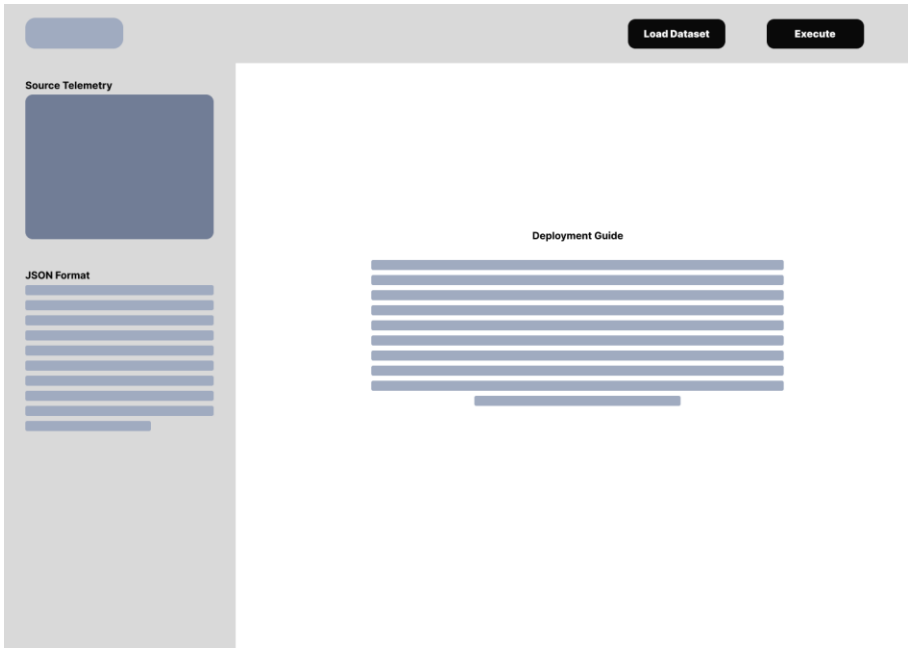


Figure 33: Developer Dashboard - Home Page (Self Composed)

This wireframe presents the main entry interface of the developer dashboard, including source telemetry input, dataset loading controls, and a deployment guide panel used during testing and evaluation.

Figure 34: Developer Dashboard - Pipeline Sequence View (Self Composed)

This interface visualises the sequential flow of telemetry through Phase 2: Telemetry & Model Construction and Phase 3: Backend-Mediated Integration, enabling step-by-step inspection of the inference process.

Figure 35: Developer Dashboard - Analytics Tab (Self Composed)

This view provides analytical insights including archetype membership distributions, multiplier trends, and session-level behavioural summaries for evaluation purposes.

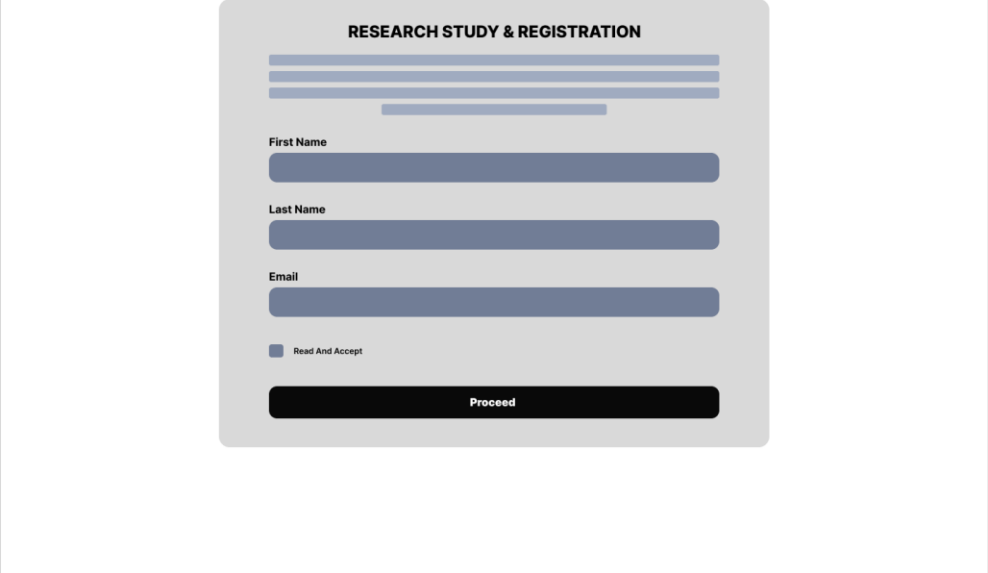


Figure 36: Gameplay UI - Registration Screen (Self Composed)

This screen captures participant details for the research study prior to gameplay, ensuring structured session identification during evaluation.

The interface is intended for researcher use during evaluation sessions, enabling observation of pipeline behaviour without direct interaction with the game environment. All displayed values are derived from the live backend inference pipeline and update on each 30-second telemetry window boundary.

H.2. UI Design: Gameplay Interface

Gameplay interface wireframes as referenced in *Section 6.6* in *Chapter 06*. The gameplay interface encompasses two screen designs used during Phase 3 evaluation sessions. The participant registration screen collects session identification information prior to game initialisation. The in-session heads-up display overlay provides minimal gameplay feedback to participants without exposing adaptive pipeline state.

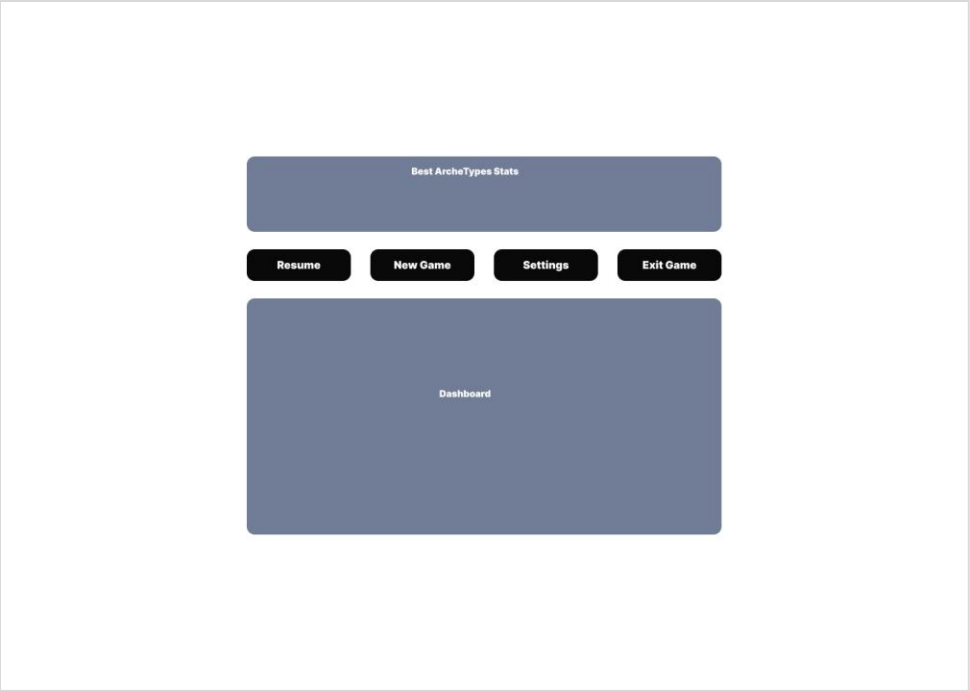


Figure 37: Gameplay UI - In-Game Menu (Self Composed)

This interface presents gameplay controls such as resume, new game, settings, and exit options, along with a dashboard access panel for testing scenarios.

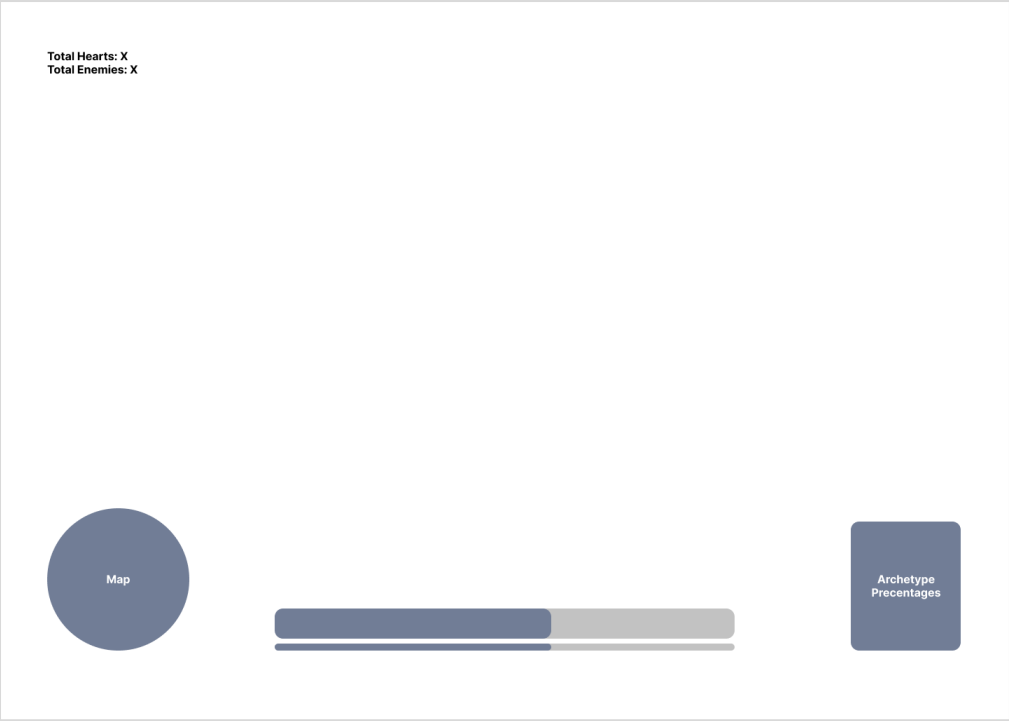


Figure 38: Gameplay UI - Main Gameplay Screen (Self Composed)

This wireframe illustrates the in-game interface including player status indicators, map display, and minimal adaptive feedback elements without exposing internal system computations.

Both interface designs follow a minimal aesthetic to avoid influencing participant behaviour during evaluation. No adaptive state information is visible to participants through either interface element.

Appendix I - Prototype User Interfaces

Prototype user interfaces as referenced in *Section 7.4* in *Chapter 07*.

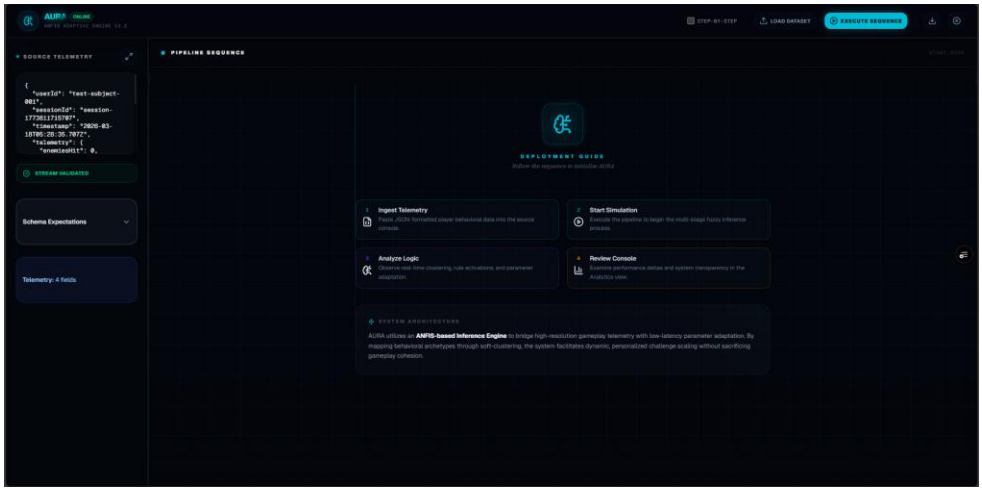


Figure 39: Developer Prototype Dashboard - Home Page (Self Composed)

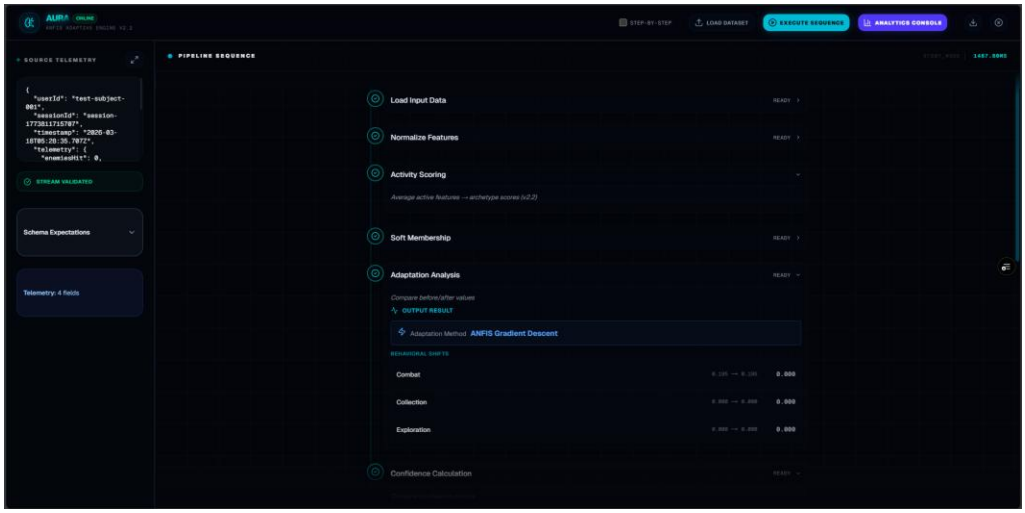


Figure 40: Developer Prototype Dashboard - Pipeline Sequence View (Self Composed)

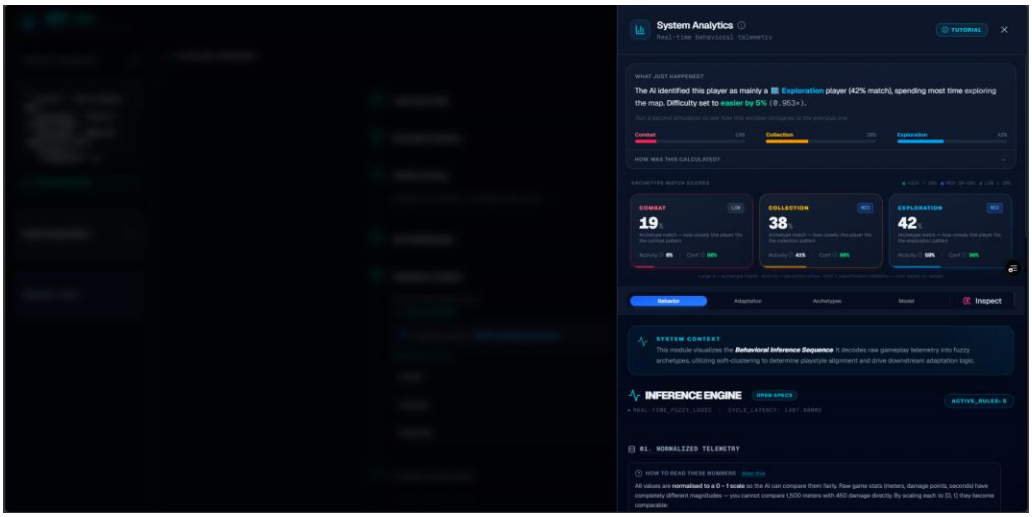


Figure 41: Developer Prototype Dashboard - Analytics Tab (Self Composed)

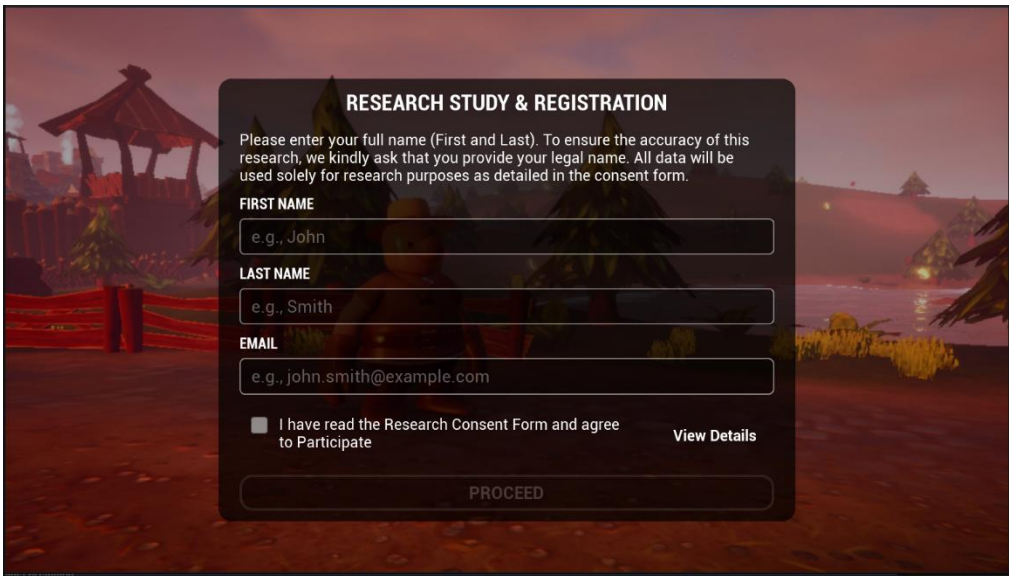


Figure 42: Gameplay Prototype UI - Registration Screen (Self Composed)

Figure 43: Gameplay Prototype UI - In-Game Menu (Self Composed)

Figure 44: Gameplay Prototype UI - Main Gameplay Screen (Self Composed)

Appendix J - Implementation Details

J.1. Phase 1 PCG Mode Configuration and Calibration Data Collection

Implementation details supporting in *Section 7.3* in *Chapter 07* Core Functionalities Implementation A parent Blueprint class defines the complete set of PCG, and player configuration variables used across all gameplay phases. These variables are organised into three parameter groups: PCG collectibles, PCG enemies, and player parameters. Three child Blueprint classes inherit from this parent and override these parameter groups to define the three calibration game modes.

Parameter	**Mode 1**	**Mode 2**	**Mode 3**
PCG Collectibles			
Fixed Collectible Count	120	60	30
Spawn Interval (s)	40	60	80
Collectible Lifetime (s)	30	45	60
PCG Enemy			
Enemy Spawn Interval (s)	40	30	20
Global Enemy Cap	35	45	55
Enemy Damage Intensity	10	13	16
Enemy Max Health	100	120	160
Player Parameters			
Stamina Regen	12	9	7
Stamina Damage	5	7	7
Dash Cooldown (s)	3	4	5
Player Damage Intensity	16	13	10

Player Max Health	180	140	100

Table 50: PCG Mode Configuration Defaults - Three Child Blueprint Classes

Mode 1 presents the most permissive configuration, characterised by high collectible density, low enemy pressure, and favourable player parameters. Mode 3 represents the most challenging configuration, while Mode 2 provides an intermediate balance between the two.

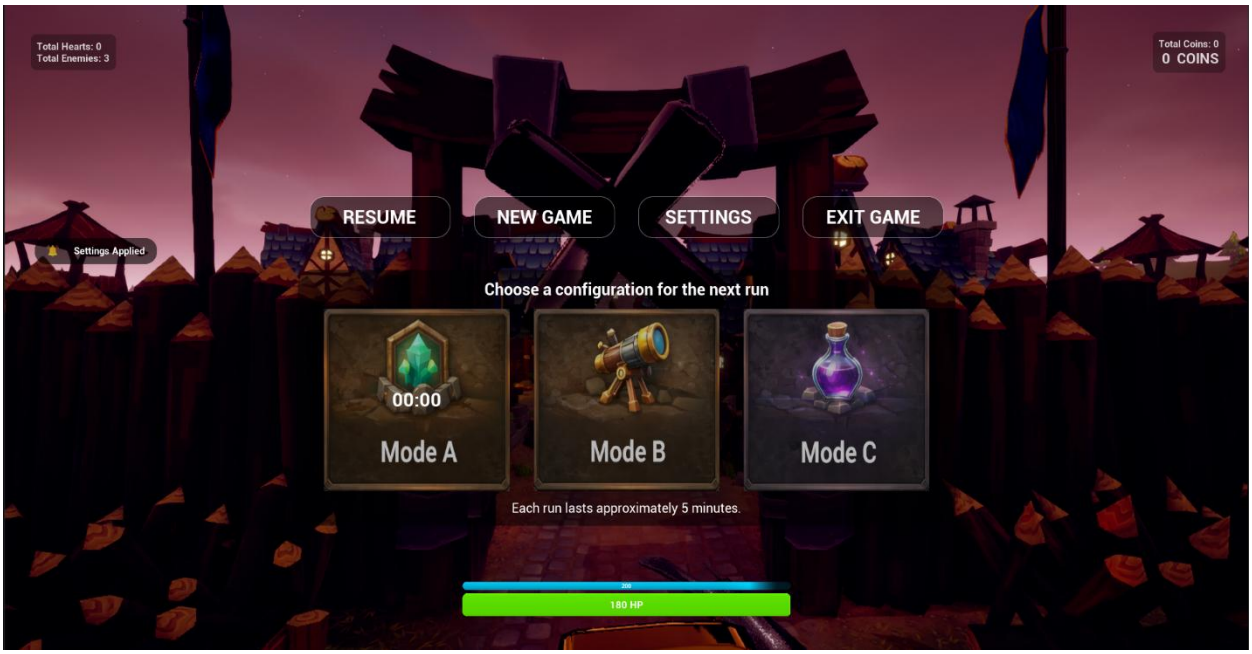


Figure 45: Phase 1 Mode Selection Interface - CollectGame (self-composed)

As shown in Figure 45: Phase 1 Mode Selection Interface - CollectGame (self-composed), participants selected gameplay modes without knowledge of their difficulty ordering, ensuring unbiased calibration across all configurations. Mode labels were anonymised as Mode A, Mode B, and Mode C during the calibration study.

Following the evaluation of all modes using the NeutralityScore metric, Mode 1 was selected as the neutral baseline configuration. This configuration was subsequently fixed and reused for all adaptive runtime sessions.

J.2. Telemetry Aggregation - 30-Second Window Timer

The telemetry aggregation system is implemented using a repeating timer driven by the Set Timer by Event node in Unreal Engine. On each one-second tick, a session counter is incremented.

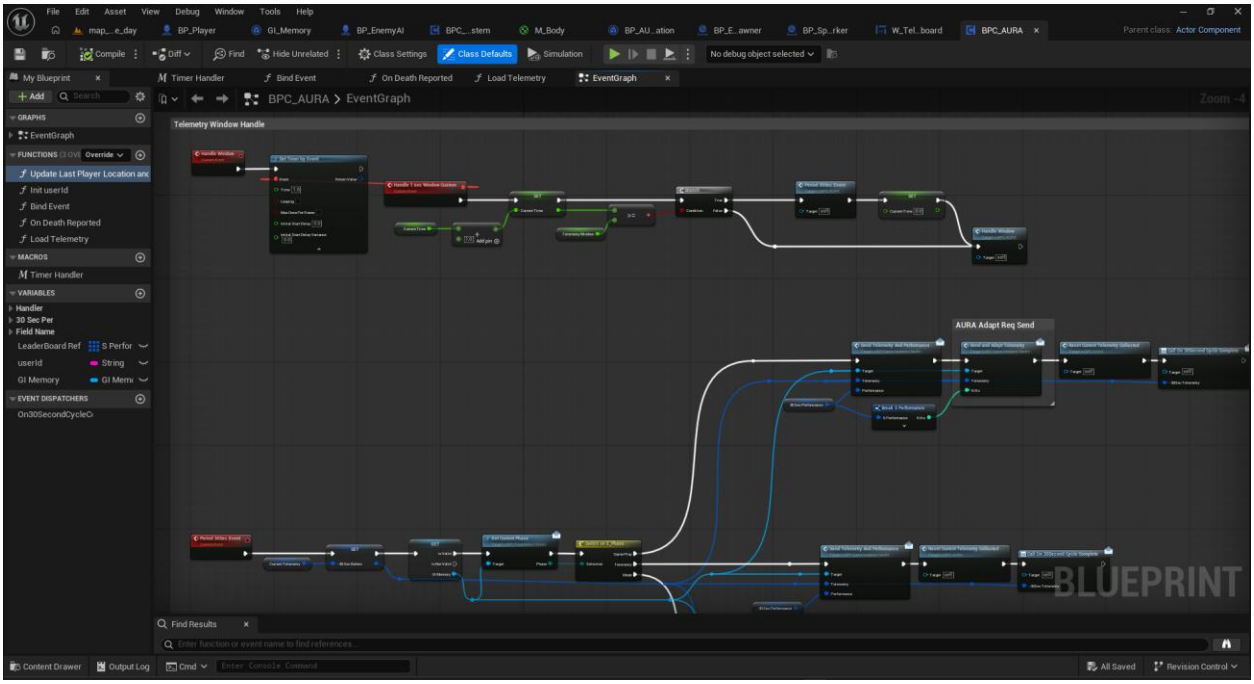


Figure 46: 30-Second Telemetry Window Timer Blueprint (Self Composed)

As shown in Figure 46: 30-Second Telemetry Window Timer Blueprint (Self Composed), when the counter reaches 30 seconds, a snapshot of all telemetry variables is captured and dispatched to the backend inference service. The counter is then reset to begin the next aggregation cycle. This fixed window approach ensures consistent temporal resolution while maintaining low computational overhead.

J.3. VaRest HTTP POST - Telemetry Dispatch

At each 30-second boundary, the collected telemetry data is serialised into a JSON object and transmitted to the backend inference service using the VaRest HTTP plugin.

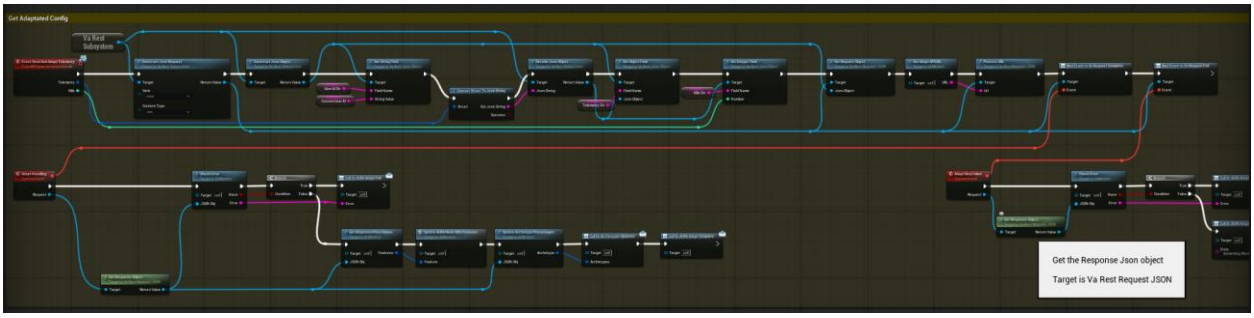


Figure 47: VaRest HTTP POST - Telemetry Dispatch Blueprint (Self Composed)

As shown in Figure 47: VaRest HTTP POST - Telemetry Dispatch Blueprint (Self Composed),

each telemetry variable is written into the JSON payload as a named field, matching the schema expected by the backend inference service. The HTTP POST request is executed asynchronously to prevent blocking the game thread during runtime execution.

J.4. Response Handling - Adaptation Application

Upon receiving a response from the backend inference service, the Unreal Engine client processes the returned multiplier and adapted parameter values.

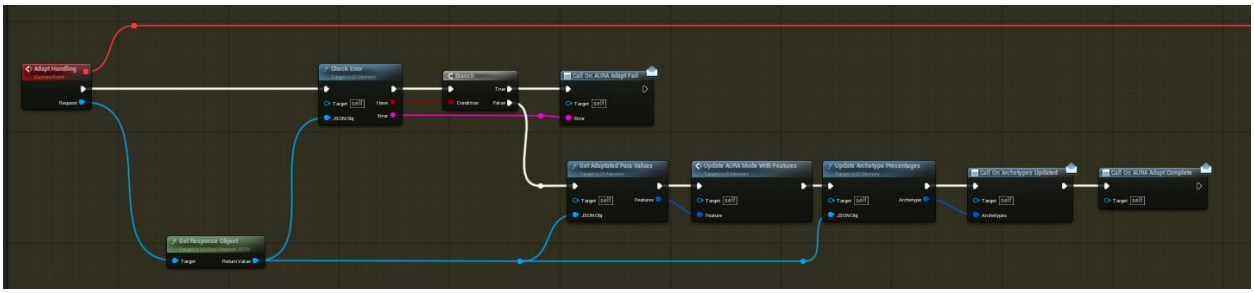


Figure 48: Response Handling and PCG Parameter Application Blueprint (Self Composed)

As shown in Figure 48: Response Handling and PCG Parameter Application Blueprint (Self Composed), an event dispatcher is used to propagate the received values across Blueprint components. Player-related parameters are updated immediately, while PCG-related parameters are applied during subsequent generation cycles. This forward-only update strategy ensures consistency in gameplay progression.

J.5. Backend Inference Pipeline

The backend inference pipeline is implemented as a POST endpoint within the Next.js application. At runtime, calibration artefacts and surrogate model parameters are loaded into memory to minimise processing latency.

```
process(telemetry: TelemetryWindow): PipelineOutput {
    const perfTimings: Record<string, number> = {};
    const t0 = performance.now();

    // Step 1: Validate
    this.step1_AcquireAndValidate(telemetry);

    // Step 2: Normalize (Min-Max, 12 features including 2 derived)
    const normalized = this.step2_NormalizeFeatures(telemetry.features);

    // Step 3: Activity scores (per-archetype averages → equal ceiling)
    const activityScores = this.step3_CalculateActivityScores(normalized);

    // Step 4: Soft membership via IDW (K=3)
    const softMembership = this.step4_FuzzyClustering(activityScores);

    // Step 5: Temporal deltas (window-to-window membership change)
    const timestamp = telemetry.timestamp
      ? new Date(telemetry.timestamp).getTime()
      : Date.now();

    const deltas = this.step5_ComputeDeltas(telemetry.userId, softMembership, timestamp);

    // Step 6: MLP forward pass (6→16→8→1)
    const { anfisInput, mlpResult } = this.step6_InferenceEngine(softMembership, deltas);

    // Step 7: Neutral-centred calibration + parameter adaptation
    const { targetMultiplier, multiplierClamped, adaptedParameters } = this.step7_AdaptationAnalysis(mlpResult.result, softMembership);

    // Step 8: Aggregate result
    perfTimings.total = performance.now() - t0;

    return {
      filtering: { passed: true, duration_seconds: telemetry.duration ?? 30 },
      normalized_features: normalized,
      activity_scores: activityScores,
      soft_membership: softMembership,
      deltas,
      anfis_input: anfisInput,
      mlp_output: mlpResult.result,
      inference: {
        rulesFired: mlpResult.activations.map((val, idx) => ({
          ruleName: `Hidden Neuron #${idx + 1}`,
          strength: val
        })).sort((a, b) => b.strength - a.strength)
      },
      target_multiplier: targetMultiplier,
      adapted_parameters: adaptedParameters,
      validation: validatePipelineResult(softMembership, deltas, adaptedParameters, multiplierClamped),
      performance_timings: perfTimings,
    } as any;
  }
```

Figure 49: Using Telemetry Data, 8-steps adaptation values generation function (Self Composed)

As shown in Figure 49: Using Telemetry Data, 8-steps adaptation values generation function (Self Composed), incoming telemetry requests are processed through a deterministic sequence of stages:

- feature normalisation
- activity scoring
- soft membership computation
- temporal delta derivation
- surrogate inference
- output bounding

This pipeline ensures that each telemetry window is transformed into a bounded adaptation multiplier and corresponding parameter adjustments.

J.6. Adaptation Analysis

The adaptation analysis stage computes the final multiplier and adapted parameters based on the behavioural state vector derived from telemetry.

```
/**
 * Neutral-centred calibration:
 *   display = clamp(1.0 + (raw - mlp_neutral) × AMPLIFICATION, 0.6, 1.4)
 *
 * mlp_neutral is the MLP's raw output for the balanced (¼,¼,¼,0,0,0) input —
 * the semantic definition of "no change needed". This ensures a balanced player
 * always maps to exactly 1.0 regardless of how the training distribution shifts
 * between retrains.
 *
 * AMPLIFICATION = 2.0: a raw deviation of ±0.4 from neutral (achievable by
 * moderate archetype + delta combinations) spans ±0.8 of the display range,
 * reaching the 0.6–1.4 extremes.
 */
private computeTargetMultiplier(rawOutput: number): { targetMultiplier: number; multiplierClamped: boolean } {
  const [minM, maxM] = this.manifest.hard_constraints.target_multiplier_range;
  const AMPLIFICATION = 2.0;
  const rescaled = 1.0 + (rawOutput - this.mlpNeutral) * AMPLIFICATION;
  const targetMultiplier = Math.max(minM, Math.min(maxM, rescaled));
  return {
    targetMultiplier,
    multiplierClamped: rescaled !== targetMultiplier
  };
}

private step7_AdaptationAnalysis(rawOutput: number, softMembership: any) {
  const { targetMultiplier, multiplierClamped } = this.computeTargetMultiplier(rawOutput);
  const adaptedParameters = applyAdaptationContract(targetMultiplier, softMembership);
  return { targetMultiplier, multiplierClamped, adaptedParameters };
}
```

Figure 50: Adaptation Analysis - Surrogate Inference Logic (Self Composed)

As shown in Figure 50: Adaptation Analysis - Surrogate Inference Logic (Self Composed), the surrogate model output is transformed relative to the neutral calibration baseline to produce bounded parameter updates. The output is constrained within predefined limits to ensure stable and controlled gameplay adaptation.

J.7. Developer Dashboard Implementation

The developer dashboard is implemented as a web application using Next.js and is designed to provide real-time visibility into the adaptive pipeline.

The dashboard includes the following components:

- A membership distribution chart showing Combat, Collection, and Exploration weights per telemetry window
- A multiplier time-series chart illustrating adaptation behaviour over time
- A session log table recording intermediate pipeline values and validation states

These components enable monitoring and verification of system behaviour without exposing adaptation state information to participants.

J.8. Implementation Challenges and Resolutions

Implementation challenges and resolutions as referenced in Section 7.5. Four primary challenges were encountered during the implementation of the AURA framework:

J.8.1. Inference Latency Constraint

Direct ANFIS inference introduced unacceptable latency for real-time gameplay. This was addressed by approximating a canonical ANFIS-inspired adaptation surface using a lightweight MLP surrogate, enabling efficient real-time inference.

J.8.2. Behavioural Signal Imbalance

Initial archetype scoring exhibited bias due to passive signal accumulation. This was corrected by restructuring feature groupings and applying balanced averaging strategies.

J.8.3. Backend Cold-Start Delay

Deployment on cloud infrastructure introduced initial latency during service startup. This was mitigated through warm-up requests and fallback mechanisms to maintain stable system behaviour.

J.8.4. Numerical Stability in Membership Computation

Floating-point precision issues affected the stability of inverse-distance weighting. Epsilon stabilisation was introduced to preserve numerical consistency while maintaining accurate membership distribution.

J.9. Evidence of Parameter Adaptation

The figure below summarises the environment adaptation through pcg using according to

parameter adapted values reviewed in Section 7.3.4 of *Chapter 07*, including individual parameters.

The following figures present implementation-level evidence of runtime parameter adaptation within the AURA framework. Adaptation is executed through a continuous telemetry inference application loop, where gameplay behaviour is aggregated, transmitted to the backend, and translated into procedural and gameplay parameter updates within Unreal Engine. All parameters originate from a neutral baseline configuration (Table 50: PCG Mode Configuration Defaults - Three Child Blueprint Classes in J.1. Phase 1 PCG Mode Configuration and Calibration Data Collection) and are dynamically updated at fixed temporal intervals.

J.9.1. Telemetry Aggregation and Dispatch

During gameplay, player actions are continuously tracked and grouped into fixed time windows. Once the interval is reached, the collected telemetry is prepared and sent to the backend without interrupting gameplay. This ensures a stable and consistent flow of behavioural data for adaptation.

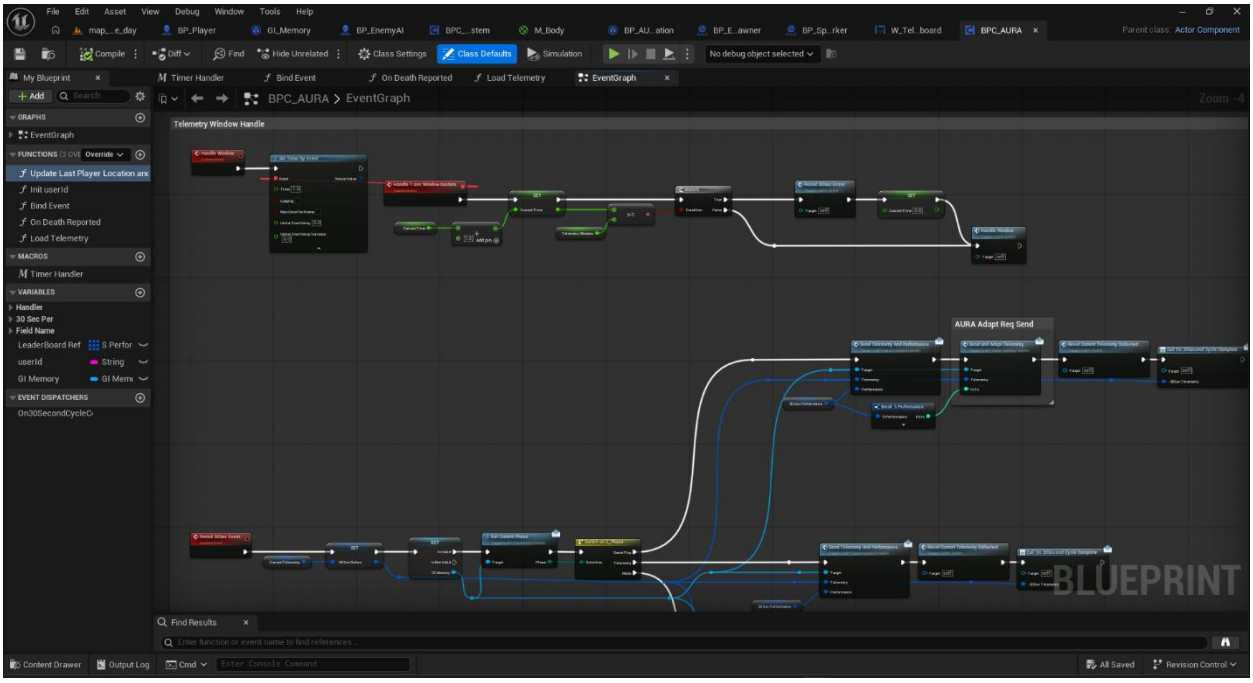


Figure 51: Telemetry aggregation and timed dispatch using BPC_AURA.

J.9.2. Backend Communication and Response Handling

After telemetry is sent, the system waits for the backend response containing the adapted values. The response is validated and processed to ensure that only correct and usable data is applied. If

an error occurs, fallback handling ensures system stability.

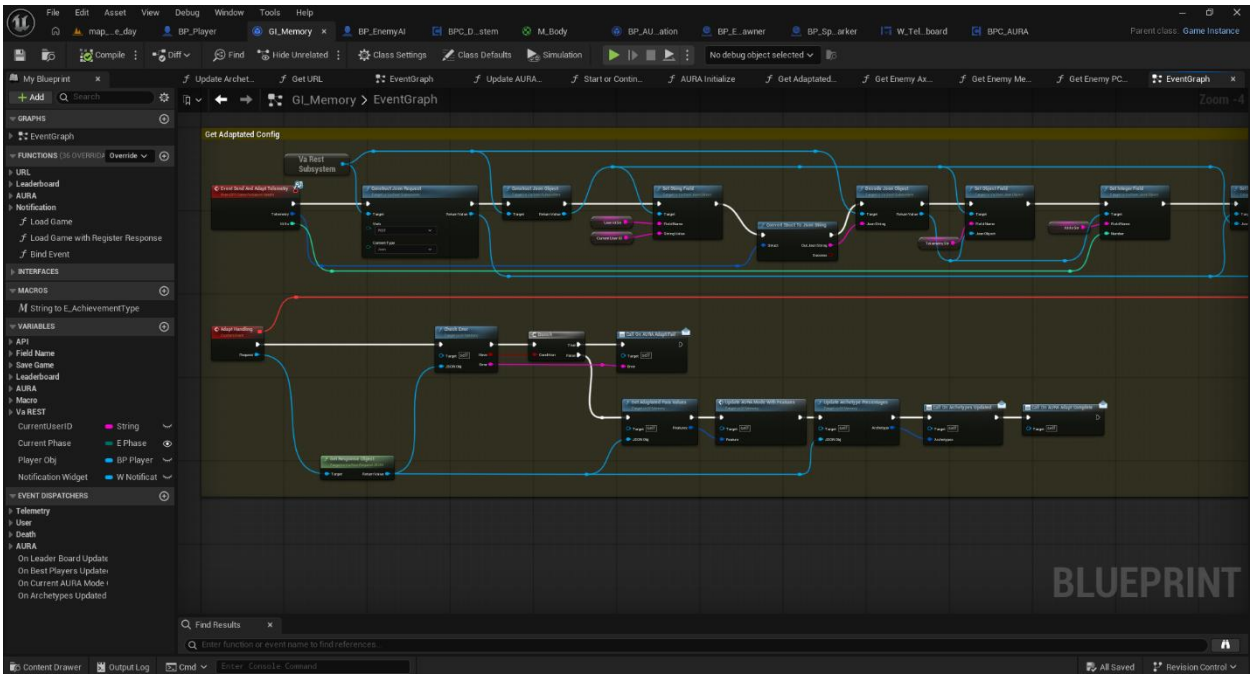


Figure 52: HTTP request construction and telemetry transmission via GI_Memory

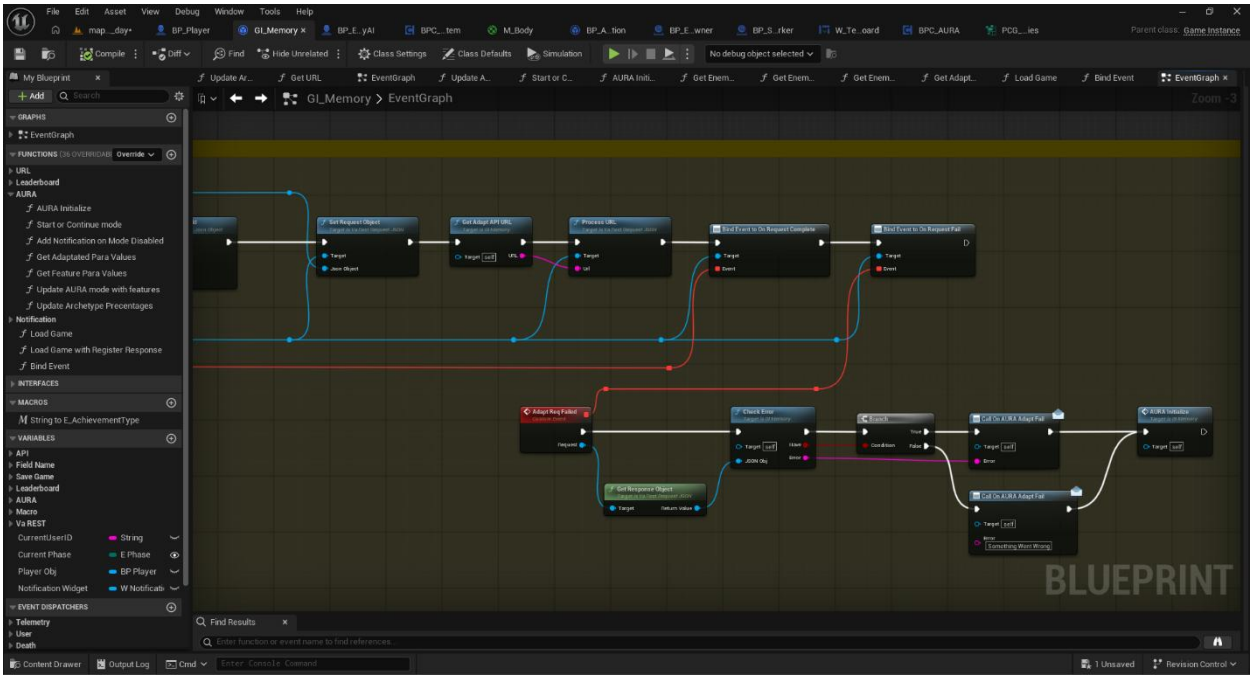


Figure 53: Backend response validation and error handling during adaptation

J.9.3. Parameter Extraction and Feature Mapping

Once the backend response is received, the returned values are extracted and mapped into structured gameplay parameters. These include player attributes, enemy behaviour, and procedural

generation settings which are used throughout the system.

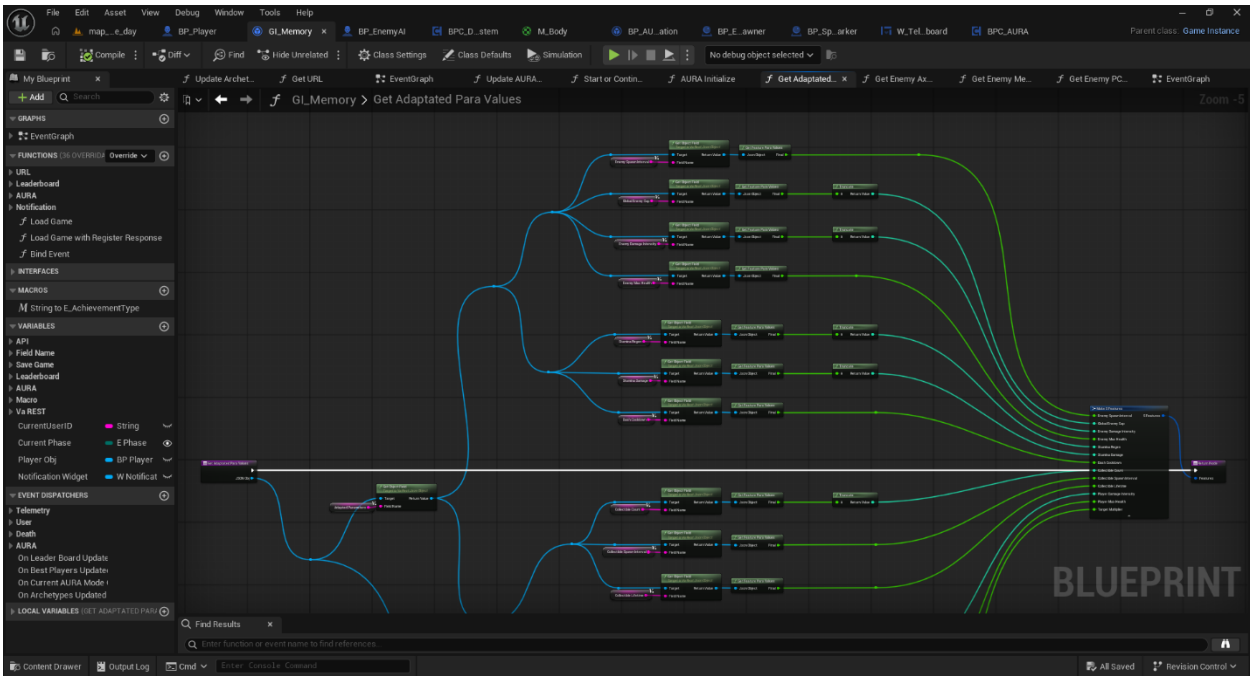


Figure 54: Extraction of adapted parameter values from backend response

J.9.4. AURA Mode Update with Adapted Features

The extracted parameters are applied to the active AURA mode, updating the system state. This step ensures that all gameplay systems receive consistent adaptation values derived from player

behaviour.

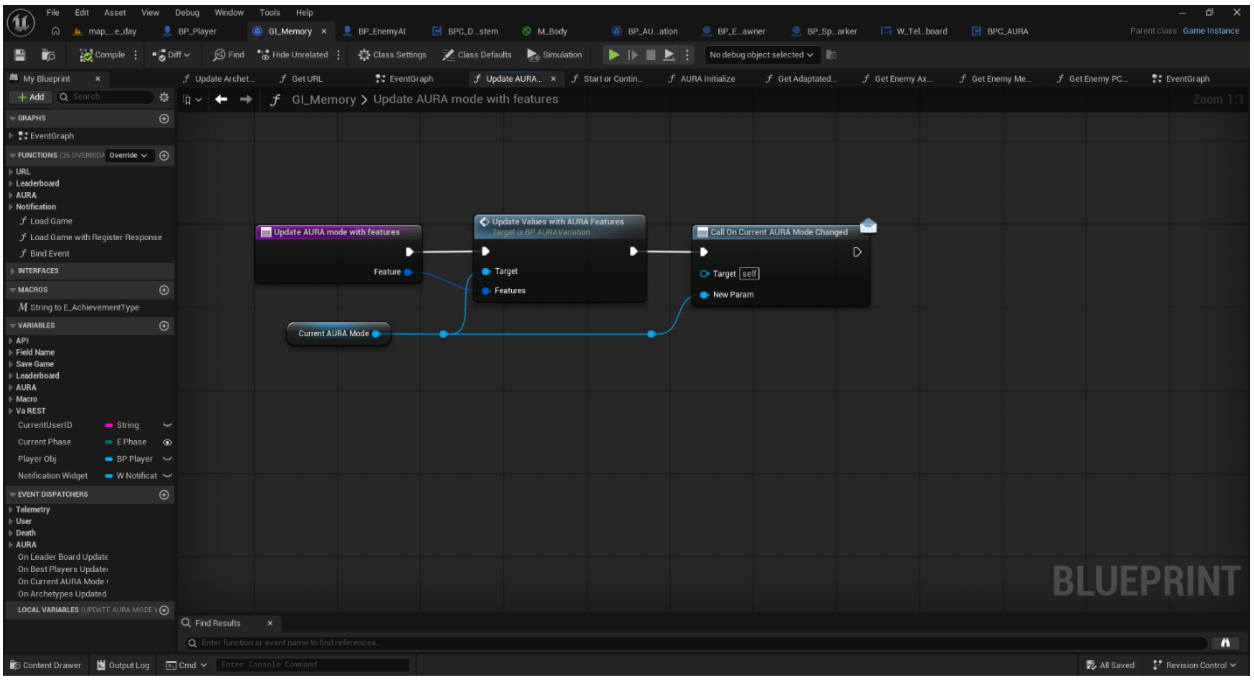


Figure 55: Updating active AURA mode with adapted feature values

J.9.5. Player Parameter Adaptation

The adapted values are applied directly to player systems such as stamina, damage, and health. These updates influence how the player experiences difficulty and progression during gameplay.

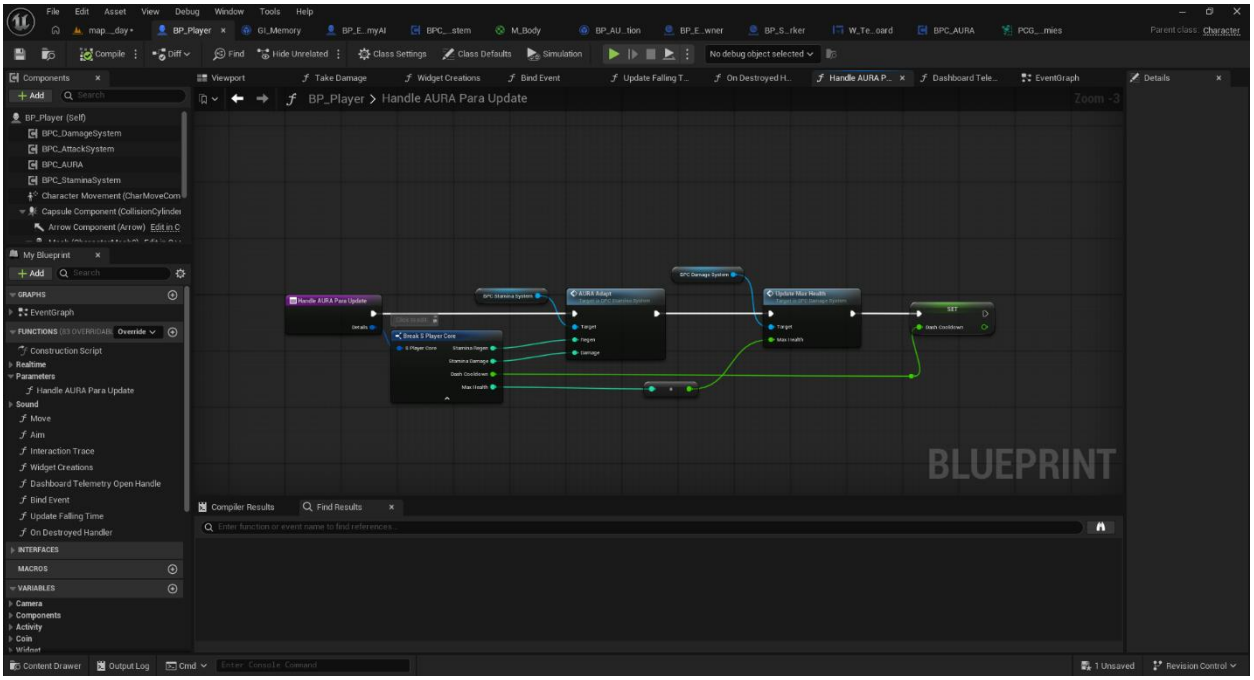


Figure 56: Player parameter application

J.9.6. Stamina and Damage Adaptation

Player stamina and damage values are updated smoothly to avoid sudden changes in gameplay. Interpolation is used to gradually adjust values over time, maintaining a natural gameplay experience.

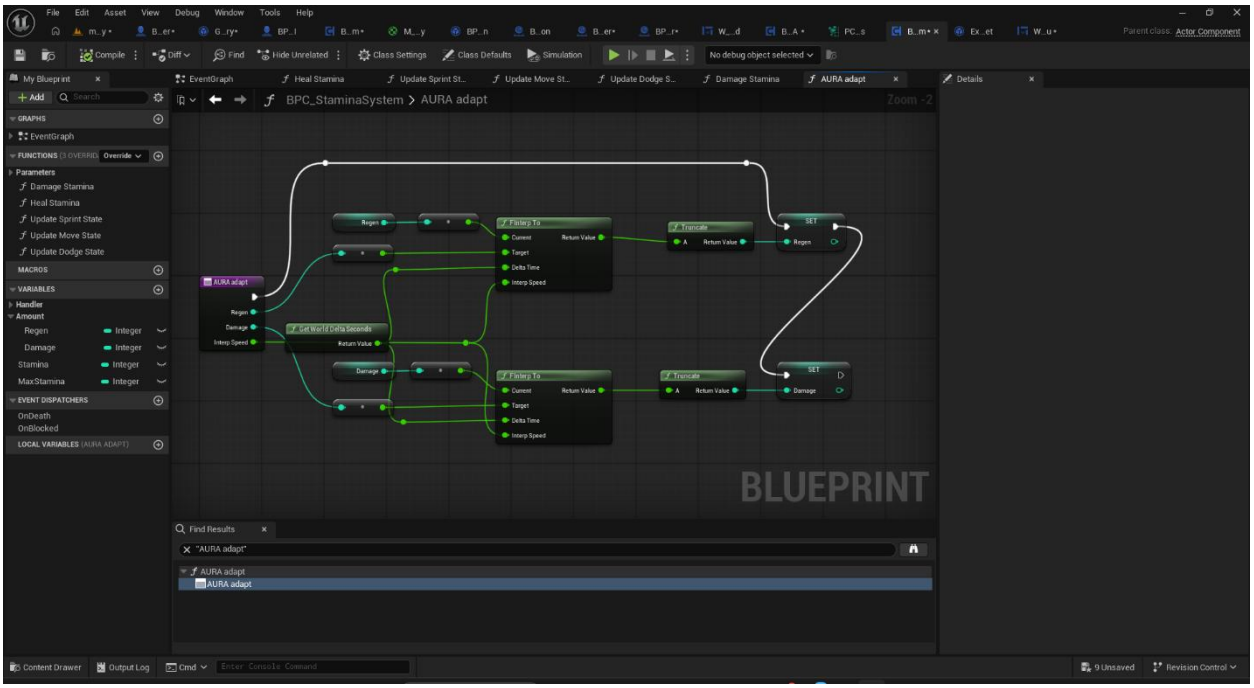


Figure 57: Smooth adaptation of stamina and damage parameters.

J.9.7. Health Adaptation with Smoothing

Health-related parameters are updated using interpolation to ensure stability. This prevents abrupt increases or decreases that could negatively affect gameplay balance.

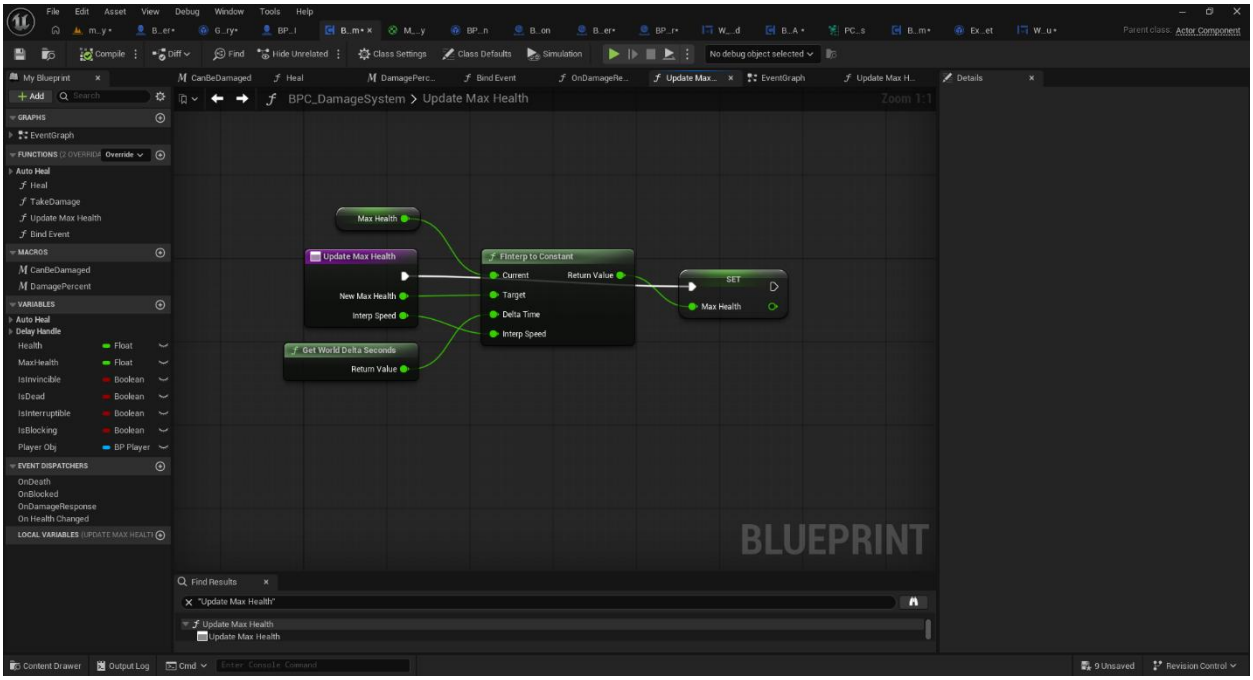


Figure 58:Interpolated update of maximum health values.

J.9.8. Collectible Spawning Adaptation

The system adapts how collectibles are spawned within the environment based on player behaviour. This allows the game to dynamically adjust rewards and exploration opportunities.

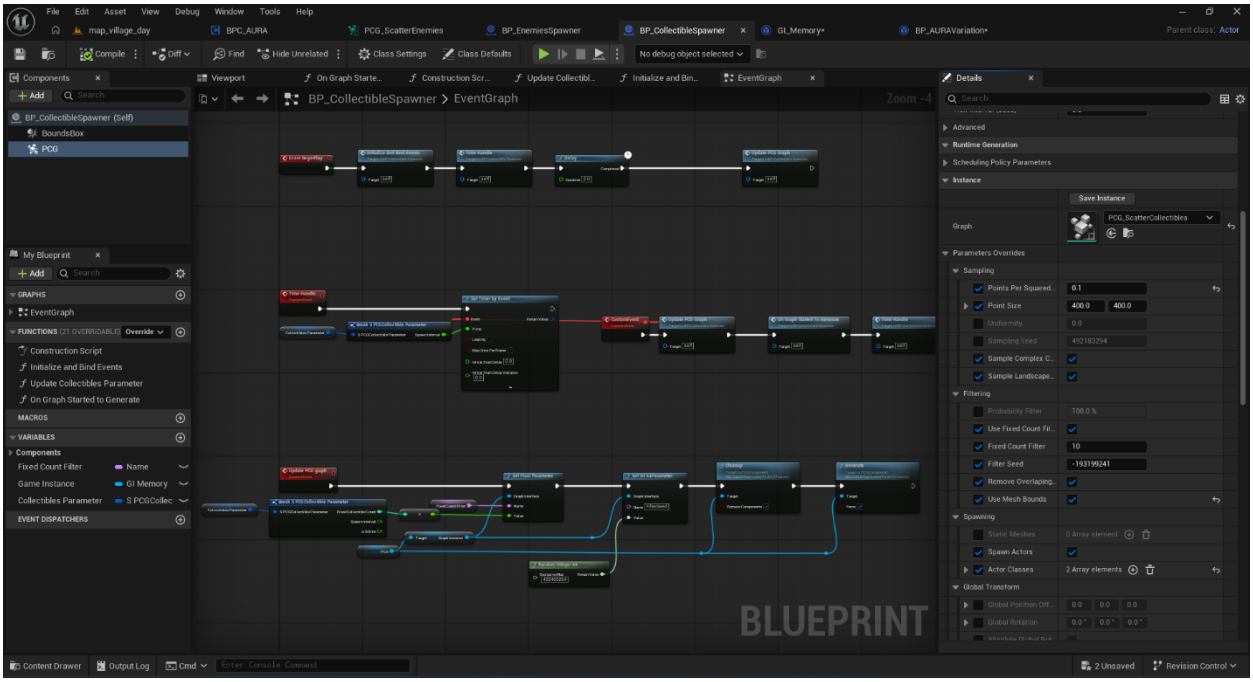


Figure 59: Adaptive collectible spawning using PCG parameters.

J.9.9. Enemy Spawning Adaptation

Enemy spawning behaviour is adjusted dynamically to reflect player performance. This includes changes in spawn density and distribution across the environment.

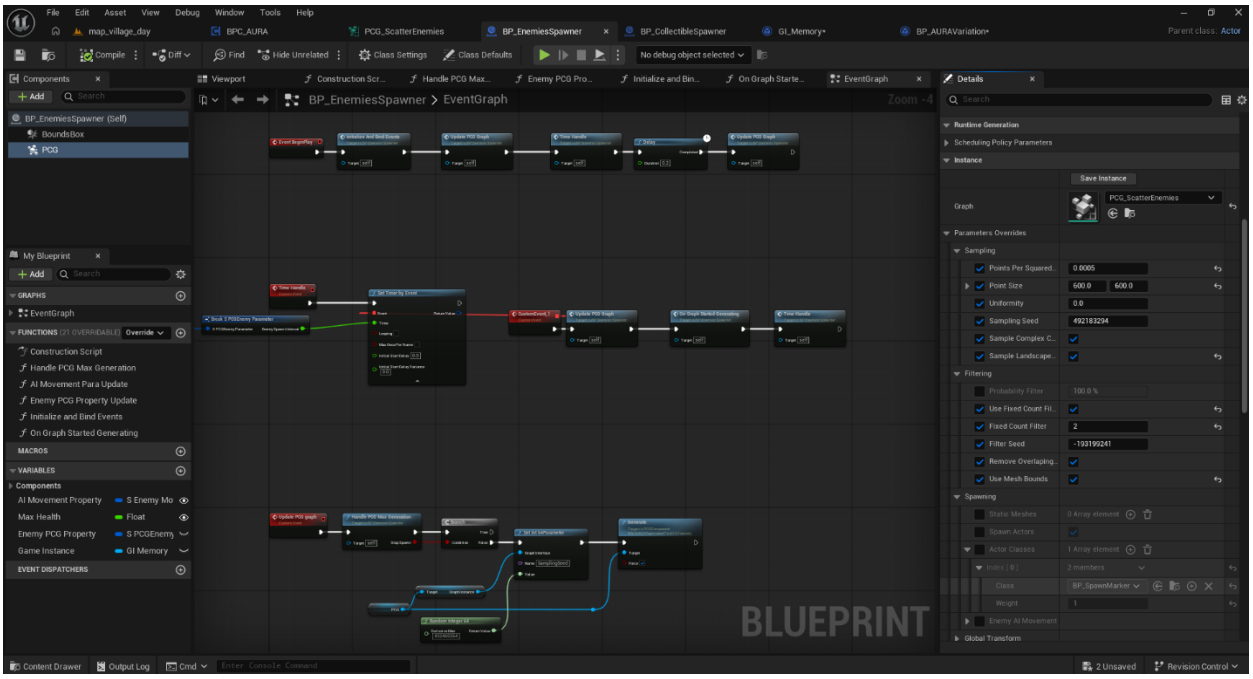


Figure 60: Adaptive enemy spawning through PCG updates.

J.9.10. PCG Graph Adaptation for Enemies

The procedural graph controlling enemy generation is updated using adapted values. This ensures

that environmental difficulty evolves alongside player behaviour.

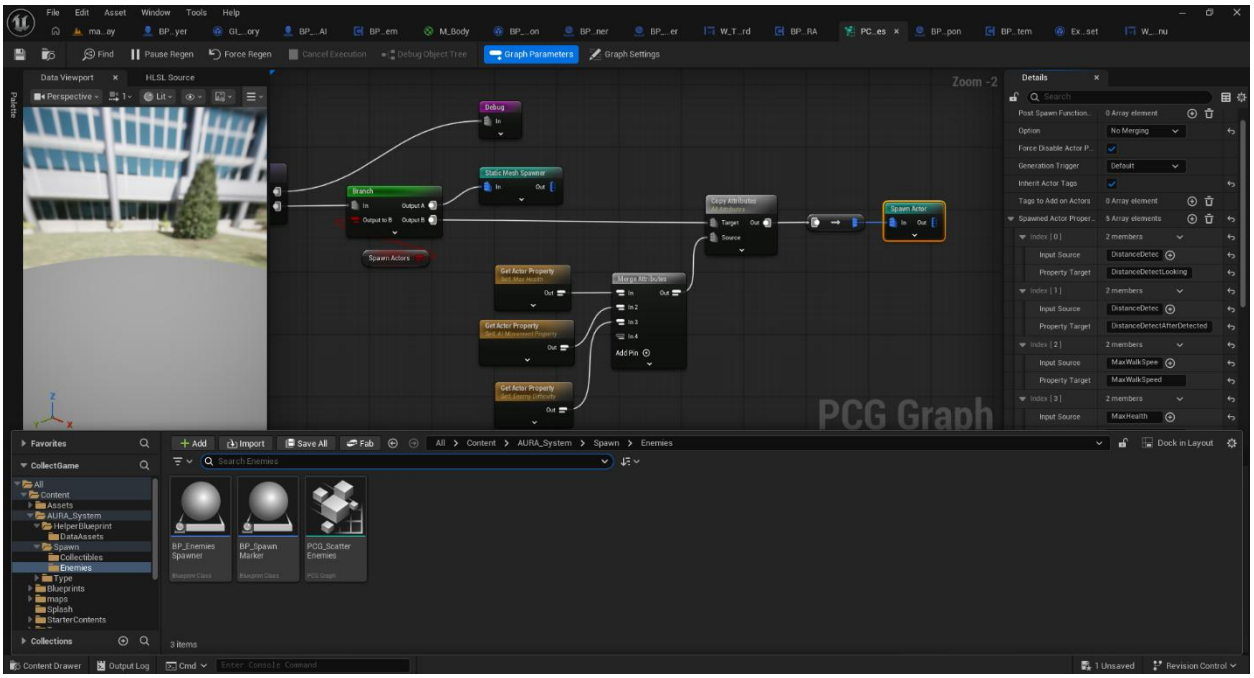


Figure 61: PCG graph controlling enemy generation with adaptive parameters.

J.9.11. PCG Graph Adaptation for Collectibles

The collectible generation graph is also influenced by adapted values, enabling dynamic placement and variation within the environment.

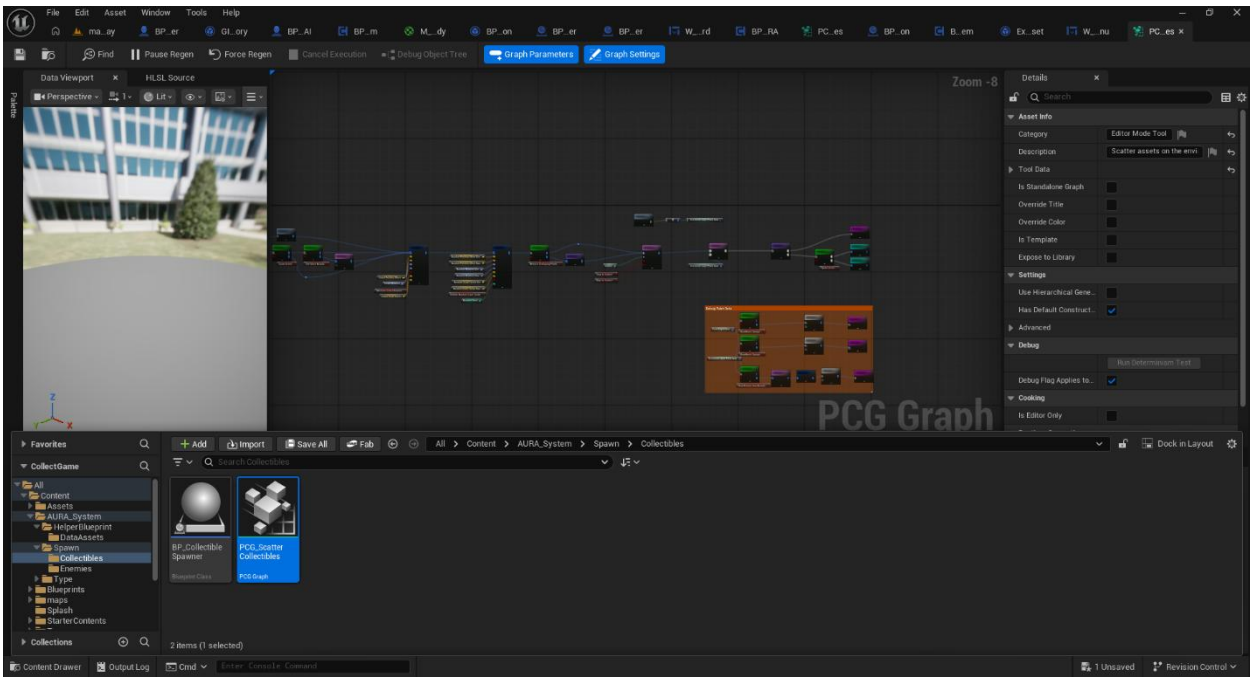


Figure 62: PCG graph for adaptive collectible distribution.

J.9.12. Archetype Behaviour Output

Finally, the system calculates behavioural archetype distributions based on player activity. These values represent how the player engages with combat, collection, and exploration, and drive further adaptation cycles.

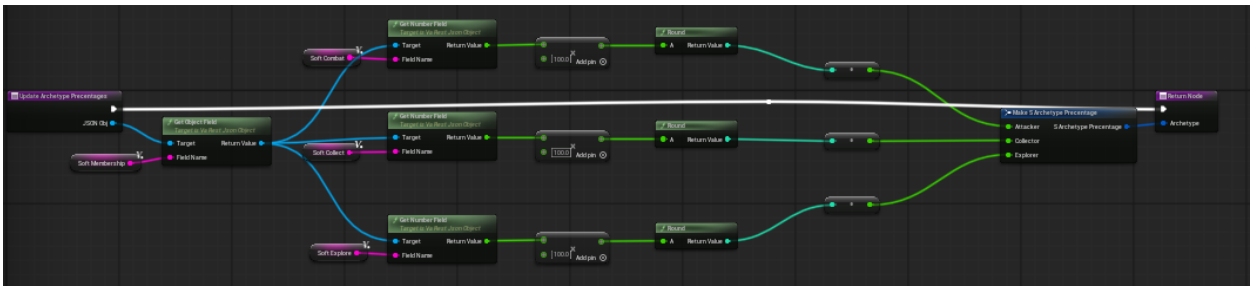


Figure 63: Computation of behavioural archetype percentages.

Below figure presents the in-game parameter display panel, confirming base and current values during an adaptive session, with all eleven parameters receiving the computed adaptation multiplier through Blueprint-mediated parameter updates at each 30-second window boundary

Figure 64: AURA In-Game PCG Parameter Adaptation - Base and Current Values (Self Composed)

Appendix K - Testing Evidence

K.1. Bootstrap Confidence Interval - Full Results

Full bootstrap confidence interval results supporting Section 8.5.1 in *Chapter 08*.

The table presents the complete results of the bootstrap confidence interval analysis conducted over 100 resampling iterations on the held-out test set.

Bootstrap Metric	Value	Notes
Iterations	100	Random resampling with replacement on test split
Mean R^2	0.9293	Across all 100 resampled test sets
Standard Deviation R^2	0.0229	Indicates stable performance across resamples
95% CI Lower Bound	0.8744	2.5th percentile of bootstrap distribution
95% CI Upper Bound	0.9544	97.5th percentile of bootstrap distribution
Min R^2 observed	0.8623	Worst-case resample
Max R^2 observed	0.9574	Best-case resample
Stored Test R^2 within CI?	✓ Yes - 0.9264 ∈ [0.874, 0.954]	Confirms stored metric is statistically reliable

Table 51: Bootstrap Confidence Interval Analysis - Full Results (100 Iterations)

K.2. Ranked Feature Sensitivity Scores

The following table presents the complete ranked sensitivity scores from the univariate sensitivity analysis referenced in *Section 8.5.2*.

Rank	Feature	Impact Score	Interpretation
1	soft_combat	0.1449	Dominant driver (non-linear peak at ~0.6 membership)

2	soft_explore	0.0981	Monotone increase across full range
3	delta_combat	0.0941	Strong inverse relationship with predicted M
4	delta_collect	0.0318	Non-monotone (moderate influence at negative deltas)
5	soft_collect	0.0213	Weakest archetype-level driver
6	delta_explore	0.0178	Lowest marginal impact (limited dynamic range)

Table 52: Ranked Feature Sensitivity Scores - MLP Surrogate

K.3. Unit Test Suite Breakdown - collect-game-website

The following table presents the complete test suite breakdown for the collect-game-website inference pipeline (Vitest v4.0.18), referenced in *Section 8.9.1*.

Test File	Module Under Test	Tests	Pass	Fail	Status
mlp.test.ts	MLP forward pass, ReLU, linear output, safety bounds, batch predictions	4	4	0	PASS
activity.test.ts	Activity scoring v2.2 - combat÷5, collect÷4, explore÷2, partial data	4	4	0	PASS
normalization.test.ts	MinMaxNormalizer v2.2 - range normalisation, upper/lower clamping, missing features	4	4	0	PASS
clustering.test.ts	KMeansSoftMembership - IDW partition-of-unity ($\sum\mu_k$=1.0), centroid membership	3	3	0	PASS
pipeline.test.ts	ANFISPipeline Integration v2.2 - full 8-step pipeline, session reset	2	2	0	PASS
adaptation.test.ts	Adaptation Sensitivity v2.2 - per-parameter sensitivity, archetype influence	2	2	0	PASS
TOTAL	6 suites - all pipeline modules covered	19	19	0	PASS

Table 53: Unit Test Suite - Vitest v4.0.18

K.4. Full Functional Test Cases

The table presents the complete black-box test case results for all 12 functional requirements. All test cases were derived from the functional requirements specified in *Section 8.7* in *Chapter 04*.

TC	FR	Action / Trigger	Expected Outcome	Actual Outcome	Method	result
1	FR-01	VaRest HTTP POST dispatched at each 30-second window boundary from CollectGame	Backend receives payload. HTTP 200 response returned to UE client. telemetry persisted to MongoDB	HTTP 200 returned. document stored in CollectGame.Telemetry MongoDB	CI / UE	pass
2	FR-02	Request with missing userId submitted to /api/pipeline	HTTP 400 returned. validation error in response body	HTTP 400 returned correctly (CI TC-A2)	CI	pass
3	FR-02	Request with telemetry object absent	HTTP 400 returned. validation error in response body	HTTP 400 returned correctly (CI TC-A3)	CI	pass
4	FR-02	Empty body submitted	HTTP 400 returned	HTTP 400 returned correctly (CI TC-A4)	CI	pass
5	FR-03	Valid 11-field telemetry payload received by NormalisationService	Feature vector normalised using frozen MinMax scaler. all values $\in$ [0,1]	Normalised vector within [0,1] (Vitest normalization.test.ts: 4/4)	Unit	pass
6	FR-04	Normalised vector passed to ActivityScoringService	Activity scores computed with v2.2 divisors: combat÷5, collect÷4, explore÷2	Scores computed correctly (Vitest activity.test.ts: 4/4)	Unit	pass
7	FR-05	Activity scores submitted to KMeans soft membership	IDW soft membership computed. $\sum\mu_k=1.0$ enforced per window	Partition-of-unity verified (Vitest clustering.test.ts: 3/3)	Unit	pass

		service				
8	FR-06	Soft membership scores and session state passed to temporal delta service	Δcombat, Δcollect, Δexplore computed with α=0.2 smoothing. zeroed on 90 s timeout	Delta computation and session reset verified (Vitest pipeline.test.ts: 2/2)	Unit	pass
9	FR-07	6-feature vector passed to MLPInferenceService	MLP (6→16→8→1) produces scalar raw output < 1 ms latency	Forward pass within valid range < 1 ms (Vitest mlp.test.ts: 4/4)	Unit	pass
10	FR-08	Raw MLP output passed to CalibrationService	display = clamp (1.0 + (raw−0.932006) × 2.0, 0.6, 1.4) M ∈ [0.6, 1.4]	Bounded M returned. boundary cases verified (Vitest adaptation.test.ts: 2/2)	Unit	pass
11	FR-09	Full pipeline completes. ParamService assembles response	HTTP response contains target_multiplier and adapted_parameters with all PCG values	Complete response returned (CI TC-A1 PASS. Vitest pipeline PASS)	CI/Unit	pass
12	FR-10	90 seconds elapses without telemetry dispatch from a userId	Session delta state zeroed. next window treated as first-window condition	Session reset verified (Vitest pipeline.test.ts)	Unit	pass
13	FR-11	Telemetry POST processed. MongoDB write invoked	Document persisted to CollectGame.Telemetry instance	Data verified in MongoDB via CollectGame.Telemetry repository	Manual	pass
14	FR-12	Developer opens collect-game-	Dashboard renders archetype	All panels rendered correctly. pipeline	Visual	pass

		website.vercel.app dashboard	membership, multiplier progression, session logs, and telemetry inputs	sequence view displays all 8 stages		
15	FR-11/12	Backend returns adapted PCG parameters. Blueprint OnRequestComplete fires	PCG params applied at next spawn cycle. player params applied immediately. no retroactive modification	Forward-only constraint observed during Phase 1 session observation	UE	pass

Table 54: Full Functional Test Cases - AURA

K.5. Module and Integration Testing

Module and integration testing evidence supporting *Section 8.9* in *Chapter 08*.

The table presents the module integration test results for all eight backend pipeline components, verifying correct data flow across defined module interfaces.

Module	Input	Expected Output	Actual Output	Status
Normalisation Service	Raw 11-field telemetry window	Normalised feature vector. all values ∈ [0,1]	Normalised vector produced within bounds	pass
Activity Scoring Service	Normalised feature vector	3 activity scores (pct_combat, pct_collect, pct_explore)	Scores computed with v2.2 divisors	pass
Clustering Service (IDW)	Activity scores + frozen centroids.json	3 soft membership weights. $\sum\mu_k=1.0$	Partition-of-unity constraint satisfied	pass
Temporal Delta Service	Current + prior soft membership. session timestamp	3 delta values. reset after 90 s timeout	Delta computation and reset verified	pass

MLP Inference Service	6-feature vector. weights from anfis_mlp_weights.json	Scalar raw output < 1 ms latency	Output produced < 1 ms per inference	pass
Calibration Service	Raw MLP output M	M = clamp(1+(raw−0.932) ×2, 0.6, 1.4)	M within [0.6, 1.4] on all edge cases	pass
Param Service	Bounded Mode 1 baseline from initial_parameters.json	Adapted PCG parameter set, JSON response assembled	Response contains all required fields	pass
Developer Dashboard (Next.js)	HTTP POST from dashboard test input	Pipeline output displayed. all 8 stages in sequence view	Dashboard renders all panels correctly	pass

Table 55: Module and Integration Testing - AURA Backend Pipeline

K.6. Soft Membership Heatmap

Soft membership heatmap supporting the clustering evaluation in *Section 8.3.3* in CHAPTER 08: TESTING.

Figure 65: Soft Membership Heatmap - Sample of 3,240 Windows (Self Composed) presents the soft membership heatmap across a random sample of 3,240 windows, confirming the IDW partition-of-unity constraint and validating that mixed behavioural signals are captured across all three archetypes.

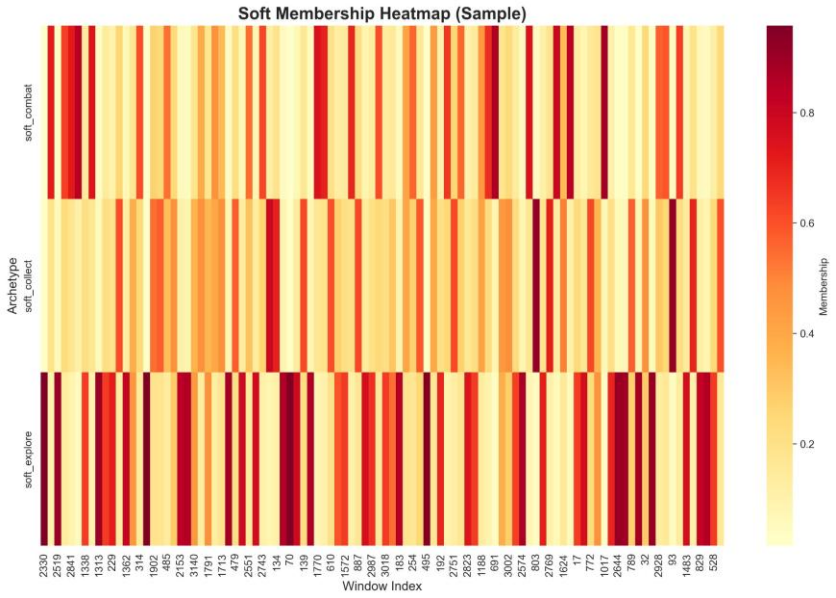


Figure 65: Soft Membership Heatmap - Sample of 3,240 Windows (Self Composed)

K.7. Temporal Delta Distributions

Temporal delta distributions supporting *Section 8.5.2* in CHAPTER 08: TESTING. Figure 66: Delta Distributions (Temporal Signals) Across 3,240 Windows (Self Composed) presents the delta distribution histograms for all three temporal signals (Δcombat, Δcollect, Δexplore) across the full dataset, confirming symmetric zero-centred distributions indicative of correct bidirectional temporal computation.

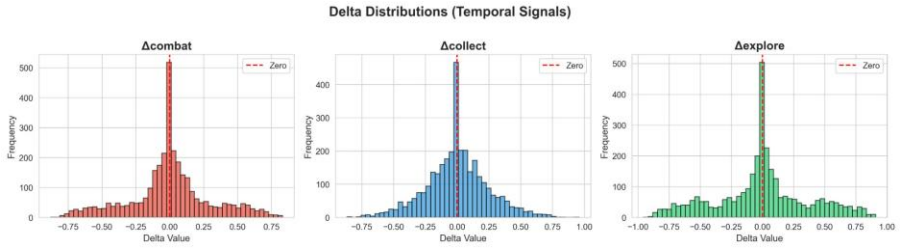


Figure 66: Delta Distributions (Temporal Signals) Across 3,240 Windows (Self Composed)

K.8. Full Residual Analysis - Test Set

Full residual analysis supporting the MLP surrogate evaluation in *Section 8.3.4* in CHAPTER 08: TESTING. The figure below represents the per-feature residual scatter plots for all six input features on the held-out test set. The absence of systematic structure across all six subplots confirms that the MLP surrogate does not exhibit model bias relative to any individual input

feature.

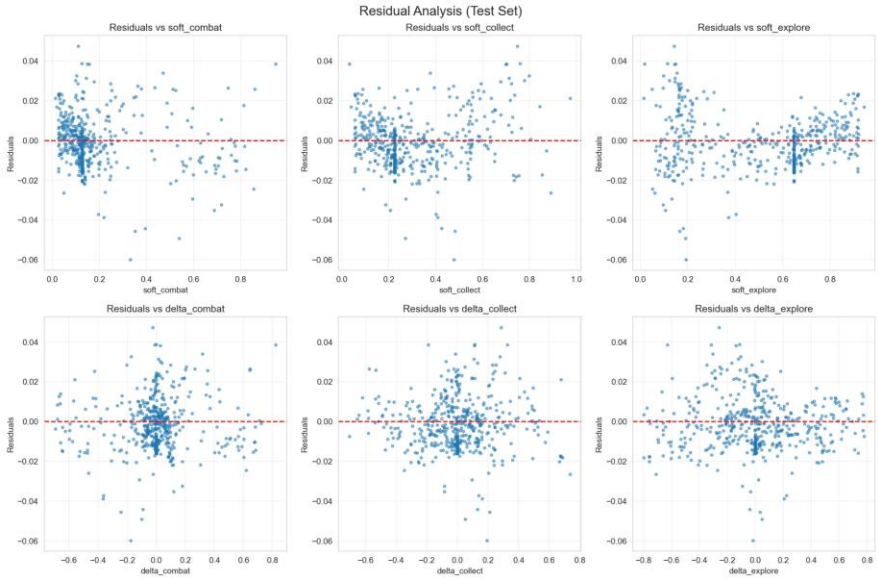


Figure 67: MLP Surrogate - Residual Analysis Across All 6 Input Features, Test Set (Self Composed)

K.9. Target Multiplier Distribution

Target multiplier distribution supporting the MLP surrogate evaluation in *Section 8.3.4* in CHAPTER 08: TESTING. The figure below presents the distribution of target multiplier values across the full 3,240-window dataset. The distribution is approximately unimodal with a mean of 0.902, confirming that training labels are centred below the neutral point of 1.0, consistent with the exploration-dominant behavioural profile of the CollectGame participant population.

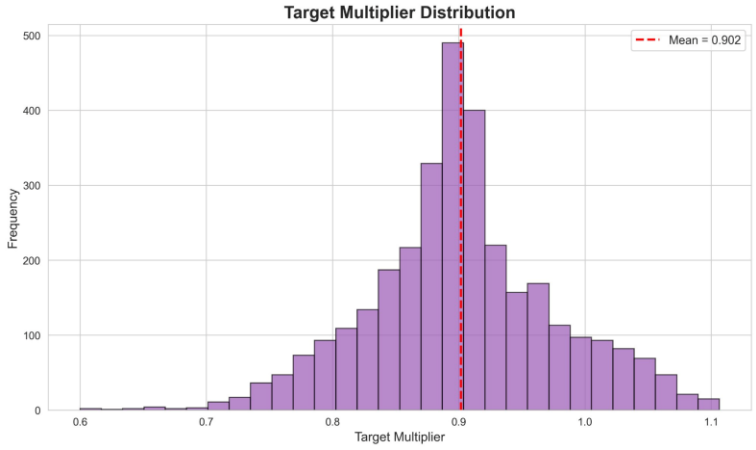


Figure 68: Target Multiplier Distribution - Full Dataset, mean = 0.902 (Self Composed)

K.10. Backend API Warm-Start Latency - CI Verification

CI verification of backend API warm-start latency supporting *Section 8.9.2* in CHAPTER 08: TESTING

The figure below presents the GitHub Actions continuous integration pipeline run confirming that the backend API warm-start latency satisfies NFR-01 under automated test conditions. The workflow, triggered via pull request on the CollectGame.Model repository, executed seven jobs in sequence across a total duration of 2 minutes 33 seconds: Build (46s), four concurrent API validation jobs (valid payload, missing userId, missing telemetry, empty body), the warm latency probe, and the Vitest unit test suite (19s). The API: Warm latency < 200ms job completed successfully in 48 seconds, confirming that five consecutive warm-run curl probes against the Render-deployed /api/pipeline endpoint each response within the 200ms threshold. All seven jobs completed with success status, as indicated by the green check marks on every job in the left panel and the workflow graph.

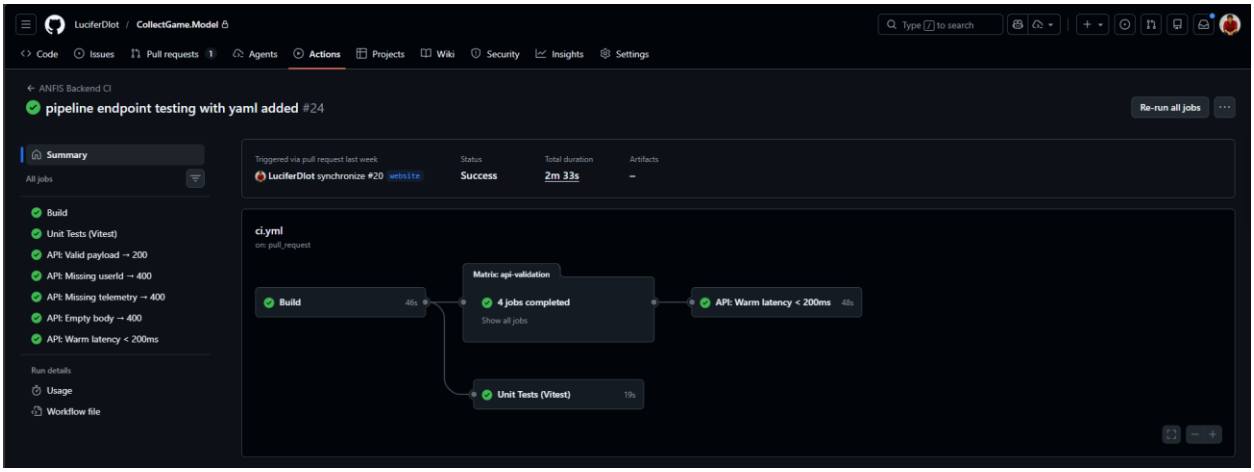


Figure 69: Backend API Warm-Start Latency - CI Verification (Self Composed)

As shown in above figure, the CI pipeline provides automated, commit-gated evidence that the backend warm-start API latency satisfies NFR-01. This measurement captures the round-trip time from the curl probe to the Render-deployed inference endpoint and back, excluding frontend JavaScript execution and React re-render overhead. The test is executed on every pull request to the main branch, providing regression protection against latency degradation across all future commits.

Appendix L - Evaluation

L.1. Evaluation Criteria

Extended self-evaluation supporting *Section 9.4*. The core six-criteria table is presented inline in *Chapter 09*. Below table presents the complete evaluation criteria used in this study, along with their corresponding objectives.

Criterion	Evaluation Purpose
The overall concept and novelty of the research project	To assess the originality of the calibration-first, telemetry-driven adaptive PCG approach and its contribution to the identified research gap in adaptive game systems.
The scope, depth, and complexity of the solution	To evaluate whether the research demonstrates sufficient academic rigour and technical ambition appropriate for final-year undergraduate research.
The design and architectural decisions of the proposed solution	To examine the soundness of the three-phase system architecture, the decoupled backend inference strategy, and the algorithmic design choices including soft membership and temporal smoothing.
Implementation of quality and technical efficiency	To assess the quality of the end-to-end pipeline, including MLP surrogate inference latency, bounded multiplier output, and unit test coverage.
Adaptive stability and behavioural alignment	To evaluate whether the framework produces stable, behaviourally coherent adaptation without oscillatory feedback or abrupt gameplay disruption.
Limitations and future improvement directions	To identify areas where the current implementation could be strengthened and to validate the future work directions proposed by the author.

Table 56: Evaluation Criteria

L.2. Remainder of Self-Evaluation

Below table presents the remaining criteria from the author's self-evaluation, supplementing the partial table in *Section 9.4*.

Criterion	Author's Self-Evaluation
Concept and novelty	The AURA framework addresses the gap between reactive DDA systems and opaque learning-based approaches by introducing a calibration-first, interpretable adaptive pipeline. The use of a neutral baseline derived from controlled calibration and the application of soft behavioural membership provide a structured and distinguishable contribution to adaptive game system design.
Scope, depth, and complexity	The project spans three fully implemented phases, including calibration, behavioural modelling, and backend deployment. The integration of data analysis, machine learning, and real-time system design demonstrates substantial technical depth and exceeds typical undergraduate expectations in both scope and engineering rigour.
Design and architectural decisions	The three-phase pipeline cleanly separates concerns: calibration provides the neutral reference, Phase 2 constructs the behavioural model, and Phase 3 handles real-time inference with no runtime retraining dependency. The multiplier bounds [0.6, 1.4], forward-only adaptation constraint, and temporal smoothing coefficient $\alpha = 0.2$ are principled design choices grounded in adaptive systems literature and validated against the Mode 1 calibration baseline. The implementation delivers functional PCG parameter adaptation across eleven variables in four categories. The acknowledged principal remaining limitation is the absence of a formal A/B evaluation of in-game adaptation quality, comparing adapted versus non-adaptive conditions using validated perceptual metrics.
Implementation of quality and technical efficiency	The MLP surrogate achieves Test $R^2 = 0.9264$ and MAE = 0.0127, confirmed by a bootstrap 95% CI of [0.874, 0.954]. Backend warm-start API round-trip latency below 200 ms under repeated request conditions, while cold-start latency on the hosting tier remained a separate platform-level limitation was confirmed across five CI probe measurements. The Vitest unit test suite and GitHub Actions CI provide automated regression protection across all pipeline stages. The primary acknowledged limitation is the absence of a formal controlled perceptual evaluation comparing adaptive and non-adaptive

	gameplay conditions.
Adaptive stability and behavioural alignment	Temporal smoothing and the bounded multiplier prevent oscillatory adaptation by responding to sustained behavioural trends rather than transient anomalies. The IDW partition-of-unity constraint ensures that no single archetype dominates inference without proportional telemetry evidence. The forward-only constraint maintains consistent player progression. Mode 1, selected via z-score NeutralityScore -0.7047, from the N=7 within-subjects calibration study, provides a stable reference ensuring that a behaviourally neutral player receives a multiplier of approximately 1.0.
Limitations and future improvement directions	The primary limitations are the calibration dataset scale (N=7), the Phase 4 UE PCG application for whole world's generation scope boundary, the Render free-tier cold-start delay (mitigated but not eliminated by the BeginPlay health check), the absence of parameter interpolation for smooth in-game transitions, and unit test coverage breadth. Each limitation is fully documented in Chapter 8 with a corresponding future mitigation strategy. These constraints define a clear roadmap for future AURA development rather than invalidating the current research contribution.

Table 57: Self-Evaluation - Continued

L.3. Selection of Evaluators

Below table presents the complete evaluator roster, including evaluator identifiers, names, positions, and affiliated organisations reviewed in *Section 9.5* of CHAPTER 09: CRITICAL EVALUATION.

Category	ID	Name	Position	Organisation
Domain Experts	DE-01	Udara Gunasekara	Product Manager - Gaming Services	Dialog Axiata PLC
Domain Experts	DE-02	Prasanth Sivakumar	Co-founder & CEO	code3x
Domain Experts	DE-03	Ravindu Cooray	Head of Studio	RAM Studios Pvt Ltd
Domain Experts	DE-04	Kavindu	Game Engineer &	SpaceTimer /

		Priyanath	Game Analyst	Friendly GameDev
Domain Experts	DE-05	Manul Singhe	Founder & CEO, Lecturer	Knight Owl, IIT
Technical Experts	TE-01	W.G. Yasintha Supun Madhushanike	Senior Game Developer	Chapelle Entertainment
Technical Experts	TE-02	Lakshan Ranasinghe	Lead Game Developer / Visiting Lecturer	Yunash / Java Institute
Technical Experts	TE-03	Chamath Nadeeshan	Game Programmer & Gameplay Engineer	Nine Hermits Games
Technical Experts	TE-04	Dakshina Wijayakulathilaka	Lead Game Developer	Mogo Games

Table 58: Selection of Evaluators

L.4. Detailed Evaluation Findings

Per-evaluator detailed findings supporting the thematic analysis in *Section 9.6.1* in CHAPTER 09: CRITICAL EVALUATION.

Below table presents the detailed per-evaluator feedback summaries collected through written email responses and structured Google Meet sessions.

ID	Evaluator	Criteria Addressed	Detailed Feedback Summary
DE-01	Udara Gunasekara	Concept & solution novelty. Industry-relevant design	Affirmed the framework as technically ambitious and well-structured, highlighting the movement beyond IF–THEN logic, the integration of ANFIS-based intelligent modelling, and the live backend as evidence of genuinely forward-thinking adaptive system design. Considered the work a well-executed project demonstrating both academic rigour and

			commercial applicability with clear potential for advanced research or commercial game AI.
DE-02	Prasanth Sivakumar	Technical soundness. dataset scale. forward-only adaptation	Confirmed the pipeline as technically sound and noted the calibration-first approach and ANFIS usage as academically distinctive. Recommended expanding the calibration cohort to 30–50 diverse players for production-directed validation. An initial characterisation of the 30-second window as a structured session mode was clarified by the author. Following clarification, Prasanth confirmed that forward-only adaptation is the preferred commercial approach as players expect game world consistency.
DE-03	Ravindu Cooray	Calibration design, soft membership, parameter interpolation, dataset scale	Described the calibration phase as the system's defining strength, specifically commended IDW soft membership as correct for handling overlapping player behaviour and validated the MLP surrogate R^2 of 0.9264 as demonstrating reliable runtime performance. The bounded multiplier and forward-only constraint were noted as vital safety rails. Identified parameter interpolation and calibration cohort scale as the two primary improvement priorities.
DE-04	Kavindu Priyanath	Behavioural analytics applicability, cross-	From a midcore hybrid game development and analytics standpoint, telemetry-driven behavioural clustering is directly applicable

		session adaptation	to studios distinguishing player types for engagement and retention optimisation. Affirmed the calibration-first design as solving a genuine production problem. Suggested extending the framework to accommodate cross-session behavioural drift as a significant advancement toward production-readiness.
TE-01	W.G. Yasintha Supun Madhushanike	End-to-end loop, temporal smoothing, telemetry window	Confirmed from shipped-game developer experience that the complete adaptive loop demonstrates genuine technical capability. Specifically noted temporal smoothing as a particularly important design decision, having observed adaptive features in commercial productions fail due to oscillation disrupting player flow. Validated the 30-second window as practical and assessed the overall system as technically sound and practically suitable for real adaptive game features.
TE-02	Lakshan Ranasinghe	Methodological rigour, phased design, interpretability, dataset justification	Conducted a detailed methodological review and confirmed the phased structure demonstrates systematic methodology rather than ad hoc development. Affirmed that the staged pipeline provides interpretability absent in black-box systems, which is especially valuable in a research context. Strongly recommended strengthening the justification of dataset design and validation choices in the final thesis to make the methodology fully

			convincing from an academic perspective.
TE-03	Chamath Nadeeshan	Challenge-mastery balance, system complexity, telemetry role	Acknowledged the established commercial role of telemetry-driven adaptive systems. Raised the broader design concern that highly adaptive systems risk diminishing challenge-mastery satisfaction, citing industry examples of players detecting and resisting algorithmic engagement systems. This concern applies to the adaptive games category generally. AURA's bound multiplier and forward-only constraint are its explicit design responses, preserving the core challenge trajectory within defined limits.
TE-04	Dakshina Wijayakulathilaka	UE architecture, decoupled inference, parameter interpolation, Phase 4 scope	From an Unreal Engine production perspective with five-plus years of experience, it is confirmed that decoupling ML inference from the game engine is architecturally sound, as in-engine inference would consume frame budget resources impractical for most productions. Validated the VaRest HTTP integration and BeginPlay warm-up health check as demonstrating sound systems-level thinking. Identified parameter interpolation as the primary implementation improvement priority and confirmed Phase 4 as the appropriate future scope boundary.

Table 59: Detailed Evaluation Findings per Evaluator

L.5. Focus Group - End-User Findings

Focus group end-user findings supporting in Section 9.6.4 in CHAPTER 09: CRITICAL EVALUATION. Below table presents the full focus group feedback from the two end-user participants. Participant names are withheld pending confirmation identifiers EU-01 and EU-02 are used throughout.

ID	Playstyle	Gameplay Experience	Usability Observations
EU-01 - Asel Hidallearachchi (Trainee Software Engineer)	Combat-dominant, aggressive approximately 25-minute session	Reported adaptation felt natural and progressive rather than reactive. Enemies became moderately more challenging over time in a manner that increased engagement without inducing frustration. No sudden difficulty spikes or empty level stretches were perceived. The gradual nature of the changes confirmed the temporal smoothing mechanism was functioning as intended from the player perspective.	Suggested a subtle in-game visual cue or post-session indicator that the system is responding to the player's style, observing that the adaptive pipeline's sophistication was not apparent during play without prior knowledge. This independently corroborates domain expert feedback regarding the absence of a player-facing transparency mechanism.
EU-02 - Vinu Kaveesha (Associate DevOps Engineer)	Exploration and collection-focused, casual player approximately 20-minute session	Reported a consistently comfortable difficulty gradient with no abrupt spikes of the kind that typically cause casual players to disengage. The beginning of the session was accessible enough to learn the controls without being overwhelmed. The	Suggested a personalised end-of-session summary feature indicating how the game adapted to the player's behaviour during the session, noting that players unaware of the adaptive system would not appreciate the work occurring beneath the

		dashboard demonstration post-session revealed that the volume of pipeline data per 30-second window was considerably more sophisticated than the in-game experience suggested, interpreted as evidence of the system's effective non-intrusiveness.	surface. This independently corroborates EU-01's suggestion and domain expert feedback.

Table 60: Focus Group End-User Findings

L.6. Evaluation of Functional Requirements

Functional requirements evaluation supporting *Section 9.8* in CHAPTER 09: CRITICAL EVALUATION. Below table presents the complete evaluation of all twelve functional requirements defined in Functional Requirements, including full descriptions, priority classifications, implementation status, and verification references.

FR ID	Priority	Requirement	Status	Verification
FR01	M	The system should collect player telemetry via a 30-second aggregation window and dispatch it to the backend via HTTP POST.	Implemented	VaRest HTTP POST in UE5. HTTP 200 confirmed. Vitest endpoint tests pass.
FR02	M	The backend shall validate, normalise, and process incoming telemetry against the 12-feature behavioural schema.	Implemented	HTTP 400 returned on malformed inputs. schema validation tests pass.
FR03	M	The backend shall compute per-archetype activity scores for Combat, Collection, and	Implemented	Activity scoring and IDW in Python pipeline and TypeScript backend.

		Exploration and apply IDW soft membership.		Vitest tests pass.
FR04	M	The backend shall perform MLP surrogate inference and apply neutral-centred calibration against the Mode 1 baseline.	Implemented	MLP R^2 = 0.9264. mlp_neutral = 0.932006. calibration formula verified by Vitest.
FR05	M	The backend shall bound the final multiplier within [0.6, 1.4] and enforce forward-only adaptation constraints.	Implemented	Clamp and forward-only logic verified. all Vitest edge-case tests pass.
FR06	S	The system shall visualise fuzzy rule activations and inference outputs for explainability.	Partially Implemented	Developer analytics dashboard fully implemented and deployed at Vercel. In-engine developer configuration panel remains a future enhancement
FR07	M	The system should update PCG parameters dynamically based on backend inference service outputs.	Implemented	A bounded multiplier is applied to 11 PCG parameters across combat, exploration, collection, and global categories via Blueprint updates every 30 seconds.
FR08	M	The system shall apply environmental updates within 200 milliseconds latency.	Implemented	Sub-200 ms warm start confirmed across 5 CI probe measurements.
FR09	S	The system should monitor performance metrics and revert to the previous configuration	Partially Implemented	PCG parameters are applied every 30 seconds with instantaneous

		when thresholds are exceeded.		updates. Smooth interpolation is not yet implemented. Forward-only application is maintained across all parameters.
FR10	S	The system should provide an editable in-engine configuration panel for the developer.	Future Scope	Phase 4 UE integration of in-engine configuration panel feature. not yet implemented.
FR11	M	The system shall record all telemetry, cluster transitions, MLP outputs, and adaptations and export session reports.	Implemented	Developer dashboard records and displays all intermediate values per session.
FR12	C	The system shall allow importing and exporting of configuration templates through the Unreal Engine interface.	Future Scope	Phase 4 feature. C-priority. identified as future work.

Table 61: Evaluation of Functional Requirements

L.7. Evaluation of Non-Functional Requirements

Non-functional requirements evaluation supporting Section 9.8. Below table presents the complete evaluation of all eight non-functional requirements defined in Non-Functional Requirements.

NFR ID	Requirement	Priority	Status	Verification Notes
NFR1	Performance	M	Implemented	warm-start API round-trip latency below 200 ms under repeated request conditions (while cold-start latency on the hosting tier remained a separate platform-level limitation) confirmed across 5 CI probe

				measurements. MLP inference under 1 ms.
NFR2	Quality of Adaptation	M	Implemented	Test R^2 = 0.9264. bootstrap CI [0.874, 0.954]. bounded multiplier [0.6, 1.4] verified by Vitest.
NFR3	Efficiency	S	Implemented	MLP inference under 1 ms. stateless per-request backend model. No runtime retraining.
NFR4	Usability	S	Partially Implemented	Developer analytics dashboard fully implemented and deployed at Vercel. The in-engine developer configuration panel remains a future enhancement.
NFR5	Scalability	S	Implemented	MVC + Repository + Service pattern. frozen JSON artefacts allow model updates without code changes.
NFR6	Reliability	M	Implemented	Session timeout and fallback verified by Vitest. GitHub Actions CI regression protection active.
NFR7	Maintainability	S	Implemented	Three-repository modular structure. MVC pattern. documented codebase with README files.
NFR8	Security	S	Partially Implemented	Anonymised telemetry GDPR-compliant consent obtained. Endpoint encryption is future enhancement.

Table 62: Evaluation of Non-Functional Requirements

Appendix M - Conclusion

M.1. Status of Research Objectives

Research objectives status as referenced in *Section 10.2.2* in *Chapter 10*.

Table 63: Status of Research Objectives presents the status of each of the fourteen research objectives stated in *Section 1.11*, together with the phase, description, method of achievement, and final status.

ID	Phase	Description	How Achieved	Status
R01	Problem ID.	Identify and formalise limitations of existing adaptive PCG systems	• Systematic literature review in Ch2. • Cold-start bias, hard clustering, and interpretability gaps formally identified and documented	Achieved
R02	Problem ID.	Define a calibration-first adaptive architecture	• Three-phase AURA architecture formally defined in Ch3 methodology and Ch6 design. • Phase-separation enforced throughout implementation	Achieved
R03	Lit. Review	Conduct comprehensive literature review on DDA, APCG, and telemetry-driven modelling	• Ch2 critically evaluates rule-based DDA, RL-PCG, and neuro-fuzzy systems across 20+ sources. • Concept map and gap analysis completed	Achieved
R04	Lit. Review	Investigate soft clustering and neuro-symbolic reasoning models	• IDW soft membership, K-Means centroid modelling, and ANFIS hybrid architecture reviewed and selected in Ch2 and Ch6	Achieved
R05	Req. Elicit.	Identify system-level requirements	• 12 functional requirements (FR01–FR12) and 8 non-functional	Achieved

		for real-time adaptive control	requirements defined in Requirements. • MoSCoW prioritisation applied	
R06	Design	Design a three-phase adaptive architecture (Calibration, Telemetry, Integration)	• Full architecture designed in Ch6 covering Phase 1 calibration pipeline, Phase 2 behavioural modelling, and Phase 3 backend inference service	Achieved
R07	Design	Design telemetry feature engineering and temporal modelling strategy	• 30-second telemetry window, 12-feature schema (10 raw + 2 derived), and temporal delta modelling (α=0.2 smoothing) designed in Ch6	Achieved
R08	Impl.	Implement behavioural clustering using soft membership modelling	• K-Means (K=3, silhouette=0.3752) implemented in CollectGame.Model. • IDW soft membership with partition-of-unity constraint ($\Sigma\mu_k$=1.0) verified	Achieved
R09	Impl.	Implement neuro-fuzzy surrogate reasoning layer	• ANFIS offline trained across 3,240 windows • MLP surrogate (6→16→8→1) trained as runtime approximation achieving R^2=0.9264, MAE=0.0127	Achieved
R10	Impl.	Integrate backend-mediated adaptive inference with Unreal Engine	• VaRest HTTP integration implemented in UE5.7. multiplier received and applied to MaxHealth and StaminaRegen. • PCG parameter adaptation is	Achieved

			implemented in Phase 3, with eleven parameters across four categories adjusted by a bounded multiplier every 30 seconds.	
R11	Testing	Evaluate adaptation stability and boundedness	• Multiplier bounding [0.6, 1.4] verified. • Forward-only adaptation constraint validated. • Oscillation absence confirmed across test sessions in Ch8	Achieved
R12	Testing	Evaluate performance impact and latency	• Backend round-trip latency confirmed <200 ms (warm-start, Render free-tier). • MLP inference <1 ms. • FPS stability validated in Ch8	Achieved
R13	Testing	Evaluate behavioural interpretability	• Archetype proportions (Combat, Collection, Exploration) exposed via Next.js dashboard • Expert evaluators confirmed interpretability of adaptive decisions in Ch9	Achieved
R14	Dissem.	Document and disseminate findings through academic publication	• Thesis documentation is complete. A public dataset and reproducible pipeline have been published on Kaggle, ensuring open access to the empirical workflow. Submission of a peer-reviewed research paper is planned as immediate post-submission work.	Partially Achieved

Table 63: Status of Research Objectives

M.2. Utilisation of Knowledge from the Degree

Knowledge utilisation mapping supporting in *Section 10.3* in CHAPTER 10: CONCLUSION. Table 64: Utilisation of Knowledge from the Degree presents the full mapping of degree modules to the pipeline components and artefacts they informed within the AURA project.

Module Name	Knowledge Applied	Where Used in AURA
Machine Learning and Data Mining	Supervised learning, regression modelling, feature engineering, model evaluation metrics (R^2, MAE), cross-validation, and bootstrap confidence interval analysis	• Phase 2: MLP surrogate design, training, and evaluation • K-Means clustering. • MinMaxScaler normalisation (CollectGame.Model)
Algorithms: Theory, Design and Implementation	Time complexity analysis, distance-based algorithms, optimisation heuristics, and structured algorithm design methodology	• IDW soft membership computation • Grid search over 108 K-Means configurations • Epsilon stabilisation for precision drift
Object Oriented Programming	OOP principles: encapsulation, abstraction, inheritance, and interface-driven design for maintainable, testable code	• Phase 3 backend architecture (MVC + Repository + Service pattern) • TypeScript class hierarchy for telemetry pipeline components
Server-Side Web Development	RESTful API design, HTTP request/response lifecycle, JSON serialisation, middleware patterns, and Express.js routing	• Phase 3: Node.js/Express.js telemetry backend deployed on Render • VaRest HTTP integration with Unreal Engine 5.7
Mathematics for Computing	Linear algebra (vector operations, distance metrics), probability theory, statistical	• IDW membership formula • NeutralityScore composite metric • ANFIS training in mathematics

	distributions, and numerical methods	• Bootstrap confidence interval computation
Software Development I and II	Structured programming, debugging methodology, software lifecycle fundamentals, and iterative development practices	• Iterative versioning protocol (v1.0→v2.2.1) across three repositories • Systematic debugging during v2.0→v2.1 archetype bias correction
Software Development Group Project	Team-based software delivery, project scoping, milestone planning, and managing technical deliverables under deadline pressure	• Applied to single-developer multi-phase project management • Kanban board coordination across CollectGame.CalibrationAnalysis, CollectGame.Model, and CollectGame.Telemetry
Database Systems	Data modelling, persistence strategies, query design, and structured data schema definition	• Telemetry data schema design • JSON artefact structure for frozen model weights (scaler_params.json, anfis_mlp_weights.json, centroids.json)
Web Design and Development	Frontend architecture, component-based UI design, and responsive web application development	• Next.js 16 App Router analytics dashboard (Vercel) • Real-time Recharts visualisations of pipeline telemetry and multiplier output
Formal Methods	Specification formalisation, invariant definition, and structured constraint expression in system design	• Forward-only adaptation constraint formalization • Multiplier bounding invariant [0.6, 1.4] • Partition-of-unity constraint

		($\Sigma\mu_k$=1.0)

Table 64: Utilisation of Knowledge from the Degree

M.3. Achievement of Learning Outcomes

Learning outcomes achievement table as referenced in *Section 10.6* in CHAPTER 10: CONCLUSION Table 65: Achievement of Learning Outcomes presents the achievement of all eight learning outcomes for the BEng Final Year Project module, with evidence drawn from the AURA project artefacts and deliverables.

LO	Official Learning Outcome Description	Evidence in the AURA Project
LO1	Select, justify, and apply appropriate methods, techniques, and tools to effectively address the challenges of the individual project	• ANFIS selected for fuzzy surface training • MLP surrogate selected for runtime inference based on latency analysis • K-Means with IDW soft membership selected over hard clustering through silhouette-guided grid search across 108 configurations • Vitest selected for automated regression testing
LO2	Plan, manage, and monitor a substantial individual project within defined timelines, milestones, and scope boundaries	• Kanban board maintained across three repositories over the full project duration • Versioning protocol (v1.0→v2.2.1) applied systematically • Phase 4 formally descoped and documented as future work when timeline constraints were identified
LO3	Identify and address system requirements considering social, legal, ethical, and professional dimensions of the project	• 12 functional requirements (FR01–FR12) and 10 non-functional requirements elicited using MoSCoW prioritization • SLEP analysis conducted in Ch5 addressing data consent, participant confidentiality, BCS Code of Conduct, and professional deployment

		standards
LO4	Conduct an extensive and critical literature review to evaluate existing work within the relevant research domain	• Ch2 critically reviews DDA, RL-PCG, telemetry-driven behavioural modelling, and neuro-fuzzy control across 20+ peer-reviewed sources • Research gaps formally identified and mapped to AURA's architectural contributions
LO5	Design, implement, and validate a technical solution through structured, modular, and iterative development practices	• Three-phase pipeline implemented across three repositories with independent validation at each phase boundary • 62-test automated suite (Vitest v1.6.1) provides regression-protected validation • GitHub Actions CI/CD pipeline enforces continuous integration
LO6	Communicate findings clearly and effectively through structured academic writing and formal documentation	• Thesis documented across ten chapters following Westminster formatting standards • Harvard referencing applied throughout • live analytics dashboard (https://collect-game-website.vercel.app/) provides communicable system observability for research community
LO7	Apply systematic problem-solving strategies to identify, analyse, and resolve technical challenges encountered during project development	• Four implementation challenges (ANFIS latency, archetype bias, cold-start, IDW precision drift) identified and resolved through structured root-cause analysis and targeted engineering fixes • Each resolution documented in Ch7
LO8	Demonstrate professional standards in research documentation, ethical conduct, and dissemination of	• Calibration participant consent obtained • Evaluator anonymisation applied where required • Open-source repositories maintained with

	results	CI/CD and version history • Thesis prepared for academic submission adhering to University of Westminster formatting guidelines

Table 65: Achievement of Learning Outcomes

Appendix N - ANFIS-Inspired Canonical Formula: Supporting Evidence

This appendix provides the supporting evidence for the canonical ANFIS-inspired adaptation surface used in AURA. It contains material referenced but not detailed in the main chapters: the original failure statistics (supporting *Section 7.3.2* and *Section 8.4*), the full formula version history, the coefficient-level fuzzy mapping, the Pearson correlation evidence underpinning the delta-dominant weighting, and the design decisions catalogue listing each rejected alternative. The formula itself and the neutral-centred calibration scheme are described in *Section 6.5.3* and *Section 7.3.2* this appendix provides the evidence trail behind those descriptions.

N.1. Original Formula Failure - Variance Collapse Evidence

The original target formula relied on a single feature (death count) and produced a near-constant target distribution. Below table presents the diagnostic statistics that confirmed variance collapse.

Metric	Value	Status
Min	1.0000	Unhealthy - no lower bound reached
Max	1.0227	Unhealthy - effectively constant
Mean	1.0101	Unhealthy - no variation
Std Dev	$\sigma = 0.0113$	CRITICAL - insufficient for gradient descent
Span	0.0227	CRITICAL - 2.3% of clamp width used
Upper clamp saturation	100%	CRITICAL - all samples at ceiling
MLP Test R^2	−4.69	FAIL - worse than predicting the mean

Table 66: Target distribution - original formula

The failure was statistical, not architectural. The MLP architecture (6→16→8→1) was sufficient. The same architecture achieved $R^2 = 0.9264$ once the target signal was redesigned. The root cause was that soft membership values are bounded [0, 1] and sum to 1.0, and the original formula had no mechanism to generate variance beyond the narrow death-count penalty

N.2. Full Formula Version History

Below table traces all six formula iterations from the original failure to the approved v2.2.1. Only the final row is deployed.

Version	Base	Soft coefs	Delta coefs	Death	Target σ	Span	R^2	Outcome
Original	1.0	-	-	-0.1 (deaths only)	0.0113	0.023	−4.69	FAILEd variance collapse
Option A	1.0	0.30/0.25/0.20	-	−0.40	~0.020	-	-	Insufficient σ, still no deltas
Option B v1	1.0	0.15/0.12/0.10	0.50/0.35/0.30	−0.20	0.058	0.38	-	Mean too high (1.02)
Option B v2	0.9	0.15/0.12/0.10	0.50/0.35/0.30	−0.20	0.060	-	-	Suboptimal coefficients
v2.2 Canonical	0.9	0.22/0.18/0.15	0.55/0.40/0.35	−0.25	0.062	0.41	0.9391	Biased below neutral - deployed but incorrect
v2.2.1 FINAL	1.0	0.22/0.18/0.15	0.55/0.40/0.35	−0.25	0.074	0.507	0.9264	APPROVED - semantically correct

Table 67: Complete formula version history

Soft coefficients apply to centered features (value − 0.5) from Option B v1 onwards. Improvement from v2.2 to v2.2.1: base corrected from 0.9 to 1.0. R^2 decreased slightly (0.9391 → 0.9264) because the corrected distribution is wider and more symmetric, making it harder to fit. This decrease is expected and correct

N.3. Coefficient-Level Fuzzy Mapping - Why Each Term Exists

The seven terms in the canonical formula each correspond to a component of Takagi-Sugeno fuzzy

inference. Below table documents this mapping and the rationale for each coefficient value.

Term	Coefficient	Fuzzy-system role	Design rationale
Base	1.0	Default defuzzification output	A balanced player with no behavioural trend receives M = 1.0 (neutral). Base encodes this semantic prior. Earlier base of 0.9 caused systematic downward bias - see N.4.
soft_combat − 0.5	0.22	Takagi-Sugeno consequent - combat archetype	Centering at 0.5 creates bidirectional contribution: values above 0.5 push M up (harder), below push down (easier). Highest soft coefficient because combat intensity is the most direct challenge signal.
soft_collect − 0.5	0.18	Takagi-Sugeno consequent - collection archetype	Same centering. Lower than combat (0.18 vs 0.22) because collection-dominant play shows weaker correlation with required difficulty change.
soft_explore − 0.5	0.15	Takagi-Sugeno consequent - exploration archetype	Lowest soft coefficient: exploration behaviour is the least predictive of challenge adjustment need. Note: Δexplore has the strongest Pearson r but as a delta, not a static membership.
Δ_combat	0.55	Adaptive network update rule - combat momentum	Primary variance driver. Pearson r = −0.471 between Δcombat and target change. Highest coefficient because combat acceleration is the strongest session-level signal for difficulty need.
Δ_collect	0.40	Adaptive network update rule - collection momentum	Secondary variance driver. Captures players shifting toward or away from collection engagement.
Δ_explore	0.35	Adaptive network update rule -	Tertiary variance driver. Note: Pearson r = 0.808 between Δexplore and Δtarget is the

		exploration momentum	strongest single correlation in the dataset - the coefficient 0.35 is lower than 0.55 to balance the asymmetric correlation signal.
death_rate	−0.25	Safety mechanism - defuzzification penalty modifier	Reduces M when death rate is elevated, assisting struggling players. Weight chosen so a death rate of 0.5/window (one death every 60 seconds) contributes −0.125 to M - noticeable but not overwhelming.

Table 68: Coefficient-level mapping to Takagi-Sugeno fuzzy inference components

N.4. The Structural Offset - Why Base 0.9 Was Incorrect

The partition-of-unity constraint (soft memberships always sum to 1.0) creates a fixed structural negative offset whenever the soft membership terms are centered. For a balanced player with equal memberships (1/3, 1/3, 1/3) and no deltas:

$0.22 \times (1/3 - 0.5) = 0.22 \times (-0.167) = -0.037$

$0.18 \times (1/3 - 0.5) = 0.18 \times (-0.167) = -0.030$

$0.15 \times (1/3 - 0.5) = 0.15 \times (-0.167) = -0.025$

Fixed offset = −0.092

This offset is structural. It exists for every possible membership combination because the partition-of-unity constraint ensures the weighted average of centered terms is always negative when weights are unequal. With base = 0.9, the balanced-player training target was 0.9 − 0.092 = 0.808. The MLP therefore learned that all realistic inputs map to sub-neutral outputs, making it impossible to ever predict a harder-than-neutral result without large positive deltas.

Correcting to base = 1.0 raises the balanced-player training target to 0.908. The MLP's actual neutral output (mlp_neutral = 0.932006, stored in anfis_mlp_weights.json) reflects this, it is close to 1.0 but not exactly 1.0 due to the structural offset. The neutral-centred calibration formula (documented in Chapter 6 §6.5.3) corrects for this remainder, guaranteeing display = 1.0 for the balanced player input regardless of the offset magnitude.

N.5. Pearson Correlation Evidence for Delta-Dominant Weighting

The decision to assign the highest coefficients to delta features rather than soft membership values was empirically motivated. Table N.4 presents the Pearson correlation analysis conducted on the full 3,240-window dataset.

Feature	**Pearson r with Δtarget**	**Interpretation**	**Weight assigned**
Δexplore → Δtarget	r = 0.808	Very strong positive - direction of exploration change predicts target change most reliably	0.35 (note: lower than 0.55 to balance asymmetry)
Δcombat → Δtarget	r = −0.471	Moderate negative — combat increase signals difficulty increase	0.55 (highest delta coefficient)
Δcollect → Δtarget	r = moderate	Weaker than combat/explore but consistent	0.40 (secondary)
soft_combat (static)	Lower than deltas	Context signal — where the player is, not where they are going	0.22 (highest soft)
soft_explore (static)	Lower than deltas	Static membership less predictive of needed adjustment	0.15 (lowest soft)

Table 69: Pearson correlation evidence underpinning the delta-dominant weighting design

Source: research archive correlation analysis, development phase.

The key finding is that delta features have higher predictive power than static membership features for the adaptive control task. A player currently showing 80% combat membership but trending away from combat (Δcombat = −0.4) requires a fundamentally different response than a player with 80% combat and a positive delta. Static membership alone cannot distinguish these cases, delta features are structurally necessary.

N.6. Calibration Verification Scenarios

The following scenarios were used to verify that the neutral-centred calibration formula (*Section 6.5.3*) produces semantically correct outputs. All 19-unit tests in collect-game-website pass against these expected values.

Input scenario	soft (C/O/E)	deltas	death_rate	Raw MLP	Display M	Correct?
Balanced / neutral	1/3, 1/3, 1/3	0, 0, 0	0	0.932	1.000	Neutral
High combat, rising	0.85, 0.10, 0.05	Δcombat=+0.3	0	~1.11	1.127	HARDER by 12.7%
High explore, rising	0.15, 0.15, 0.70	Δexplore=+0.3	0	~0.87	0.829	easier by 17.1%
Collector, rising	0.10, 0.80, 0.10	Δcollect=+0.3	0	~0.88	0.865	easier by 13.5%
Struggling player	/	combat=−0.4	0.5	~0.85	0.840	easier by 16.0%

Table 70: Calibration verification scenarios

Display M computed using clamp(1.0 + (raw − 0.932006) × 2.0, 0.6, 1.4).
For derivation of mlp_neutral and amplification constant, see *Section 6.5.3*.

N.7. Design Decisions - Rejected Alternatives

Below table documents the seven principal design decisions for the canonical formula, together with the alternative that was considered and the reason it was rejected.

Decision	Chosen	Alternative considered	Reason alternative was rejected
Base value	1.0	0.9	Base 0.9 caused systematic downward bias. training neutral point = 0.808, MLP never

			learned harder-than-neutral outputs (see N.4. The Structural Offset - Why Base 0.9 Was Incorrect).
Membership centering	Subtract 0.5 (centered)	Use raw values	Raw values are always positive [0, 1], creating only upward pressure on M. Centering at 0.5 creates symmetric bidirectional influence aligned with fuzzy IF-THEN semantics.
Primary variance drivers	Delta coefficients (0.55, 0.40, 0.35)	Higher soft membership coefficients	Static memberships sum to 1.0. Increasing their coefficients cannot overcome the variance ceiling. Deltas are unbounded and provide the entropy needed for gradient descent.
Death penalty coefficient	−0.25	Higher penalty (−0.5 or above)	A death rate of 0.5/window under a −0.5 coefficient would reduce M by 0.25. Large enough to produce trivially easy gameplay after a few deaths. The lower coefficient −0.25 assists struggling players without trivialising difficulty.
Output clamp	[0.6, 1.4]	[0.5, 1.5]	Expert evaluators identified M = 1.5 as 'unfair' during Chapter 9 evaluation. M = 0.5 was identified as trivially easy. The narrower clamp preserves the challenge-mastery balance required by Flow theory.
Calibration method	Neutral-centred (mlp_neutral anchor)	Min-max rescaling using output range	Min-max using extreme delta inputs (±1.0) pulled min_raw to 0.49. Since realistic players produce deltas in [−0.3, +0.3], every real session appeared above the midpoint → always HARDER. Same symptom as original bias, opposite direction. Semantic anchoring is input-distribution-independent.

Amplification constant	Fixed 2.0	Empirically tuned per-retrain value	A fixed constant ensures consistent display range behaviour across retrains. Empirical tuning would require manual recomputation each time notebooks 06-07 are re-executed.

Table 71: Design decisions catalogue