University of Nottingham
UK | CHINA | MALAYSIA

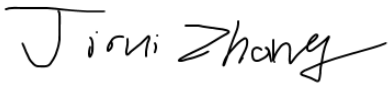
Generative AI for In-game Dynamic Non-playable Characters Creation and Adaptation

Submitted **December 29, 2025**, in partial fulfilment of the conditions for the award of the degree **BSc Computer Science with Artificial Intelligence**.

Jirui Zhang 20513930
Supervisor: **Jeremie Clos**
School of Computer Science
University of Nottingham

I hereby declare that this dissertation is all my own work, except as indicated in the text:

Signature



Date: December 29, 2025

I hereby declare that I have all necessary rights and consents to publicly distribute this dissertation via the University of Nottingham's e-dissertation archive.*

Academic Year: 2025–2026

CONTENTS

ABSTRACT

***Abstract*—This project investigates how natural-language descriptions might drive real-time NPC creation in a Vampire Survivors–style Unity prototype. This paper suggests a text-to-NPC pipeline mapping descriptions to NPC attributes, visuals, and behaviours via schema-constrained prompting, JSON specifications, and engine-side binding. An asynchronous C# wrapper embeds a locally deployed Qwen3-VL model. The system enables robust, real-time attribute tuning of NPCs from text at this stage, supporting a solid base for future work on sprites, behavioural generation, and user-centred evaluation.**

Generative AI for In-game Dynamic Non-playable Characters Creation and Adaptation

Jirui Zhang[1] Jeremie Clos[1]

[1]University of Nottingham

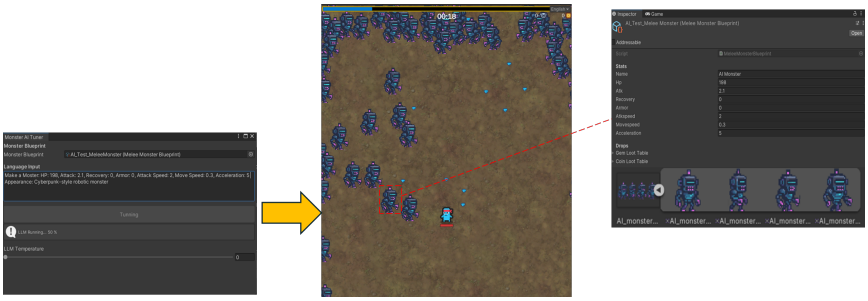


Fig. 1: **AI-Driven Monster Generation System.** An interactive pipeline where language-guided tuning updates monster attributes and appearance, and the modified blueprint is deployed directly into the game.

I. Introduction

A. Background

Non-Playable Characters (NPCs) are central to player experience in modern video games. NPCs have traditionally been developed using deterministic techniques such as Finite-State Machines (FSMs) and Monte Carlo Tree Search (MCTS). Their construction relies heavily on specifying explicit code-level behavior [1]. Though these control mechanisms produce both predictability and computational efficiency, NPCs are controlled entirely by models, animations, and behaviour scripts, owing to NPCs' complexity and sensitivities to immersion [2], [3]. As a result, NPCs exhibit repetitive behaviour, which reduces their expressive potential; they exhibit limited responsiveness in terms of flexibility and emotional range [4]. Furthermore, the authoring of such rule-based control systems demands considerable technical expertise, which remains complex and error-prone for developers [5]. To overcome these restrictions, generative methodologies that produce character information-rich in both content and capabilities—while substantially reducing the technical expertise required of developers—have gained interest.

With this motivation in mind, certain areas of Generative AI are where automation becomes attractive for NPC production. An additional recent review [6] reveals that existing generative models can generate a variety of assets for characters, including visual appearances [7], narrative backstories [8], and high-level behavioural descriptions [9] directly from natural language prompts. This substantially reduces the technical barrier for developers. However, this is only illustrative of the growing capability of AI-assisted character authoring. Although many such generation processes are performed offline, the resulting outputs are not particularly engine-ready. For example, produced meshes will require retopology, texture correction, rigging, or other optimisation processes before they can be integrated into a real-time asset pipeline [10].

In addition, Generative AI has also spurred developments in in-game NPC interaction. Incorporating LLMs into game engines enables the NPCs to process dynamic and context-aware dialogue, which exceeds preset script trees [11].On this backswing, some recent research using Gemini and Sentence-BERT has shown that NPCs generate richer dialogues; on average, in response to player inputs, they will even produce text-based quests [12]. The resulting immersion approaches take it a step further, giving NPC language the potential to be human-centred. However, they remain productive only for language generation (and will always be constrained by the same constraints), but do not actually produce or modify the complete multimodal architecture of an NPC (i.e., models/animations, behaviour structure, executable logic).

While some advancements have been made in AI-based character generation and LLM-based real-time interaction, current research still treats both as distinct. Note that there is no single pipeline that connects natural language input to the real-time synthesis and deployment of a playable NPC (this capability is not supported in current systems). The marriage of both generative and interactive techniques in a combined end-to-end workflow to generate, instantiate, and adapt NPCs during gameplay is seldom even considered.

B. Motivation

a. Research Motivation

Generative AI has established that models can generate many of the basic components necessary to construct an NPC: visual assets [7] and character animations [13], behavioural specifications, and executable logic [14]. Nonetheless, these capabilities are not used alone, and current AI-assisted NPC creation, particularly, is in a more scattered model [6]. For this reason, none of these processes act together in the most advanced or optimal way to produce a fully developed, playable agent. This fragmentation hinders generative AI from harnessing its potential as a multimodal synthesis framework capable of generating coherent, embodied entities.

Thus, the investigation of an integrated approach to text-to-NPC generation clearly has research value as a critical exploration of the upper limit of AI multimodal generative capabilities, as well as a preliminary step toward holistic agent synthesis. Other similar integrated capabilities have been verified in fields such as web interface synthesis [15] and autonomous software agent development [16], while the game domain remains untouched. This research will first examine whether natural-language intent can lead to the complete generation of a functional NPC. The final conclusion will contribute to larger questions regarding how generative AI can transition from asset-level generation to a holistic, controllable, and embodied system synthesis.

Besides multimodal integration, current applications of generative AI in games mainly work on low-complexity text-only tasks as well. Typical current runtime integration is centred on LLM-based dialogue [12], branching narrative variety [17], or quest text generation [18]. Although useful, these applications are limited to evaluating the performance of generative AI on simple text-based tasks in real-time gaming environments. They fail to address whether generative AI can still perform well within a real-time game loop when faced with complex, multimodal tasks. Hence, we address this research gap through this project by assessing generative AI under very high-end runtime conditions: building, instantiating, and managing fully functional NPCs during gameplay. This allows for more stringent real-time evaluation of generative AI's performance, robustness, and applicability to interactive systems.

b. Technical challenge

Implementing text-to-NPC generation in a real-time game context raises many technical challenges.

First, the system should be able to bring together various components of NPC design: 3D models, animations, behaviours, and working code, within natural-language descriptions, and convert them into an engine-compatible structure. This requirement poses a major technical challenge because existing AI methods used in games primarily focus on dynamic text generation and narrow-scope procedural content [19], and do not facilitate the multimodal integration required to build engine-ready NPCs.

Secondly, the whole flow must run under very strict real-time constraints: it must run during play or with very small loading windows. Keeping latency minimal is essential because even small latency delays affect responsiveness, in effect, player performance, and the perceived coherence of interactive environments [20]. However, recently reported studies indicate that even dialogue-only AI-enabled NPC systems suffer around 7s average cycle latencies and that cycle latency, after some delay, increases further with context length [21]. These results shed light on practical constraints of real-time interaction and stress the importance of excellent algorithmic efficiency and system integration.

c. Practical Impact

For the AI industry, games have historically been used as a controlled laboratory to evaluate algorithms in contexts corresponding to everyday scenarios, such as dynamic decision-making, uncertainty management, and multi-agent cooperation [22]. However, the majority of current platforms limit AI assessment to narrowly targeted agent behaviours. This paper takes this trend further by developing a solution for real-time NPC generation within this model, providing a far more open and challenging AI testing environment.

For the gaming industry, it would provide an avenue for simpler, cheaper NPC generation, as opposed to the laborious classical approaches; player-customisable NPCs have always been a significant challenge for developers. The proposed method will allow users to generate unique characters in real time using only natural-language descriptions when being implemented. The ability to generate user-generated content has been proven to positively affect player engagement, diversify content, and prolong the game [23].

C. Aim & Objectives

The aim of this project is to prove that real-time, generative-AI-powered NPC creation can be performed. It looks to allow players to generate NPCs—with unique appearances and behaviours—right from their natural-language descriptions, and to interact with them in a live game environment.

This project is mainly motivated by the following objectives:

1) **Survey & Scope:** Examine the state of the art in text-to-NPC, LLM-in-games, and runtime procedural generation to map out project scope, toolchain, and evaluation metrics.
2) **Runnable Game Prototype:** Design a small level prototype of a game to support the testing.
3) **Text-to-NPC Architecture:** Develop an end-to-end pipeline (prompting $\rightarrow$ parsing $\rightarrow$ asset/animation generation $\rightarrow$ behavioural logic $\rightarrow$ runtime injection) with clear interfaces and data schemas.
4) **LLM Deployment & Integration:** Instill an LLM (local/server) and interoperate with the engine; define prompt/response schema, latency targets, caching, safety guardrails; expose APIs for dialogue and decision-making.
5) **Real-time NPC Generation:** Instant image assets, attack sequences, and behaviour trees/graphs in natural language, making them digestible and loadable to the engines.
6) **Evaluation:** A technical and a user-based evaluation will be made of the system. This will involve assessing performance metrics and conducting playtests to collect feedback about immersion, usability, and variety in gameplay. The findings will be analysed to determine their strengths, where they can improve, and when they need to be revisited.

II. Related Work

Procedural Content Generation (PCG) [24], [25] is a foundational research area for algorithmically generating game content with limited user input. Therefore, NPC generation can be viewed as a part of PCG. The traditional search-based approach [26], [27] uses a stochastic search/optimisation algorithm to search for content that meets the requirement. The evaluation algorithm [28] is the most common in this traditional search method. More recent developments have introduced machine learning into PCG: Slover-based approaches [29] use machine-learned

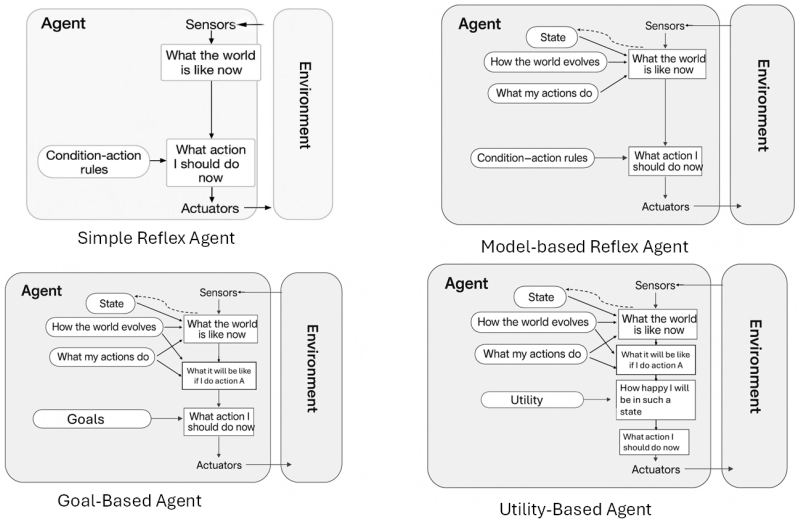


Fig. 2: **Agent Architecture Overview.** Each diagram illustrates how the agent processes percepts from the environment through sensors, transforms them via internal state or models, and selects an action that is returned to the environment through actuators.

models for content evaluation, while PCG via Machine Learning (PCGML) [30] generates content directly from the machine learning model. Building on these developments, modern Generative-AI PCG approaches [31], [32] apply Generative Adversarial Networks (GANs) [33], [34] and transformer-based generators [35], [36] in PCG, marking a shift from optimisation-driven generation toward fully generative content synthesis. Although existing PCG research has extensively explored the generation of most forms of game content, the generation of complete NPCs remains largely unaddressed. This project seeks to fill this gap by enabling end-to-end NPC synthesis directly from natural-language descriptions.

Multimodal generative models aim to generate a similar entity in different modalities given a unimodal source instance [37]. Generative approaches can be divided into three categories: grammar-based [38]–[44], encoder-decoder [45]–[48], and continuous generation models [49]–[54]. Within the encoder-decoder category, Large Language Models (LLMs) have recently made significant progress. It employs Transform-based architectures [55] that map text inputs to latent representations and decode them into coherent, structured output. Models such as Flamingo [56] and LLaVA [57], extend encoder-decoder frameworks to jointly process visual and textual inputs, enabling image-grounded text generation without task-specific supervision. Some concurrent work, including PaLI-X [58] -expand this capability to richer multimodal settings, achieving increasingly coherent cross-modal synthesis. More recent systems, such as Qwen3-VL [59], further advance this trend by unifying visual understanding, multimodal reasoning, and text-conditioned generation within a single architecture. These models offer a strong theoretical basis for translating natural language into visual or behavioural outputs. However, their objectives remain broad and domain-general, and they do not synthesise the coordinated combination of appearance, attributes, and behaviour required for a complete NPC. This project addresses this gap by specialising multimodal generation techniques for end-to-end NPC construction.

Intelligent Agent was first introduced in *Artificial Intelligence: A Modern Approach* [60]. It is defined as any entity that perceives its environment through sensors and acts upon that environment through actuators. Furthermore, it can be categorized as simple reflex agents [61], model-based reflex agents [62], goal-based agents [63], and utility-based agents [64]. The structure of different types of agents is shown in Figure 2 [65]. In common commercial games, NPCs are implemented as simple reflex agents whose behaviour is determined by reactive condition-action rules and a set of tunable parameters [5]. Traditional systems treat these parameters as static values, authored manually or generated procedurally [24]. Subsequent work explored optimisation-based approaches that search parameter spaces to produce agents of varying difficulty or behavioural profiles [26]. However, none of these approaches constructs the full multi-modal specification of an NPC. This project addresses this limitation by enabling end-to-end NPC generation—including appearance, attributes, and behaviour—directly from natural-language descriptions.

III. Description of the work

A. System Overview

a. Problem Statement

The proposed system is a real-time text-to-NPC generation pipeline integrated into a Vampire-Survivor-style game prototype. Its purpose is to allow players or developers to generate fully functional NPCs—complete with appearance, attributes, and behavioural logic—purely from natural-language descriptions during gameplay.

The core benefit of this solution is the automation of NPC creation, significantly reducing the manual labour and specialist skills required to design and integrate new game entities. The system extends the game's runtime capabilities by supporting dynamic content creation, enabling players to influence the game world in real time and providing developers with a flexible tool for rapid prototyping, testing, and content exploration.

b. Project Scope

In Scope

The project will deliver a real-time text-to-NPC generation system capable of:

- Accepting natural-language descriptions from a player or developer.
- Parsing semantic information (appearance, attributes, behaviours).
- Generating multimodal NPC components:
  - 2D visual assets
  - Character attributes
- Assembling all generated components into an engine-ready NPC prefab.
- Injecting the NPC directly into a running game session (Vampire Survivor–style prototype).

Out of Scope

- Training new generative models from scratch (only existing models such as Qwen3-VL will be integrated).
- Full-scale commercial game development. (Only a Vampire Survivor–style prototype will be used).

B. System Analysis

a. System Dependencies

- **LLM:** Required for text parsing, semantic extraction and sprite/mesh creation.
- **Game Engine:** Must support runtime prefab instantiation, script attachment and rendering.
- **Hardware Resources:** GPU/CPU capable of executing the LLM and game prototype at low latency.

b. Assumptions

- A stable LLM (e.g., Qwen3-VL) is available and can be queried with acceptable latency.
- The game engine (Unity) supports runtime instantiation of prefabs.
- Players provide understandable, natural-language descriptions.
- Generated NPC assets fit engine limits (resolution, size, animation format).
- The Vampire-Survivor-style prototype is assumed to be structurally stable and sufficiently modular.

c. Risks

- **High Latency:** Generating may exceed acceptable real-time constraints, affecting gameplay flow.
- **Model Inconsistency:** LLM or generative models may output invalid or unusable structures.
- **Engine Compatibility:** Generated textures, models, or JSON schemas may not be directly compatible with the target engine without preprocessing.
- **System Overload:** Real-time generation may cause CPU/GPU bottlenecks during gameplay.
- **Ambiguous Player Input:** Natural-language descriptions may be vague or contradictory, lowering NPC quality.
- **Third-party Dependency Risks:** The system depends on external models (LLMs, diffusion models), which may have rate limits or version changes.
- **Prototype Limitations:** Undocumented constraints or architectural limitations may restrict the kinds of NPCs that can be generated or require additional refactoring time to support the text-to-NPC pipeline.

d. Constraints

- **LLM Constraints:** LLMs are generative—not deterministic planners—so behaviour outputs require post-processing or fallback logic. This leads to some non-compliant outputs:
  - Malformed output can break prefab assembly unless validated or corrected.
  - Generated spritesheets may miss certain animation frames/ contain inconsistent art style / exceed Unity sprite import guidelines.
  - LLM may produce code-like pseudostructures that require sanitisation
  - LLM inference time increases with prompt length and image generation complexity.
- **Game Engine (Unity) Constraints:** Generated assets must conform to the Unity constraints.
  - A valid Prefab need to include: render, Animator and controller, collider, Behavior scripts.
  - A valid Sprite must consider these constraints: Texture Size, Compression formats, Pivot points, Frame slicing grid, sprite count per sheet.
- **Hardware:** The computational resources available during runtime impose strict bounds on multiple limitations, which manifest as:
  - **LLM scale:** Local hardware (CPU/GPU memory) limits the deployment of large multimodal models, forcing the system to use lighter LLM variants.
  - **Inference performance:** GPU/CPU throughput determines the maximum achievable generation speed. Image generation or complex behavioural synthesis may introduce unacceptable latency, limiting the system's real-time capabilities.
  - **Gameplay simulation:** The Vampire Survivors-style prototype relies on CPU-heavy swarm simulation. Hardware limitations cap the number of spawned dynamic NPCs and constrain behaviour complexity.
  - **Threading and concurrency:** Unity's main thread cannot be blocked by LLM calls or asset processing. Systems without dedicated multi-core capacity or without asynchronous execution support will degrade in frame rate and cause gameplay stutter.

C. System Architecture

a. System Context Diagram

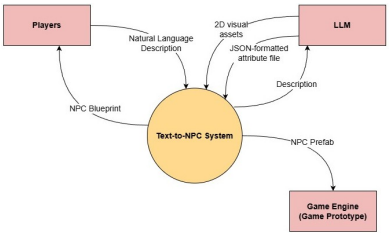


Fig. 3: **System Context Diagram.** The diagram shows the high-level architecture of the text-to-NPC system

b. Data Flow Diagram

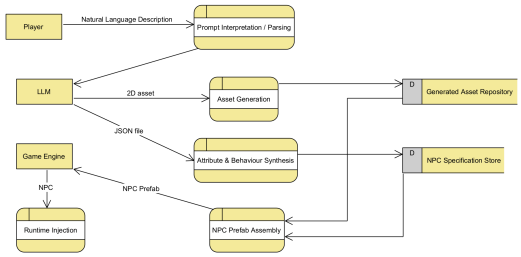


Fig. 4: **Data Flow Diagram.** The figure outlines the end-to-end flow of data as player input is parsed, transformed by the LLM, assembled into NPC assets and specifications, and injected into the game engine as a playable entity

D. Functional Behaviour

This subsection formalises how the system is expected to behave from the user's perspective. It introduces the main use case for text-to-NPC generation.

a. Use Case Diagram

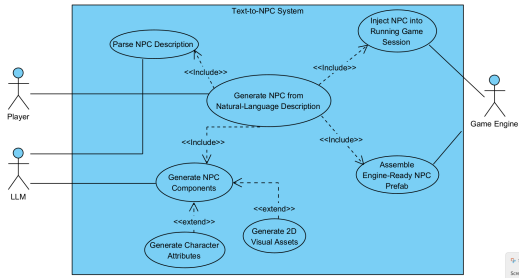


Fig. 5: **Use case diagram.** It illustrates how the main use case—generating an NPC from natural-language input—is supported by parsing, component generation, prefab assembly, and runtime injection.

b. Use Case Specification

UC-1	**Generate NPC From Natural-Language Description**
Primary Actor(s)	Player, Developer
Stakeholders & Interests	Game Engine (must receive valid prefab), Designer (requires rapid iteration)
Trigger	User enters an NPC description and clicks *Generate*
Pre-conditions	LLM service available; game engine running; stable prototype loaded
Post-conditions	A fully generated NPC prefab is produced and injected into the game
Main Success Scenario	1. User provides natural-language input in UI. 2. System sends description to LLM. 3. LLM returns JSON attributes & visual assets. 4. System synthesises behaviour logic. 5. System assembles NPC prefab. 6. Game engine spawns NPC in active scene.
Extensions	3a. LLM error → System displays error and requests re-prompt. 5a. Prefab assembly fails → fallback NPC created.
Priority	High
Special Requirements	Generation must complete within real-time game

TABLE I: **Use case specification for UC-1: Generate NPC from Natural-Language Description.** This table details the actors, workflow, conditions and alternative paths associated with the system's primary use case.

E. Requirements Analysis

a. Functional Requirements

ID	**UI Element / Function**	**Type / Control**	**Specification Description**	**Data Type / Constraints**	**Business Rules / Validation**
FR-1	Monster Blueprint Selector	Dropdown / Object Picker	System shall allow the user to select a base monster blueprint as a structural template for generated NPC attributes and behaviours.	Unity Object Reference (ScriptableObject / Prefab)	Must reference a valid monster blueprint; cannot be empty.
FR-2	Language Input Box	Multiline Text Field	System shall accept a natural-language description and forward it to the LLM for interpretation.	UTF-8 Text, up to 500 chars	Must contain at least one describable trait (stat or appearance).
FR-3	Generate Trigger	Button (disabled state)	System shall trigger the LLM generation process when the user initiates it.	Boolean Action	Button disabled during active LLM generation.
FR-4	LLM Status Indicator	Progress Label / Loading Bar	System shall display the progress of the LLM request (e.g., percentage or "Running...").	String / Float	Must update in real time during generation.
FR-5	LLM Temperature	Slider (0–1)	The system shall allow the user to adjust the sampling temperature for generation variability.	Float (0.0 – 1.0)	Must remain within bounds; reset on scene load.
FR-6	JSON Attribute Output (Internal)	Not directly visible (Inspector updates automatically)	System shall parse LLM JSON output and bind resulting values to NPC fields (HP, ATK, movement stats).	Structured JSON	Missing or malformed fields must trigger error fallback.
FR-7	Visual Asset Output (Internal)	Sprite Preview in Inspector	System shall import LLM-generated sprite assets and assign them to the NPC blueprint's animation slots.	PNG spritesheet (≤ engine limits)	Resolution must meet Unity import constraints; format must be loadable.
FR-8	NPC Prefab Assembly	Backend Process	System shall assemble all generated attributes and sprites into a runtime-instantiable NPC prefab.	Unity Prefab	Prefab must pass all required component checks (collider, animator, scripts).
FR-9	NPC Injection into Game	Backend → Game Engine	System shall instantiate the generated NPC into the active Vampire Survivors prototype scene.	GameObject Instance	Must not break existing gameplay loop; spawn position determined by spawner rules.
FR-10	Error Handling	System Behaviour	On LLM error (timeout, invalid JSON, missing sprite), system shall display error and revert to safe state.	N/A	Error must be logged; UI must not freeze.

TABLE II: **Functional requirements derived from UC-1.** This table specifies the user-interface behaviours and backend processes required to support the end-to-end generation, assembly and injection of NPCs from natural-language descriptions.

b. Non-Functional Requirements

Requirement ID	Categories	Description
NFR-P1	Performance	NPC generation (LLM request → prefab assembly) shall be completed within a reasonable time under standard conditions.
NFR-R1	Reliability	System shall gracefully handle LLM timeouts or failed requests by providing fallback NPCs.
NFR-R2	Reliability	JSON attribute validation must prevent malformed data from breaking prefab assembly.
NFR-R3	Reliability	Sprite import failures shall fall back to placeholder textures.
NFR-R4	Reliability	NPC instantiation shall never halt or crash the gameplay loop.
NFR-R5	Reliability	System failure shall not corrupt existing save files or player state.
NFR-S1	Scalability	Architecture must support swapping in other LLMs.
NFR-U1	Usability	UI must allow NPC generation with no more than one user action (enter text → generate).
NFR-U2	Usability	UI error messages must be human-readable.
NFR-U3	Usability	All controls must remain accessible during gameplay testing mode.
NFR-M1	Maintainability	Prefab assembly logic must be fully decoupled from game-specific logic.
NFR-SEC1	Security	User input must be sanitised to avoid injection attacks in logs or file generation.
NFR-SEC2	Security	No generated assets may access or overwrite existing core game files.
NFR-C1	Compatibility	Generated assets must adhere to Unity Sprite import requirements.
NFR-C2	Compatibility	JSON output must follow the NPC specification schema.
NFR-T1	Testability	Error recovery must be testable by simulating invalid JSON and missing assets.

TABLE III: **Summary of non-functional requirements.** The table outlines the key constraints governing system behaviour, including performance targets, reliability safeguards, usability expectations, maintainability rules, security constraints and compatibility requirements.

IV. Method

We introduce a text-to-NPC generation pipeline designed to translate natural-language descriptions into engine-ready 2D non-player characters.

A. Problem Definition and Notation

The input to our system is a natural-language description $d \in \mathcal{D}$ provided by a player or developer, optionally conditioned on a base monster blueprint $b \in \mathcal{B}$ from the game prototype. The goal is to construct a function that maps this textual description to a complete, engine-ready NPC specification:

$$f_\theta(d, b) = (A, V, B). \tag{1}$$

Here, the output (A, V, B) represents the core components required for instantiation of a 2D enemy, which is a common representation in NPCs [5], [27]:

- A denotes the numerical attribute set, including scalar parameters. Formally, A is a mapping $A : \mathcal{K}_A \to \mathbb{R}$ from attribute keys to real-valued quantities.
- V denotes the visual representation, implemented as a spritesheet $V \in \mathbb{R}^{H_s \times W_s \times 4}$ together with an animation partition specifying idle, movement, and attack frame sequences.
- B denotes the behaviour parameter vector, $B \in \mathbb{R}^m$, which conditions the enemy controller script.

We next outline how these quantities are defined. Rather than emitting (A, V, B) directly, the system first generates an intermediate textual specification s, produced by a multimodal LLM through constrained prompting. This intermediate processing approach is widely used in PCG area [66], [67]. We formalise this as

$$g_\theta(d, b) = s. \tag{2}$$

where s is a JSON document [68], [69] that must conform to a predefined schema $\mathcal{S}$ describing attribute fields, appearance directives, and behaviour tags . A validator ensures $s \in \mathrm{JSON}_{\mathcal{S}}$; malformed outputs are corrected, regenerated or replaced by safe defaults.

Given a valid specification s, a second module converts it into a concrete game engine artifact [27], [30]:

$$h_\theta(s) = (A, V, B). \tag{3}$$

The function h_θ includes sprite generation, animation slicing, attribute binding and behaviour parameter extraction, producing engine-level data structures compatible with the Vampire Survivors prototype.

Finally, Unity provides a spawning operator [70]

$$\mathrm{Spawn}(A, V, B) = \tilde{n}. \tag{4}$$

which instantiates the NPC $\tilde{n}$ as a `GameObject` with renderer, collider, Animator and AI controller components initialised according to (A, V, B).

Over-complete specification [71]–[73]. Not all components of (A, V, B) are independent. For instance, collider radius can be derived from sprite dimensions, and behaviour aggressiveness may correlate with movement speed. We nevertheless task the LLM with generating all components explicitly. This redundancy increases controllability, simplifies validation and enables cross-checking for semantic inconsistencies, improving stability during runtime assembly [24].

B. Feature Backbone

The system processes natural-language descriptions by decomposing them into three latent semantic subspaces—appearance, attributes and behaviour—mirroring the factorised representation used in agent-based modelling [5]. Instead of treating text as a monolithic input, the backbone explicitly extracts structured features that correspond to the components of the final NPC triplet (A, V, B).

Textual Feature Encoding: The multimodal LLM encodes the input description in a sequence of contextual embeddings $\{e_i\}_{i=1}^{T}$ capturing high-level semantics [55], [74]. In contrast to traditional captioning models, the encoding is schema-aware [75]–[77]: Prompt tokens are also strengthened by a series of field markers that embed force on data into already designated semantic channels. This allows the backbone to keep consistent mappings in place between language and engine-level representation [78].

Latent Factorisation into NPC subspaces: Following disentangled representation principles [79], the feature encoded output is projected to three specialised latent vectors:

- z_A for scalar attributes
- z_V for visuals
- z_B for behavioural cues

This factorisation allows downstream modules to operate independently while preserving semantic consistency across modalities.

Visual Conditioning Backbone: For appearance generation, the backbone extracts appearance cues from z_V and injects them into the image-generation head as conditional embeddings, similar to text-conditioned diffusion and vision–language models [80]–[82]. This ensures that generated spritesheets maintain coherence with textual descriptions while remaining compatible with the game's animation pipeline.

Behavioural Embedding Backbone: The behaviour vector z_B is mapped into a normalised parameter vector used to condition the game's enemy controller. This mirrors parameter-driven agent conditioning in prior work [83], but extends it by deriving parameters directly from natural language.

C. Technology

The objective of this project is to assess whether natural-language descriptions can drive the real-time creation of complete, engine-ready NPCs within an active game loop. To support this investigation, we adopt three key technological components: a multimodal large language model (Qwen3-VL), a real-time game engine (Unity), and a validated, high-load game prototype (Vampire Survivors–style) as the test environment. Each component is selected to meet specific theoretical and engineering requirements, as detailed below.

a. Multimodal Large Language Model: Qwen3-VL

To translate natural-language descriptions into the triplet of NPC components—attributes, appearance, and behaviour—we employ Qwen3-VL [59] as the system's semantic backbone for three reasons.

- **Strong multimodal reasoning capabilities:** Qwen3-VL is trained on paired image-text datasets and is compatible with vision-conditioned text generation. This allows for descriptors interpretation, which is required for generating game-ready sprites and behaviour parameters [59], [84].
- **Reliable structured output for schema-based generation:** Qwen-series models [59] highlight tool-use

alignment and structured responses [68], which makes them great for churning out consistent (A, V, B) outputs.
- **Lightweight In contrast to proprietary models:** Qwen3-VL [59] supports more lightweight local deployment. This is necessary for controlled, reproducible latency measurements [85], less reliance on external APIs [86], lower hardware requirements [87], and experimental assessment under real-time constraints [5].

b. Game Engine: Unity

Unity [88] serves as the runtime environment for NPC instantiation, rendering, animation, and physics interaction. Its component-based architecture aligns well with the needs of dynamically generated game entities. It has three benefits:

- **Runtime prefab instantiation:** Unity supports dynamic creation of GameObjects [89], allowing the system to assemble NPCs during gameplay by attaching components.
- **Mature 2D asset and animation pipeline [90]:** Unity's sprite slicing, atlas importing, animation timelines, and State Machine–based Animator system make it highly suitable for handling LLM-generated sprite-sheets.
- **Strong debugging and extensibility support:** Unity's editor enables real-time visual debugging of generated NPCs, which is crucial for iterative evaluation, failure analysis, and user testing [91].

D. Algorithmic Components

This subsection describes only the algorithmic components implemented in the current system, focusing on attribute-level tuning, schema-constrained prompting, and safe updates to Unity assets.

a. Schema-Constrained Prompt Generation

The current system supports attribute-level tuning by constructing a structured prompt that limits the LLM to modifying only fields present in the selected `MonsterBlueprint`. The prompt generator:

1) enumerates editable fields from the blueprint,
2) embeds their current numeric values into the prompt,
3) attaches a minimal JSON schema specifying key names and numeric types,
4) enforces a "JSON-only" output rule to simplify parsing.

This follows lightweight schema-constrained generation used in controlled LLM outputs [68], but adapted to the restricted attribute-tuning scope of the prototype.

b. JSON Parsing and Safe Field Binding

The response from the LLM is parsed into a `MonsterTuning` object using standard JSON deserialization. The binding logic:

1) checks whether each returned field is present and correctly typed,
2) updates the corresponding blueprint field only when a valid value is supplied,
3) leaves all unspecified properties unchanged to preserve internal blueprint integrity.

This cautious update strategy is consistent with safety practices in PCG systems, where incomplete or partially valid generations must not corrupt existing assets [24].

c. Prefab Update via Unity's Asset Pipeline

The underlying *MonsterBlueprint* asset is updated with the current prototype. The algorithm:

1) writes modified fields back into the *ScriptableObject*
2) marks the asset as *EditorUtility.SetDirty*
3) commits changes using *AssetDatabase.SaveAssets*

All future enemy spawns in the game automatically reflect the tuned attributes as the current spawn logic already consumes the modified blueprint. This also provides a clear separation between generative tuning (editor-side) and gameplay instantiation (runtime), reducing coupling and avoiding fragile dependencies.

d. Minimal Error-Handling Strategy

The prototype implements a basic but sufficient fallback layer:

1) invalid JSON $\rightarrow$ logged; no updates applied
2) missing fields $\rightarrow$ ignored without breaking the pipeline
3) LLM request errors $\rightarrow$ surfaced to the UI via status messages

Less challenging than full PCG error-recovery pipelines, this ensures editor stability and prevents malformed generations from corrupting assets.

e. Latency-Aware Asynchronous Execution

All LLM interactions are executed via async/await, ensuring that Unity's main editor thread remains responsive:

1) LLM calls run asynchronously,
2) UI remains interactive during generation,
3) no blocking operations occur on the editor loop.

This aligns with real-time responsiveness guidelines for interactive creative tools [20] and provides early evidence that text-driven NPC tuning can be integrated into game-development workflows without disrupting user experience

E. System Workflow

At runtime, the system implements the mapping $f_\theta(d, b) = (A, V, B)$ through a sequence of stages that transform a natural-language description into an engine-ready NPC and inject it into the running game. Figure 6 provides an overview of this pipeline.

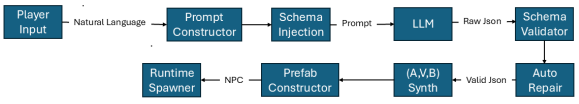


Fig. 6: **Pipeline diagram.** It illustrates six stages that transform a natural-language description into an engine-ready NPC and inject it into the running game.

1) **Stage 1: Player Input**
   At runtime, the user interacts with the tuner window, providing a selected MonsterBlueprint and a natural-language instruction string. The process begins when the user presses Tuning, which triggers an asynchronous LLM request.

2) **Stage 2: Prompt Constructor**
The system builds a structured prompt from: raw description d, blueprint b, and a list of editable fields. This prompt follows a constrained JSON-output template, instructing the LLM to:
   - modify only specific numerical fields
   - output JSON only
   - omit explanations
   - use blueprint attributes as initial context

   This corresponds to the semantic function [92]:
$$\tilde{d} = \text{PromptTemplate}(d, b, S) \quad (5)$$
3) **Stage 3: LLM-Based JSON Generation**
The LLM receives the structured prompt and returns a JSON object. This is the realised version of the theoretical equation 2: $s_{raw} = g_\theta(d, b)$.
4) **Stage 4: JSON Validation and Parsing**
This stage performs:
   - **Syntax validation:** must be valid JSON
   - **Schema validation:** keys must be among permitted fields
   - **Type checking:** numeric values must parse correctly
   - **Graceful fallback:** errors are logged; no game crash
$$s = \text{ValidateAndRepair}(s_{raw}) \quad (6)$$
5) **Stage 5: Attribute Binding into Game Engine**
Once the parsed object is obtained, the system updates the blueprint fields, directly instantiates the abstract mapping of Equation 3: $(A, V, B) = h_\theta(s)$.
6) **Stage 6: Update Engine Assets**
Blueprint changes are committed to disk using Unity's asset pipeline. This ensures deterministic reproducibility and persistent tuning across sessions, maps directly to Equation 4: $\text{Spawn}(A, V, B) = \tilde{n}$.

V. Design

A. Gameplay Design

The prototype employed in this project is based on the gameplay structure of Vampire Survivors–style arena shooters, a genre characterised by high-density enemy swarms, continuous progression, and minimalist player control. In this gameplay loop, you have to navigate a 2D arena as enemies spawn at increasing rates, forcing constant movement, kiting, and frequent combat. As shown in Figure 1.

Enemies in this genre usually show extremely simple reflex-based AI—approaching the player directly and attacking when within range—instead of goal-oriented behaviour or deliberative behavior.

An experience in battle in this prototype consists of:

- **Continuous enemy spawning:** Enemies are generated at designated spawner time intervals using predefined spawner rules, establishing the ratcheting up the pressure characteristic of bullet-heaven games.
- **Direct pursuit behaviour:** Each enemy actively follows the player with a straightforward chase algorithm, aligned with the reflex-agent model described by Brooks [61].
- **Parametrised combat interactions:** Enemy performance—movement speed, attack cooldown, health, and damage—is controlled by scalar parameters, consistent with parameterised agent concepts in game designn [5].
- **High-density encounters:** The game aims to sustain dozens to hundreds of simultaneous NPCs. High-load crowd environments are considered valuable for stress-testing AI systems and evaluating computational robustness [93].

B. NPC Design

a. NPC Attributes

The current prototype implements enemies as parameterised 2D agents driven by a shared Monster controller. Each runtime enemy is instantiated from a MonsterBlueprint ScriptableObject's sub-object, which encapsulates the numerical "stats" visible in the Unity inspector (Table IV):

Attribute	Field Type	Used In	Effect in Gameplay
Hp	float	currentHealth	Determines how many hits the monster can take.
Atk	float	damage calculation	Higher values increase damage dealt to player.
Movespeed	float	Rigidbody2D drag + movement vector	Controls chase speed and evasiveness.
Acceleration	float	rb.drag computation	Affects responsiveness & knockback recovery.
AtkSpeed	float	attack cooldown logic	Determines attack frequency.
Armor	float	damage reduction	Influences effective durability.

TABLE IV: Enemy Attribute Definitions

These fields form the concrete instance of the abstract attribute set A defined in the Method section.

b. NPC Initialization

At runtime, the Monster component is responsible for binding these design-time parameters to engine behaviour. During Setup, the controller:

1) Initialises *currentHealth* from *monsterBlueprint.hp* (plus any buff);
2) Configures a *Rigidbody2D* and two colliders (a circular "legs" collider and a box hitbox) to control physical size and interaction;
3) Initialises the *SpriteAnimator* with the walk sequence from the blueprint;
4) Computes drag as $acceleration/(speed^2)$, linking the abstract acceleration parameter to Unity's physics model.

c. Damage, Death and Reward System

Damage handling and death behaviour are also centralised in this controller. The *TakeDamage* method subtracts from *currentHealth*, triggers a brief hit–flash shader, applies knockback proportional to $\sqrt{drag}$, and, when health reaches zero, runs the Killed coroutine.

This coroutine removes the monster from the *EntityManager*'s *LivingMonsters* list, plays death particles, optionally drops loot via the blueprint's gem/coin loot tables, and finally returns the instance to the pool.

This design keeps the kill/loot loop entirely data-driven: changing the blueprint immediately changes how valuable and how durable an enemy feels in-game.

C. LLM Interaction Design

The design of the prompting strategy is central to the reliability and controllability of the text-to-NPC pipeline. Given that large language models can produce unconstrained or ambiguous text when not appropriately guided, the system adopts a schema-aligned, instruction-driven prompting design.

The prompt is constructed around three core principles:

a. Schema-Constrained Output

The model must produce JSON that conforms to a lightweight NPC schema, containing only editable attributes.

To enforce this structure, the prompt includes:

- a complete list of allowed keys
- their expected data types
- numeric ranges and safety constraints
- An explicit instruction to output JSON only

This converts the LLM from a free-form generator into a constrained semantic function (aligned with Equation 2).

b. Blueprint-Grounded Contextualisation

The prompt contains the present values of the selected *MonsterBlueprint* for the instruction template. Basing the generation procedure on the game's internal state reflects conditioning methods employed in grounded LLM systems, where the model is guided by structured world-state information [67].

This grounding serves two design objectives:

1) It restricts the model's search space to modifications rather than unconstrained invention.
2) It increases semantic alignment between user intent and engine-level attributes by providing the LLM with concrete numerical context.

c. Natural-Language Instruction Layer

Output is well-structured, but input remains purely natural language. The prompt preserves the authorial flexibility of free-form text while channeling the model to make appropriate parameter adjustments.

D. Data Specification Design

a. NPC Data Model Overview

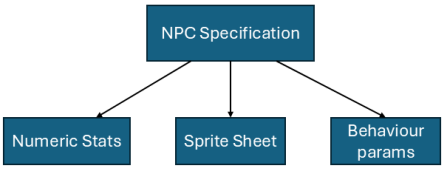


Fig. 7: **NPC data diagram.** It shows how the generated specification branches into numerical attributes, visual assets, and behaviour parameters used by the engine.

b. JSON Specification (LLM Output Schema)

The LLM outputs a structured JSON document $s \in \text{JSON}_{\mathcal{S}}$ conforming to a predefined schema $\mathcal{S}$. Table V summarises the fields, types and constraints.

Field	Type	Constraints
hp	number	$hp > 0$
atk	number	$atk \geq 0$
movespeed	number	$0.01 \leq movespeed \leq 5.0$
acceleration	number	$1 \leq acc \leq 20$
atkspeed	number	$atkspeed > 0$
armor	number	$armor \geq 0$
behaviour_style	string	$\in$ {boomerang, boss, melee, ranged, throwing}

TABLE V: JSON Schema Specification for LLM Output

c. Engine-side Data Structures

Unity maintains a canonical representation of NPC attributes via the `MonsterBlueprint` ScriptableObject. Table VI shows how each JSON field is bound to an engine-side property.

Blueprint Field	Type	Source / Mapping
hp	float	JSON.hp → monsterBlueprint.hp
atk	float	JSON.atk → monsterBlueprint.atk
movespeed	float	JSON.movespeed → monsterBlueprint.movespeed
acceleration	float	JSON.acceleration → monsterBlueprint.acceleration
atkspeed	float	JSON.atkspeed → monsterBlueprint.atkspeed
armor	float	JSON.armor → monsterBlueprint.armor
walkSpriteSequence	Sprite[]	Engine-generated animation frames
gemLootTable	LootTable	Engine-defined loot behaviour

TABLE VI: MonsterBlueprint: Engine-side Attribute Container

At runtime, these blueprint fields are consumed by the `Monster` class, as shown in Table VII.

Runtime Field	Type	Description
currentHealth	float	Initialised from Blueprint.hp
rb.drag	float	Computed from acceleration / movespeed^2
monsterSpriteAnimator	Animator	Uses visual component V
OnKilled	UnityEvent	Triggered when HP ≤ 0

TABLE VII: Runtime Monster Data Mapping

E. UI Design

a. Tuning Window UI

The Monster AI Tuner is a custom Unity Editor window providing an interface for the text-to-NPC generation pipeline. It implements the following components:

- **Blueprint Selector:** The top-level field allows the user to assign a MonsterBlueprint ScriptableObject. This connects the UI to engine-side data structures and defines which NPC instance will be modified through LLM-based tuning.
- **Natural-Language Input Area:** Multi-line text box captures user instructions. This UI choice reflects findings from natural-language PCG tools, where free-form text enables expressive control without requiring programming knowledge.
- **Tuning Action Button:** Pressing Tuning triggers the asynchronous LLM pipeline. A loading state prevents duplicate requests and keeps editor interaction safe and responsive.

- **Temperature Slider:** A simple numerical slider controls LLM randomness. Lower values produce stable, predictable tuning; higher values create more exploratory or exaggerated stats. This exposes a minimal but meaningful parameter to support experimentation.

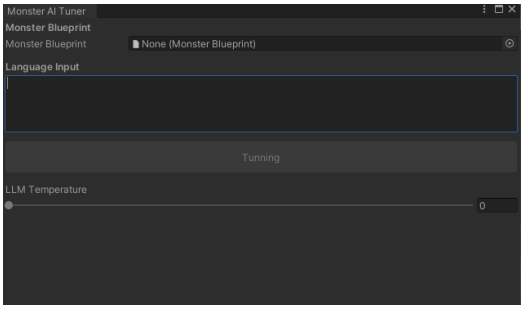


Fig. 8: **Tuning Window**. It demonstrates the UI layout and conveys how the LLM connects to Unity.

b. Main Scene UI

The main scene presents a simplified Vampire Survivors–style menu, serving as the player's entry point into the experiment environment. Its design focuses on functional clarity:

- **Title Banner:** A large centred banner communicates the experimental clone context.
- **Mode Buttons (Start, Shop, Upgrades):** Three primary actions allow entering the game, previewing upgrades, or navigating to secondary systems. They mirror standard layout conventions in VS-like games—large icon-based buttons arranged horizontally.

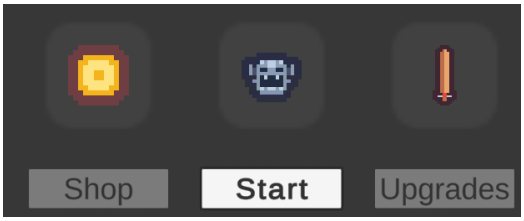


Fig. 9: **Mode Buttons Interface.** It shows the core navigation buttons used to enter gameplay and auxiliary systems.

c. Character Selection UI

The character selection UI allows players to choose their starting avatar before entering gameplay. This screen implements typical roguelite conventions:

- **Character Cards**: Each character is displayed with:
  - portait sprite
  - core stats
  - starting weapons/abilities
  - purchase/selection button

  This UI provides the anchor for evaluating how player stats interact with AI-generated NPC behaviour during runtime.
- **Unlock/Buy Logic:** Characters may be locked behind currency requirements.

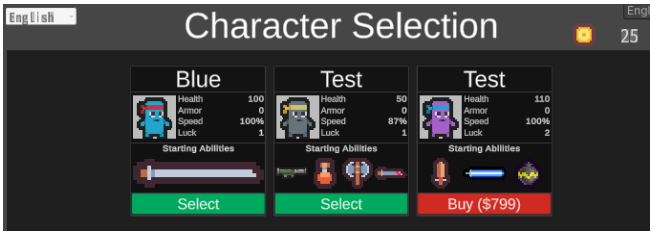


Fig. 10: **Character Selection UI.**

F. Design Rationale

Design Element	Section	Rationale
Vampire Survivors–style gameplay	Section 5.1	High-density enemy environments provide natural stress tests for real-time NPC generation; reflex-based AI makes attribute-driven tuning meaningful [5], [61].
Parameterised NPC attributes	Section 5.2.1	Compatible with the theoretical triplet (A, V, B); leaving room for additional enemy types and more complex behaviours; enables interpretable, scalar-level tuning [94]; avoids requiring structural behavioural changes; safely editable via JSON. [5]
Schema-aligned, instruction- driven prompting design	Section 5.3	This design reduces ambiguity, improves structural consistency, and aligns with best practices in LLM-based tool invocation and controllable generation [78], [95].
Schema-constrained Output	Section 5.3.1	Significantly increase the rate of valid, machine-interpretable outputs [68]; ensuring that the system reliably produces data that can be parsed and applied to Unity assets
Blueprint-Grounded Contextualisation	Section 5.3.2	Provides a stable, serialised data container; clean mapping from JSON → Unity runtime; avoids dynamic code generation [90].
Purely Natural-language Input	Section 5.3.3	Enabling non-expert designers or players to tune NPCs without interacting with internal data structures; reflects the "mixed-initiative" philosophy widely employed in procedurally generated content systems [24], in which human creativity is combined with machine-level formalisation.
JSON as LLM output format	Section 5.4.1	LLMs reliably produce JSON; easy to validate; direct mapping to engine fields reduces parsing ambiguity [24], [68].
Simplified UI (Tuning Window + minimal runtime UI)	Section 5.5	Removes visual noise; focuses user evaluation on AI-generated NPC behaviour; supports consistent testing. [24]

TABLE VIII: **Design rationale for Game Design.**

Collectively, these design decisions ensure that the proposed system can

1) interpret natural-language descriptions reliably
2) translate them into safe and meaningful NPC parameters
3) integrate these parameters into a high-density game environment without breaking gameplay stability

This alignment between design intent and system behaviour directly supports the project's research objective of evaluating the feasibility of real-time text-to-NPC generation in a commercially relevant gameplay setting.

VI. Implementation & Progress

There is a follow-through in project management and milestones set forth in the original project proposal, with some changes to the plan as needed due to implementation difficulties and lessons learned over the long haul of development. In this chapter we present the planned work structure, progress to date, variances from the plan, and how time and resources have been managed.

A. Implementation (Key Contribution)

1. Foundational literature review

2. LLM Deployment & Integration

The system is integrated into a locally deployed instance of Qwen3-VL via a custom Unity C wrapper. The model

is hosted via Ollama and exposed at a local endpoint (http://localhost:11434/api/generate). Unity communicates with the model via an asynchronous HTTP pipeline implemented in LLM.cs.

The integration layer does four main things:

1) **Constructing request payloads:** The system constructs an *OllamaGenerateRequest* which contains the model name (*qwen3-vl:8b*), prompt string, temperature, and generation options. This structure is serialised into JSON and sent in *UnityWebRequest*.
2) **Executing asynchronous inference:** All requests are executed with async/await, ensuring that LLM calls do not block the Unity editor thread, which is critical for maintaining interactive responsiveness.
3) **Parsing responses:** The returned JSON is deserialised into an *OllamaGenerateResponse*, from which the textual field response is extracted as the model's output. The wrapper also extracts code blocks when present, enabling structured responses.
4) **Graceful failure handling:** Network failures, timeouts, and parsing errors are logged and surfaced back to the UI without interrupting the editor environment.

3. Text-to-NPC prototype pipeline (attribute tuning)

The current prototype implements the first functional subset of the full text-to-NPC pipeline: attribute-level tuning driven by natural-language instructions. This module connects the LLM, JSON parsing, and Unity's *ScriptableObject* system into an operational end-to-end loop.

It has following functionalities:

1) **Prompt construction & LLM invocation:** When the user presses Tuning, the system constructs a schema-guided prompt from the current *MonsterBlueprint* values and the user's free-form language input. The prompt is assembled in *BuildPrompt()* (line-level implementation in *MonsterAITunerWindow.cs*), and sent asynchronously to the locally deployed Qwen3-VL model.
2) **JSON deserialization and validation:** The LLM is required to output a minimal JSON object containing only editable fields. The system parses it using *JsonConvert.DeserializeObject<MonsterTuning>()* (MonsterTuning.cs).
   If the output is missing fields, has incorrect types, or cannot be deserialized, the tool:
   - logs the error without crashing the editor
   - performs no unsafe writes to engine assets
   - preserves the integrity of the orginal *MonsterBlueprint*

   This forms the prototype's basic validation and fallback mechanism.
3) **Safe binding into engine data structures:** Once validated, the JSON object is mapped onto the Unity *MonsterBlueprint*. The fields explicitly returned by the LLM are updated and commited through Unity's pipeline. This selective overwrite scheme prevents accidental deletion or corruption of designer-defined data.
4) **A tuning window for user's input** (Figure 8).

4. Real-Time NPC Generation

The runtime NPC generation pipeline is built around a simple but scalable Unity asset structure, consisting of three primary directories:

- **Blueprint:** *ScriptableObjects* storing tunable numerical stats.
- **Prefab:** Enemy prefab templates referencing colliders, animation controllers, and renderer components.
- **Sprite:** sheets and animation frames used by the Monster controller during initialisation.

This design avoids costly prefab manipulation, simplifies the integration of LLM-based tuning, and keeps the runtime pipeline stable while supporting future extensions such as visual (sprite) or behavioural (AI scripts) generation.

B. Project Management

a. Planned Work Structure

The Gantt chart (figure 11) below shows the Planned project schedule. Different colours are used to represent different stages of work:

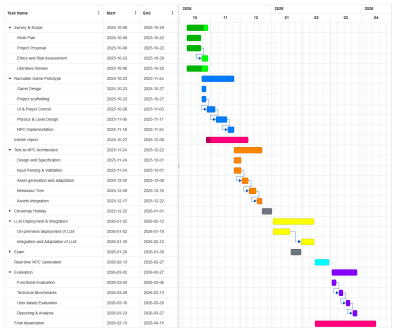


Fig. 11: Gantt chart: Work Plan

b. Deviations and Adjustments

Several deliberate adaptations were made during implementation compared to the original proposal for the project.

- **Scope Refinement: Attribute-Level Generation First:** Initially, full-text-to-NPC was the focus of the proposal; the initial target was in fact full-text-to-NPC generation—including attributes (A), visuals (V), and behaviours (B). However, implementing early sprite generation with behaviour synthesis added too much uncertainty, particularly about the model, Unity asset stability and determinism. Therefore a well thought out and robust implementation was focused on attribute-tuning pipeline.
- **LLM Deployment Timeline Adjustment:** The proposed local model implementation was scheduled at a later time in the project. However, the task progressed rather well, and the local deployment with the integration of the language model was observed to be less difficult than expected which made it possible for me to do this ahead of time.
- **Cancellation of Development for Runnable Game Prototype:** To speed up the process and to make the project goal text-to-NPC generation and not to create a full game, the prototype is based on an open-source Vampire Survivors–style implementation available on GitHub (Matthias Broske, VampireSurvivorsClone, MIT-licensed).

c. Time and Resource Management

It follows a well structured, resource aware project workflow designed to balance research, implementation, and evaluation within the constraints of an academic timeline. Time for assignment was directed by the Gantt plan (Figure 11) and refined iteratively based on implementation feedback.

Efficient Allocation of Development Time

Time was spread over three major work streams:

- Infrastructure setup (LLM, Unity integration).
  - Early progress on LLM deployment permitted later steps to work on pipeline correctness rather than tooling. Async execution and modular code design minimised debugging overhead.
- Text-to-NPC pipeline implementation.
  - Focus was set on settling the attribute-level pipeline before expanding the scope of development, blocking parallel development of unstable subsystems, stabilizing subsystems and keeping the system in the pipeline progressing iteratively and maintaining stable subsystems.
- Prototype preparation and testing.
  - By employing a pre-existing open-source Vampire Survivors clone, substantial development time was saved, enabling faster iteration on core research goals.

This modular allocation helped prevent bottlenecks and allowed work packages to progress independently when possible.

Resource Management (Compute, Tools, Assets)

The project relies on lightweight local hardware (single-machine deployment). Several design choices optimised limited computational resources:

- Local inference via Qwen3-VL-8B avoided cloud costs but also made for repeatable testing with controlled latency.
- Ollama offered a minimal, resource-efficient runtime suitable for iterative debugging.
- Unity's *ScriptableObject* ecosystem enabled safe, incremental updates without expensive runtime prefab generation.
- A significant amount of asset reuse from the open-source game lowered art and animation requirements.

Such decisions guaranteed that development and experimentation was possible on my personal hardware in a way that doesn't necessitate expensive GPUs.

Risk-Controlled Scheduling

Work was purposefully phased to avoid cascading failures:

- High-uncertainty tasks (sprite generation, behavioural synthesis) were postponed.
- Schema and validation layers were completed before any generative component was trusted.
- LLM integration was isolated into a dedicated module to limit system-wide refactoring risks.
- This staged design and strategy minimized rework and preserved the project on a predictable timeline.

Progress Monitoring and Adjustment

Self-evaluation (weekly progress reviews) were performed to:

- track slippage relative to the Gantt chart
- redistribute work depending on blockers
- maintain alignment with research deliverables

Where delays emerged (e.g., tuning prompt stability, JSON schema refinement), tasks were broken down into smaller milestones in order to preserve forward momentum.

d. Future Work

Future Task	Description and Planned Actions
Spritesheet Generation	Extend the existing attribute-only pipeline to create complete NPC visual assets. Next steps include introducing a diffusion-based image generator, validating sprite layout consistency, and creating a slicing/animation import pipeline.
Behavioural Generation	Language-driven behaviour styles require expanding JSON schema, mapping behaviour tags to controller presets, and adding validation to prevent unsafe or contradictory behaviours.
Evaluation Framework	Perform systematic evaluation of latency, correctness, usability, and gameplay stability. User testing will test expert vs. non-expert workflows for accessibility and mixed-initiative design effectiveness.
Local LLM Optimisation	Explore quantisation, prompt caching, and batching strategies to improve inference speed and stability. Aims to meet real-time generation.
Game Prototype Refactor	Refactor the prototype for modularity: separating NPC controllers, spawn systems, and visual pipelines to support A, V, B generation cleanly. This reduces coupling and prevents instability during LLM-driven asset injection.
Validation Extensions	Add robust validation checks for generated spritesheets, behaviours, and animation clips to ensure they satisfy Unity's structural requirements before runtime.
Content-Safety Layer	Integrate content-filtering prompts and rule-based sanitisation to prevent inappropriate or harmful NPC descriptions, aligning with ethical development standards.
Licensing & Compliance Review	Document all assets and model usage clearly. Ensure that reused code and generated assets comply with MIT licensing and Qwen3-VL permissions.
User Accessibility Evaluation	Plan and conduct comparative studies on how experts vs. non-experts interact with the tool, assessing opportunities for UI simplification and enhanced guidance.

TABLE IX: **Future Work Plan.** Tasks required to extend the prototype towards full (A,V,B) text-to-NPC generation and ensure legal, ethical, professional compliance.

C. Reflections

a. Reflection on Progress and Planning

The project moved forward more or less as planned, however important lessons formed some of the existing adjustments:

- Attribute tuning was the right minimal viable subsystem: Trying to implement full (A, V, B) generation from the start would have produced instability and slowed the ability of comparing partial results.
- Deploying local model proved easier than anticipated: In contrast to initial thought, as the research indicates from a risk evaluation, Qwen3 VL integration went smoothly. This freed time to refine, prompt, schema design and Unity binding logic.
- Game development was not valued enough—reuse was critical: The entire arena shooter needed to get built

from scratch. Building from scratch would have spent too much time.
- Prompt stability and JSON correctness needed more iterations: The LLM behaviour was less predictable than expected. Stabilising output required multiple rounds of schema refinements and validation checks, indicating the need for defensive programming in LLM-driven systems.

b. Personal Reflection:

This project has brought the tension to the fore between research ambition and practical engineering restrictions. Several lessons emerged:

- Real-time systems need strict robustness; even smaller JSON failures can break the game loop.
- Prompt design is an iterative, empirical process—a one-shot solution.
- New generative AI built into existing engines demands careful interface design and defensive validation.
- Timeboxing involved complex components (e.g., sprite generation) to avoid scope creep.
- AI techniques are emerging that can significantly augment game design workflows, but only when appropriately combined with fine-tuning constraints.

c. LSEPI Considerations

- **Legal — Open-source licensing and model usage:** The original game prototype is MIT-licensed; attribution is provided and derivative use complies fully. Qwen3-VL is released under a permissive license that permits local research use. License compliance is essential before any public dissemination of the system.
- **Social — Accessible tools for non-experts:** The system is structured to empower players and designers to create NPCs using natural language. This lowers impediments to creativity but presents the threat of misuse, such as the production of inappropriate content.
- **Ethical — Model behaviour and safety:** LMs can hallucinate or produce unstable outputs. Further safety work will be necessary once visual or behavioral generation is introduced.
- **Professional — Responsible engineering practices:** The project follows:
  - modular architecture
  - failure-tolerant design
  - reproducible experiments
  - proper attribution of reused resources

  These practices match professional software engineering expectations.

REFERENCES

[1] A. Filipović, "The role of artificial intelligence in video game development," *Kultura polisa*, vol. 20, pp. 50–67, 11 2023.

[2] M. Victorin, "Towards the creation of believable non-player characters using procedural content generation," Dissertation, Stockholm University, Sweden, 2024.

[3] V. Bulitko, M. Walters, and M. R. G. Brown, "Evolving npc behaviours in a-life with player proxies," in *AIIDE Workshops*, 2018. [Online]. Available: https://api.semanticscholar.org/CorpusID:58005451

[4] H. Ou, "Ai-powered npcs in virtual environments: Creating believable characters through machine learning," *Applied and Computational Engineering*, vol. 145, pp. 96–102, 2025.

[5] G. Yannakakis and J. Togelius, *Artificial Intelligence and Games*, ser. Computer Science. Springer International Publishing, 2018. [Online]. Available: https://books.google.co.uk/books?id=HK1MDwAAQBAJ

[6] Z. Wu, Z. Chen, D. Zhu, C. Mousas, and D. Kao, "A systematic review of generative ai on game character creation: Applications, challenges, and future trends," *IEEE Transactions on Games*, pp. 1–15, 2025.

[7] Z. Hu, Y. Ding, R. Wu, L. Li, R. Zhang, Y. Hu, F. Qiu, Z. Zhang, K. Wang, S. Zhao, Y. Zhang, J. Jiang, Y. Xi, J. Pu, W. Zhang, S. Wang, K. Chen, T. Zhou, J. Chen, Y. Song, T. Lv, and C. Fan, "Deep learning applications in games: A survey from a data perspective," *Applied Intelligence*, vol. 53, no. 24, pp. 31 129–31 164, 2023. [Online]. Available: https://doi.org/10.1007/s10489-023-05094-2

[8] I. Mani, *Computational Modeling of Narrative / by Inderjeet Mani.*, 1st ed., ser. Synthesis Lectures on Human Language Technologies. Cham: Springer International Publishing, 2013.

[9] W. Westera, R. Prada, S. Mascarenhas, P. Santos, J. Dias, M. Guimarães, K. Georgiadis, E. Nyamsuren, K. Bahreini, Z. Yumak, C. Christyowidiasmoro, M. Dascalu, G. Gutu-Robu, and S. Ruseti, "Artificial intelligence moving serious gaming: Presenting reusable game ai components," *Education and Information Technologies*, vol. 25, 01 2020.

[10] R. M. Paweroi, M. Syarawy, and M. Köppen, "A three-tier generative ai workflow for metaverse asset creation," *Cluster Computing*, vol. 28, no. 16, p. 1017, 2025.

[11] N. Kishore and S. Lalitha, "Humanizing npcs-ai driven realistic conversations and reactions," in *2025 IEEE 14th International Conference on Communication Systems and Network Technologies (CSNT)*. IEEE, 2025, pp. 788–793.

[12] J. P. W. Hardiman, D. C. Thio, A. Y. Zakiyyah, and Meiliana, "Ai-powered dialogues and quests generation in role-playing games using google's gemini and sentence bert framework," *Procedia computer science*, vol. 245, pp. 1111–1119, 2024.

[13] Y. Xu, "Ai animation character behavior modeling and action recognition in virtual studio," *International journal of advanced computer science & applications*, vol. 14, no. 10, 2023.

[14] M. Uludagli and K. Oguz, "Non-player character decision-making in computer games," *The Artificial intelligence review*, vol. 56, no. 12, pp. 14 159–14 191, 2023.

[15] S. Khan, "Role of generative ai for developing personalized content based websites," *International Journal of Innovative Science and Research Technology*, vol. 8, pp. 1–5, 09 2023.

[16] C. Ebert and P. Louridas, "Generative ai for software practitioners," *IEEE Software*, vol. 40, no. 4, pp. 30–38, 2023.

[17] M. Kreminski, N. Wardrip-Fruin, R. Rouse, M. Haahr, and H. Koenitz, "Throwing bottles at god: Predictive text as a game mechanic in an ai-based narrative game," in *Interactive Storytelling*, ser. Lecture Notes in Computer Science. Switzerland: Springer International Publishing AG, 2018, vol. 11318, pp. 275–279.

[18] "Personalized quest and dialogue generation in role-playing games: A knowledge graph- and language model-based approach," in *Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems*. New York, NY, USA: ACM, 2023, pp. 1–20.

[19] N. Mehta, "The role of ai in game development and player experience," in *Proceedings of the International Conference on Innovative Computing & Communication (ICICC 2024)*, 2025, p. –, written January 17, 2025; Posted to SSRN February 26, 2025.

[20] M. Claypool and K. Claypool, "Latency and player actions in online games," *Commun. ACM*, vol. 49, no. 11, p. 40–45, Nov. 2006. [Online]. Available: https://doi.org/10.1145/1167838.1167860

[21] M. Korkiakoski, S. Sheikhi, J. Nyman, J. Saariniemi, K. Tapio, and P. Kostakos, "An empirical evaluation of ai-powered non-player characters' perceived realism and performance in virtual reality environments," 2025. [Online]. Available: https://arxiv.org/abs/2507.10469

[22] C. Hu, Y. Zhao, Z. Wang, H. Du, and J. Liu, "Games for artificial intelligence research: A review and perspectives," *IEEE Transactions on Artificial Intelligence*, vol. 5, no. 12, pp. 5949–5968, 2024.

[23] H. Postigo, "Of mods and modders," *Games and Culture*, vol. 2, pp. 300–313, 10 2007.

[24] N. Shaker, J. Togelius, and M. J. Nelson, *Procedural Content Generation in Games*, ser. Computational Synthesis and Creative Systems. Cham: Springer Cham, 2016.

[25] J. Togelius, E. Kastbjerg, D. Schedl, and G. N. Yannakakis, "What is procedural content generation? mario on the borderline," ser. PCGames '11. New York, NY, USA: Association for Computing Machinery, 2011. [Online]. Available: https://doi.org/10.1145/2000919.2000922

[26] J. Togelius, G. N. Yannakakis, K. O. Stanley, and C. Browne, "Search-based procedural content generation: A taxonomy and survey," *IEEE transactions on computational intelligence and AI in games.*, vol. 3, no. 3, pp. 172–186, 2011.

[27] ——, "Search-based procedural content generation," in *Proceedings of the 2010 International Conference on Applications of Evolutionary*

Computation - Volume Part I, ser. EvoApplicatons'10. Berlin, Heidelberg: Springer-Verlag, 2010, p. 141–150. [Online]. Available: https://doi.org/10.1007/978-3-642-12239-2_15

[28] T. Bäck and H.-P. Schwefel, "An overview of evolutionary algorithms for parameter optimization," *Evolutionary Computation*, vol. 1, no. 1, pp. 1–23, 1993.

[29] A. M. Smith and M. Mateas, "Answer set programming for procedural content generation: A design space approach," *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 3, no. 3, pp. 187–200, 2011.

[30] A. Summerville, S. Snodgrass, M. Guzdial, C. Holmgård, A. K. Hoover, A. Isaksen, A. Nealen, and J. Togelius, "Procedural content generation via machine learning (pcgml)," *IEEE Transactions on Games*, vol. 10, no. 3, pp. 257–270, 2018.

[31] X. Mao, W. Yu, K. D. Yamada, and M. R. Zielewski, "Procedural content generation via generative artificial intelligence," 2024. [Online]. Available: https://arxiv.org/abs/2407.09013

[32] S. Alyaseri, "Generative ai in procedural content generation for computer games: Key contributions and trends," in *2025 IEEE Region 10 Symposium (TENSYMP)*, 2025, pp. 1–8.

[33] V. Volz, J. Schrum, J. Liu, S. M. Lucas, A. M. Smith, and S. Risi, "Evolving mario levels in the latent space of a deep convolutional generative adversarial network," *CoRR*, vol. abs/1805.00728, 2018. [Online]. Available: http://arxiv.org/abs/1805.00728

[34] A. Hald, J. S. Hansen, J. Kristensen, and P. Burelli, "Procedural content generation of puzzle games using conditional generative adversarial networks," in *Proceedings of the 15th International Conference on the Foundations of Digital Games*, ser. FDG '20. New York, NY, USA: Association for Computing Machinery, 2020. [Online]. Available: https://doi.org/10.1145/3402942.3409601

[35] S. Mohaghegh, M. A. R. Dehnavi, G. Abdollahinejad, and M. Hashemi, "Pcgpt: Procedural content generation via transformers," 2023. [Online]. Available: https://arxiv.org/abs/2310.02405

[36] T. Zhao and Z. Fan, "Enhancing procedural game level generation using transformer-based neural architectures," in *2024 International Symposium on Internet of Things and Smart Cities (ISITSC)*, 2024, pp. 1–6.

[37] T. Baltrušaitis, C. Ahuja, and L.-P. Morency, "Multimodal machine learning: A survey and taxonomy," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 41, no. 2, pp. 423–443, 2019.

[38] A. Kojima, T. Tamura, and K. Fukunaga, "Natural language description of human activities from video images based on concept hierarchy of actions," *International Journal of Computer Vision*, vol. 50, no. 2, pp. 171–184, 2002. [Online]. Available: https://doi.org/10.1023/A:1020346032608

[39] A. Barbu, A. Bridge, Z. Burchill, D. Coroian, S. Dickinson, S. Fidler, A. Michaux, S. Mussman, S. Narayanaswamy, D. Salvi, L. Schmidt, J. Shangguan, J. M. Siskind, J. Waggoner, S. Wang, J. Wei, Y. Yin, and Z. Zhang, "Video in sentences out," in *Proceedings of the Twenty-Eighth Conference on Uncertainty in Artificial Intelligence*, ser. UAI'12. Arlington, Virginia, USA: AUAI Press, 2012, p. 102–112.

[40] B. Z. Yao, X. Yang, L. Lin, M. W. Lee, and S.-C. Zhu, "I2t: Image parsing to text description," *Proceedings of the IEEE*, vol. 98, no. 8, pp. 1485–1508, 2010.

[41] S. Li, G. Kulkarni, T. L. Berg, A. C. Berg, and Y. Choi, "Composing simple image descriptions using web-scale n-grams," in *Proceedings of the Fifteenth Conference on Computational Natural Language Learning*, ser. CoNLL '11. USA: Association for Computational Linguistics, 2011, p. 220–228.

[42] M. Mitchell, J. Dodge, A. Goyal, K. Yamaguchi, K. Stratos, X. Han, A. Mensch, A. Berg, T. Berg, and H. Daumé III, "Midge: Generating image descriptions from computer vision detections," in *Proceedings of the 13th Conference of the European Chapter of the Association for Computational Linguistics*, W. Daelemans, Ed. Avignon, France: Association for Computational Linguistics, Apr. 2012, pp. 747–756. [Online]. Available: https://aclanthology.org/E12-1076/

[43] D. Elliott and F. Keller, "Image description using visual dependency representations," in *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, D. Yarowsky, T. Baldwin, A. Korhonen, K. Livescu, and S. Bethard, Eds. Seattle, Washington, USA: Association for Computational Linguistics, Oct. 2013, pp. 1292–1302. [Online]. Available: https://aclanthology.org/D13-1128/

[44] J. Thomason, S. Venugopalan, S. Guadarrama, K. Saenko, and R. Mooney, "Integrating language and vision to generate natural language descriptions of videos in the wild," in *Proceedings of COLING 2014, the 25th International Conference on Computational Linguistics: Technical Papers*, J. Tsujii and J. Hajic, Eds. Dublin, Ireland: Dublin City University and Association for Computational Linguistics, Aug. 2014, pp. 1218–1227. [Online]. Available: https://aclanthology.org/C14-1115/

[45] E. Mansimov, E. Parisotto, J. L. Ba, and R. Salakhutdinov, "Generating images from captions with attention," 2016. [Online]. Available: https://arxiv.org/abs/1511.02793

[46] S. Reed, Z. Akata, X. Yan, L. Logeswaran, B. Schiele, and H. Lee, "Generative adversarial text to image synthesis," 2016. [Online]. Available: https://arxiv.org/abs/1605.05396

[47] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial networks," 2014. [Online]. Available: https://arxiv.org/abs/1406.2661

[48] A. Rohrbach, M. Rohrbach, and B. Schiele, "The long-short story of movie description," in *Pattern Recognition: 37th German Conference, GCPR 2015, Aachen, Germany, October 7-10, 2015, Proceedings*. Berlin, Heidelberg: Springer-Verlag, 2015, p. 209–221. [Online]. Available: https://doi.org/10.1007/978-3-319-24947-6_17

[49] R. Anderson, B. Stenger, V. Wan, and R. Cipolla, "Expressive visual text-to-speech using active appearance models," in *2013 IEEE Conference on Computer Vision and Pattern Recognition*, 2013, pp. 3382–3389.

[50] S. Deena and A. Galata, "Speech-driven facial animation using a shared gaussian process latent variable model," in *Advances in Visual Computing*, G. Bebis, R. Boyle, B. Parvin, D. Koracin, Y. Kuno, J. Wang, J.-X. Wang, J. Wang, R. Pajarola, P. Lindstrom, A. Hinkenjann, M. L. Encarnação, C. T. Silva, and D. Coming, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 89–100.

[51] H. Zen, K. Tokuda, and A. W. Black, "Statistical parametric speech synthesis," *Speech Communication*, vol. 51, no. 11, pp. 1039–1064, 2009. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0167639309000648

[52] A. Graves, A.-r. Mohamed, and G. Hinton, "Speech recognition with deep recurrent neural networks," in *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*, 2013, pp. 6645–6649.

[53] W. Chan, N. Jaitly, Q. Le, and O. Vinyals, "Listen, attend and spell: A neural network for large vocabulary conversational speech recognition," in *2016 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2016, pp. 4960–4964.

[54] N. Srivastava and R. R. Salakhutdinov, "Multimodal learning with deep boltzmann machines," in *Advances in Neural Information Processing Systems*, F. Pereira, C. Burges, L. Bottou, and K. Weinberger, Eds., vol. 25. Curran Associates, Inc., 2012. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2012/file/af21d0c97db2e27e13572cbf59eb343d-Paper.pdf

[55] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," 2023. [Online]. Available: https://arxiv.org/abs/1706.03762

[56] J.-B. Alayrac, J. Donahue, P. Luc, A. Miech, I. Barr, Y. Hasson, K. Lenc, A. Mensch, K. Millican, M. Reynolds, R. Ring, E. Rutherford, S. Cabi, T. Han, Z. Gong, S. Samangooei, M. Monteiro, J. Menick, S. Borgeaud, A. Brock, A. Nematzadeh, S. Sharifzadeh, M. Binkowski, R. Barreira, O. Vinyals, A. Zisserman, and K. Simonyan, "Flamingo: a visual language model for few-shot learning," 2022. [Online]. Available: https://arxiv.org/abs/2204.14198

[57] H. Liu, C. Li, Q. Wu, and Y. J. Lee, "Visual instruction tuning," 2023. [Online]. Available: https://arxiv.org/abs/2304.08485

[58] X. Chen, J. Djolonga, P. Padlewski, B. Mustafa, S. Changpinyo, J. Wu, C. R. Ruiz, S. Goodman, X. Wang, Y. Tay, S. Shakeri, M. Dehghani, D. Salz, M. Lucic, M. Tschannen, A. Nagrani, H. Hu, M. Joshi, B. Pang, C. Montgomery, P. Pietrzyk, M. Ritter, A. Piergiovanni, M. Minderer, F. Pavetic, A. Waters, G. Li, I. Alabdulmohsin, L. Beyer, J. Amelot, K. Lee, A. P. Steiner, Y. Li, D. Keysers, A. Arnab, Y. Xu, K. Rong, A. Kolesnikov, M. Seyedhosseini, A. Angelova, X. Zhai, N. Houlsby, and R. Soricut, "Pali-x: On scaling up a multilingual vision and language model," 2023. [Online]. Available: https://arxiv.org/abs/2305.18565

[59] S. Bai, Y. Cai, R. Chen, K. Chen, X. Chen, Z. Cheng, L. Deng, W. Ding, C. Gao, C. Ge, W. Ge, Z. Guo, Q. Huang, J. Huang, F. Huang, B. Hui, S. Jiang, Z. Li, M. Li, M. Li, K. Li, Z. Lin, J. Lin, X. Liu, J. Liu, C. Liu, Y. Liu, D. Liu, S. Liu, D. Lu, R. Luo, C. Lv, R. Men, L. Meng, X. Ren, X. Ren, S. Song, Y. Sun, J. Tang, J. Tu, J. Wan, P. Wang, P. Wang, Q. Wang, Y. Wang, T. Xie, Y. Xu, H. Xu, J. Xu, Z. Yang, M. Yang, J. Yang, A. Yang, B. Yu, F. Zhang, H. Zhang, X. Zhang, B. Zheng, H. Zhong, J. Zhou, F. Zhou, J. Zhou, Y. Zhu, and K. Zhu, "Qwen3-vl technical report," 2025. [Online]. Available: https://arxiv.org/abs/2511.21631

[60] S. J. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach (4th Edition)*. Pearson, 2020. [Online]. Available: http://aima.cs.berkeley.edu/

[61] R. A. Brooks, "Intelligence without representation," *Artificial intelligence*, vol. 47, no. 1-3, pp. 139–159, 1991.

[62] L. P. Kaelbling, M. L. Littman, and A. R. Cassandra, "Planning and acting in partially observable stochastic domains," *Artificial intelligence*, vol. 101, no. 1-2, pp. 99–134, 1998.

[63] N. Nilsson, "Teleo-reactive programs for agent control," *Journal of artificial intelligence research*, vol. 1, pp. 139–158, 1993.

[64] S. J. Russell and E. Wefald, *Do the right thing: studies in limited rationality*. MIT press, 1991.

[65] H. N. Varzeghani and Z. Samadyar, "Intelligent agents: A comprehensive survey," *Inte J Electron Commun Comput Eng*, vol. 5, no. 4, pp. 790–798, 2014.

[66] S. Värtinen, P. Hämäläinen, and C. Guckelsberger, "Generating role-playing game quests with gpt language models," *IEEE Transactions on Games*, vol. 16, no. 1, pp. 127–139, 2024.

[67] F. Martinović, D. Mlinarić, J. Dončević, A. Krajna, and I. Botički, "Towards game level generation through llm and gan," 2025.

[68] F. Neubauer, J. Pleiss, and B. Uekermann, "Ai-assisted json schema creation and mapping," 2025. [Online]. Available: https://arxiv.org/abs/2508.05192

[69] T. Bray, "The javascript object notation (json) data interchange format," Tech. Rep., 2014.

[70] J. Halpern, *Characters, Coroutines, and Spawn Points*. Berkeley, CA: Apress, 2019, pp. 233–275. [Online]. Available: https://doi.org/10.1007/978-1-4842-3772-4_7

[71] M. S. Lewicki and T. J. Sejnowski, "Learning overcomplete representations," *Neural computation*, vol. 12, no. 2, pp. 337–365, 2000.

[72] S. Druckmann and D. Chklovskii, "Over-complete representations on recurrent neural networks can support persistent percepts," *Advances in neural information processing systems*, vol. 23, 2010.

[73] A. Natsev, M. R. Naphade, and J. R. Smith, "Over-complete representation and fusion for semantic concept detection," in *2004 International Conference on Image Processing, 2004. ICIP'04.*, vol. 4. IEEE, 2004, pp. 2375–2378.

[74] T. B. Brown, B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, A. Herbert-Voss, G. Krueger, T. Henighan, R. Child, A. Ramesh, D. M. Ziegler, J. Wu, C. Winter, C. Hesse, M. Chen, E. Sigler, M. Litwin, S. Gray, B. Chess, J. Clark, C. Berner, S. McCandlish, A. Radford, I. Sutskever, and D. Amodei, "Language models are few-shot learners," *CoRR*, vol. abs/2005.14165, 2020. [Online]. Available: https://arxiv.org/abs/2005.14165

[75] A. Rastogi, X. Zang, S. Sunkara, R. Gupta, and P. Khaitan, "Towards scalable multi-domain conversational agents: The schema-guided dialogue dataset," *CoRR*, vol. abs/1909.05855, 2019. [Online]. Available: http://arxiv.org/abs/1909.05855

[76] J. Jiang, K. Zhou, Z. Dong, K. Ye, X. Zhao, and J.-R. Wen, "StructGPT: A general framework for large language model to reason over structured data," in *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, H. Bouamor, J. Pino, and K. Bali, Eds. Singapore: Association for Computational Linguistics, Dec. 2023, pp. 9237–9251. [Online]. Available: https://aclanthology.org/2023.emnlp-main.574/

[77] R. Shin, C. Lin, S. Thomson, C. Chen, S. Roy, E. A. Platanios, A. Pauls, D. Klein, J. Eisner, and B. Van Durme, "Constrained language models yield few-shot semantic parsers," in *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, M.-F. Moens, X. Huang, L. Specia, and S. W.-t. Yih, Eds. Online and Punta Cana, Dominican Republic: Association for Computational Linguistics, Nov. 2021, pp. 7699–7715. [Online]. Available: https://aclanthology.org/2021.emnlp-main.608/

[78] A. Madaan, N. Tandon, P. Gupta, S. Hallinan, L. Gao, S. Wiegreffe, U. Alon, N. Dziri, S. Prabhumoye, Y. Yang, S. Gupta, B. P. Majumder, K. Hermann, S. Welleck, A. Yazdanbakhsh, and P. Clark, "Self-refine: Iterative refinement with self-feedback," 2023. [Online]. Available: https://arxiv.org/abs/2303.17651

[79] I. Higgins, D. Amos, D. Pfau, S. Racaniere, L. Matthey, D. Rezende, and A. Lerchner, "Towards a definition of disentangled representations," 2018. [Online]. Available: https://arxiv.org/abs/1812.02230

[80] R. Rombach, A. Blattmann, D. Lorenz, P. Esser, and B. Ommer, "High-resolution image synthesis with latent diffusion models," *CoRR*, vol. abs/2112.10752, 2021. [Online]. Available: https://arxiv.org/abs/2112.10752

[81] P. Dhariwal and A. Nichol, "Diffusion models beat gans on image synthesis," 2021. [Online]. Available: https://arxiv.org/abs/2105.05233

[82] A. Radford, J. W. Kim, C. Hallacy, A. Ramesh, G. Goh, S. Agarwal, G. Sastry, A. Askell, P. Mishkin, J. Clark, G. Krueger, and I. Sutskever, "Learning transferable visual models from natural language supervision," 2021. [Online]. Available: https://arxiv.org/abs/2103.00020

[83] A. Juliani, V.-P. Berges, E. Teng, A. Cohen, J. Harper, C. Elion, C. Goy, Y. Gao, H. Henry, M. Mattar, and D. Lange, "Unity: A general platform for intelligent agents," 2020. [Online]. Available: https://arxiv.org/abs/1809.02627

[84] V. Balazadeh, M. Ataei, H. Cheong, A. H. Khasahmadi, and R. G. Krishnan, "Physics context builders: A modular framework for physical reasoning in vision-language models," 2025. [Online]. Available: https://arxiv.org/abs/2412.08619

[85] W. Kwon, Z. Li, S. Zhuang, Y. Sheng, L. Zheng, C. H. Yu, J. E. Gonzalez, H. Zhang, and I. Stoica, "Efficient memory management for large language model serving with pagedattention," 2023. [Online]. Available: https://arxiv.org/abs/2309.06180

[86] M. Kim, P. Pinyoanuntapong, B. Kim, W. Saad, and D. Calin, "Edge vs cloud: How do we balance cost, latency, and quality for large language models over 5g networks?" in *2025 IEEE Wireless Communications and Networking Conference (WCNC)*, 2025, pp. 1–6.

[87] E. Li, L. Zeng, Z. Zhou, and X. Chen, "Edge ai: On-demand accelerating deep neural network inference via edge computing," *IEEE Transactions on Wireless Communications*, vol. 19, no. 1, pp. 447–457, 2020.

[88] A. Hussain, H. Shakeel, F. Hussain, N. Uddin, and T. L. Ghouri, "Unity game development engine: A technical survey," *Univ. Sindh J. Inf. Commun. Technol*, vol. 4, no. 2, pp. 73–81, 2020.

[89] J. Gregory, *Game engine architecture*. AK Peters/CRC Press, 2018.

[90] D. Calabrese, *Unity 2D game development*. Packt publishing, 2014.

[91] R. Hare and Y. Tang, "Player modeling and adaptation methods within adaptive serious games," *IEEE Transactions on Computational Social Systems*, vol. 10, no. 4, pp. 1939–1950, 2023.

[92] H. Sahota, "Introduction to prompt templates in langchain," *Heartbeat*, 2023.

[93] S. Risi and J. Togelius, "Increasing generality in machine learning through procedural content generation," 2020. [Online]. Available: https://arxiv.org/abs/1911.13071

[94] R. Hunicke, "The case for dynamic difficulty adjustment in games," in *Proceedings of the 2005 ACM SIGCHI International Conference on Advances in Computer Entertainment Technology*, ser. ACE '05. New York, NY, USA: Association for Computing Machinery, 2005, p. 429–433. [Online]. Available: https://doi.org/10.1145/1178477.1178573

[95] T. Schick, J. Dwivedi-Yu, R. Dessì, R. Raileanu, M. Lomeli, L. Zettlemoyer, N. Cancedda, and T. Scialom, "Toolformer: Language models can teach themselves to use tools," 2023. [Online]. Available: https://arxiv.org/abs/2302.04761