AI-Powered Storyboard to 3D Scene Generation: A Comparative Analysis of Vision-Language Models for Iterative Positioning in Unreal Engine

A Thesis

Submitted to the Faculty

of

Drexel University

by

Tyler Varacchi

in partial fulfillment of the

requirements for the degree

of

Master of Science in Digital Media

2025

ACKNOWLEDGMENTS

I would like to express my sincere gratitude to my advisor, Prof. Emil Polyak, for his guidance and support throughout this research.

I would also like to thank my committee members, Prof. Tony Rowe and Prof. Michael Wagner, for their valuable feedback and insights.

In the preparation of this thesis, I utilized AI-assisted tools, including Grammarly, ChatGPT-4o, and Claude Sonnet 4.5 Extended Thinking, for limited editorial and coding assistance. These tools did not generate or interpret any original research findings. For a detailed record of how these tools were used, please refer to Appendix B.

Contents

List of Tables

List of Figures

- **Anthropic Claude API**: Anthropic. (2025). *Claude Sonnet 4.5 Extended Thinking API* [Cloud AI service]. https://www.anthropic.com/api
- **Depth-Anything-V2**: Yang, L., Kang, B., Huang, Z., Zhao, H., Xu, Z., Zhao, F., & Mei, T. (2024). *Depth Anything V2* [Monocular depth estimation model]. https://github.com/DepthAnything/Depth-Anything-V2
- **DUSt3R**: Wang, S., Leroy, V., Cabon, Y., Chidlovskii, B., & Revaud, J. (2024). *DUSt3R: Geometric 3D Vision Made Easy* [3D reconstruction model]. https://arxiv.org/abs/2312.14132
- **Git**: Software Freedom Conservancy. (2025). *Git* (Version 2.51.2) [Version control system]. https://git-scm.com/
- **LLaVA-13B**: Liu, H., Li, C., Wu, Q., & Lee, Y. J. (2023). *LLaVA-13B* (Version 13B) [Vision-language model]. https://llava-vl.github.io/
- **NumPy**: Harris, C. R., et al. (2025). *NumPy* (Version 2.3.4) [Scientific computing library]. https://numpy.org/
- **Ollama**: Ollama Team. (2025). *Ollama* (Version 0.12.10) [Local LLM inference server]. https://ollama.ai/
- **OpenAI API (ChatGPT-4o)**: OpenAI. (2025). *ChatGPT-4o API* [Cloud AI service]. https://platform.openai.com/docs/models/gpt-4o
- **OpenCV**: Bradski, G. (2000). *OpenCV* (Version 4.12.0) [Computer vision library]. https://opencv.org/
- **pandas**: McKinney, W. (2025). *pandas* (Version 2.3.3) [Data analysis library]. https://pandas.pydata.org/
- **Pillow (PIL)**: Clark, A. (2025). *Pillow* (Version 12.0.0) [Image processing library]. https://python-pillow.org/

- **PyTorch**: Paszke, A., et al. (2019). *PyTorch* (Version 2.9.0) [Deep learning framework]. https://pytorch.org/
- **Python**: Python Software Foundation. (2025). *Python* (Version 3.13.6) [Programming language]. https://www.python.org/
- **Qt / PySide6**: The Qt Company. (2025). *Qt for Python (PySide6)* (Version 6.10.0) [Software library]. https://doc.qt.io/qtforpython/
- **requests**: Reitz, K., & Cordasco, I. (2025). *requests* (Version 2.32.5) [HTTP library for Python]. https://requests.readthedocs.io/
- **Transformers**: Wolf, T., et al. (2020). *Transformers* (Version 4.56.1) [State-of-the-art NLP library]. https://huggingface.co/transformers/
- **Unreal Engine**: Epic Games. (2025). *Unreal Engine* (Version 5.6) [Computer software]. https://www.unrealengine.com/

- **AI Score Hallucination** Systematic over-estimation of positioning quality by vision-language models, where self-reported confidence scores exceed actual positioning success rates. See Section 5.4.

- **Automated Positioning** The system's primary value: users can initiate positioning processes that run unattended (including overnight), returning to completed 3D scenes. Enables workflows for users with Unreal Engine knowledge but limited layout skills. Processing time: 7.8-69.1 minutes per panel depending on AI provider, compared to 1.7 minutes for expert manual positioning.

- **Binary Score Distribution** Pattern in Claude Sonnet 4.5 Extended Thinking's scoring behavior 41.7% of panels scored 100/100, 58.3% scored 70-78/100, with zero scores in the 80-89 range. Contrasts with ChatGPT-4o and LLaVA-13B's narrow unimodal clustering at 80-85/100.

- **BLIP-2 (Bootstrapping Language-Image Pre-training)** A vision-language model using frozen pretrained vision encoders trained on photorealistic datasets. Aligns visual and textual representations through bootstrapped training methods.

- **Calibration (Confidence Calibration)** Alignment between an AI model's predicted confidence and its actual accuracy. Modern deep neural networks are often poorly calibrated, showing overconfidence.

- **Claude Sonnet 4.5 Extended Thinking** Anthropic's flagship vision-language model with Extended Thinking mode.

- **CLIP (Contrastive Language-Image Pre-training)** A foundational vision-language model by OpenAI that jointly trains vision and language encoders on image-text pairs using contrastive learning to align visual and linguistic representations.

- **Closed-Loop Iterative Positioning** Positioning approach where AI models analyze rendered 3D scenes, compare them to 2D target compositions, and propose corrective adjustments

across multiple cycles until reaching the system-defined 80% match score threshold or maximum iterations (30)

- **Complexity Stratification** Categorization of storyboard panels into three levels based on geometric challenges: Simple (single character, clear depth), Medium (2-3 characters with moderate spatial complexity), and Complex (severe overlap, depth ambiguity).

- **Depth Ambiguity** Challenge where multiple 3D configurations can produce identical 2D projections. For example, a character 50 units vs 150 units from the camera may both appear "in front" from the hero camera perspective, requiring multi-angle analysis to resolve it.

- **ChatGPT-4o** OpenAI's multimodal flagship model extending GPT-4 with visual input processing.

- **Hero Camera** Primary viewpoint matching the intended final composition, positioned to replicate the storyboard's perspective. Distinct from scout cameras; provides the definitive comparison image against the target storyboard.

- **Iterative Refinement** Core algorithmic approach of adjusting 3D positions through refinement cycles guided by visual feedback. Measured average iterations across 12 test panels: LLaVA-13B 7.8, ChatGPT-4o 12.5, Claude Sonnet 4.5 Extended Thinking 22.2.

- **Level Sequence** Unreal Engine's non-destructive keyframe animation system managing scene state through timeline.

- **LLaVA-13B (Large Language and Vision Assistant)** Open-source 13-billion parameter vision-language model using CLIP vision encoder + linear projection + LLaMA transformer.

- **Match Score** AI-reported visual similarity between generated 3D scene and target storyboard, scaled 0-100%. System stops iteration when score exceeds hardcoded 80% threshold or maximum iteration count (30) is reached. Primary metric for positioning quality assessment.

- **Minimum Viable Product (MVP)** Classification of the StoryboardTo3D system: a functional prototype showing that vision-language models can perform storyboard-to-3D positioning tasks. Good enough for empirical evaluation of AI capabilities and limitations, but not production-ready for studio deployment.

- **Multi-Angle Capture** Seven-camera sequential capture system using six scout cameras (front, back, left, right, top, 3/4 view) plus the hero camera. It provides comprehensive spatial context for AI analysis. It enables detection of depth errors, occlusion issues, and rotation misalignments not visible from hero camera alone.

- **Panel 9 Case Study** Bench scene where all three models failed positioning yet reported divergent confidence scores, exemplifying calibration differences between vision-language models. See Section 5.4.

- **Positioning Accuracy** Primary outcome metric measuring expert-validated success using binary criteria (character visibility, camera angle correctness, spatial relationship match). See Section 5.1.2.

- **Research Through Design (RTD)** Methodology where the StoryboardTo3D plugin produces knowledge and understanding through design.

- **Scout Camera** Programmatically spawned cameras surrounding scene center at six orthogonal positions (front, back, left, right, top, 3/4 view), each positioned 400-600 Unreal Units from center with the correct field of view. Used for multi-angle capture; distinct from the hero camera which provides the final composition viewpoint.

- **Score Variance** Standard deviation of AI-reported match scores across test panels, revealing different scoring philosophies between models despite similar mean scores. See Section 5.4.3.

- **Sequencer (Unreal Engine)** Keyframe animation system managing positioning through timeline tracks.

- **Storyboard** Hand-drawn visual blueprint for sequences, foundational in animation and film production since Walt Disney Studios in the 1930s. It communicates character positioning, camera angles, spatial composition, and emotional tone through 2D sketches.

- **StoryboardTo3D** Unreal Engine 5.6 plugin automating storyboard-to-3D conversion using vision-language models and iterative positioning refinement.

- **T-Pose** Standard character pose with arms extended horizontally and legs straight, used throughout thesis experiments as static baseline.

- **Texture Bias** Phenomenon in computer vision models where predictions rely primarily on texture-based features rather than shape or structural information. Limits performance on non-photorealistic inputs.

- **Vision-Language Model (VLM)** Neural network architectures combining visual understanding with natural language reasoning capabilities. Extends language models with vision encoders to process and understand multimodal inputs.

- **Vision Transformer (ViT)** Neural network architecture that applies transformer models to computer vision by treating images as sequences of patches processed through self-attention.

ABSTRACT

AI-Powered Storyboard to 3D Scene Generation: A Comparative Analysis of Vision-Language Models for Iterative Positioning in Unreal Engine

Tyler Varacchi

Hand-drawn storyboards enable rapid visualization and critique of narrative concepts in animation and game development. However, translating these 2D sketches into 3D scenes requires considerable manual effort for camera placement, object positioning, and depth perception. Existing research prototypes convert text-based screenplays into preliminary synthetic scenes but do not efficiently translate visual storyboard sketches into 3D layouts using production asset libraries.

This thesis presents StoryboardTo3D, a novel Unreal Engine 5.6 plugin that automates the translation of hand-drawn storyboard panels into fully blocked-out 3D scenes. The plugin employs ChatGPT-4o, LLaVA-13B, and Claude Sonnet 4.5 Extended Thinking for visual analysis, implementing a multi-angle capture system using seven viewpoints (one scout camera repositioned to six angles plus one hero camera, captured sequentially) to provide spatial context for AI positioning. Iterative positioning refinement with AI feedback loops progressively adjusts object placement.

This research methodology combined Research Through Design with quantitative performance analysis and qualitative assessment. Development progressed through systematic testing of multiple AI backends. The plugin was created as a minimum viable product with an architecture to scale in the future.

This research contributes both a minimum viable product that advances previsualization and novel insights into applying modern AI vision capabilities to creative production pipelines. This work also demonstrates the potential of AI-assisted 3D scene generation from sketches, while documenting the associated failures, designs, and architectures for future production.

Keywords: Artificial Intelligence, Vision-Language Models, Storyboard Analysis, 3D Scene Gen-

eration, Unreal Engine, Iterative Positioning, ChatGPT-4o, Claude Sonnet 4.5 Extended Thinking, LLaVA-13B, Previsualization, Animation Pipeline

Chapter 1:
INTRODUCTION

This chapter lays the foundation for understanding AI-powered storyboard-to-3D scene generation. Section 1.1 explains the production workflow challenges that motivated this research. It examines how hand-drawn storyboards remain central to animation and game development. Section 1.2 identifies specific technical gaps in existing approaches, particularly the absence of systems using modern vision-language models for storyboard interpretation. Section 1.3 presents the core questions driving this research. Some questions include whether AI models trained on photorealistic imagery can handle sketch-style input and how accurately these models assess their own positioning decisions. Section 1.4 frames the technical and empirical findings, from the StoryboardTo3D plugin architecture as well as AI score hallucination. Finally, Section 1.5 provides a roadmap of the remaining chapters.

1.1 Background and Motivation

Studios still use hand-drawn storyboards as their blueprint for animation and game development before committing to full production. However, translating these 2D sketches into 3D scene layouts creates workflow bottlenecks. Chapter 2 details these production workflow challenges and technical requirements.

Vision-language models now enable storyboard interpretation (Chapter 2). Applying these models to storyboard-to-3D raises many questions. Can VLMs trained primarily on photorealistic images handle artistic sketches with abstract geometry, loose linework, and stylization? How accurately can AI models judge whether a 3D scene matches a 2D storyboard target? This capability is important for iterative systems that rely on AI self-assessment to know when they have completed positioning. Do different commercial VLM providers (OpenAI, Anthropic) and open-source

alternatives (LLaVA-13B) perform comparably, or do the fundamental differences in architecture, training data, and calibration lead to poor or contrasting positioning quality or reliability?

This thesis presents *StoryboardTo3D*, a minimum viable product (MVP) plugin for Unreal Engine 5.6 (Epic Games, 2025) that automates storyboard-to-3D translation using vision-language models and iterative positioning refinement. Evaluation across three AI models (ChatGPT-4o, Claude Sonnet 4.5 Extended Thinking, LLaVA-13B) using 36 test scenarios revealed significant differences in positioning accuracy and self-assessment calibration. The research demonstrates that while AI-powered positioning is technically feasible, substantial calibration gaps exist between AI confidence scores and actual positioning quality, which we term AI score hallucination.

1.2 Problem Statement

The main technical challenge in automated storyboard-to-3D conversion is resolving the ambiguity of projecting 2D artistic representations into 3D spatial coordinates. A storyboard sketch drawn with foreshortening cues implies depth relationships. However, multiple 3D configurations can produce nearly identical 2D projections. For example, if a character is drawn in front of a bench, it could be positioned 50 or 150 units away from the bench. Both configurations may appear correct from the hero camera viewpoint, but they have different spatial relationships when viewed from other angles.

This research addresses three challenges:

1.2.1 Challenge 1: 2D-to-3D Depth Inference Ambiguity

Hand-drawn storyboards lack precise depth information. Artists use visual cues to suggest spatial relationships: overlap, foreshortening, relative scale. These cues may be too ambiguous for the AI. An AI system must infer 3D positions from incomplete information, which requires spatial reasoning that can interpret ambiguity.

Current computer vision depth estimation models (Ranftl et al., 2021) (monocular depth prediction, stereo reconstruction) are trained on photorealistic images and may perform poorly on artistic

sketches. Sketch-to-3D systems exist, but they typically reconstruct individual objects rather than creating multi-object scenes with character positioning and camera placement using existing assets.

1.2.2 Challenge 2: AI Score Hallucination and Unreliable Self-Assessment

The iterative positioning system requires AI models not only to propose adjustments but also to accurately assess when the generated 3D scene sufficiently matches the target storyboard. If models show AI score hallucination, assigning high match scores to flawed compositions, the system risks accepting inaccurate scenes as complete.

Prior research on AI confidence calibration has documented biases in the self-reported confidence of large language models (Kadavath et al., 2022). However, confidence calibration in vision-language models performing spatial reasoning tasks appears to be largely unexplored. If different VLM providers show different calibration behaviors, then systems must account for model-specific reliability when creating convergence thresholds and validation workflows.

1.2.3 Challenge 3: Multi-Model Performance Tradeoffs

The landscape of vision-language model providers presents production studios with competing options. Commercial API services (OpenAI's ChatGPT-4o, Anthropic's Claude Sonnet 4.5 Extended Thinking) offer cutting-edge performance but have per-request costs and API latency, while open-source local models (LLaVA-13B running on Ollama) provide privacy advantages and zero cost but potentially lower accuracy or reliability.

Understanding the trade-off between accuracy, computational efficiency, cost, privacy, and calibration reliability across these providers is very important for deployment decisions.

1.3 Research Questions

This thesis investigates the following research questions:

1. **RQ1: Convergence Behavior and Scene Complexity** — Do vision-language models achieve

reliable convergence in iterative storyboard positioning workflows, and how do iteration requirements and convergence rates vary with scene complexity?

2. **RQ2: Multi-Model Comparison** — How do commercial VLMs (ChatGPT-4o, Claude Sonnet 4.5 Extended Thinking) and open-source alternatives (LLaVA-13B) compare across AI-reported match scores, iteration efficiency, computational cost, and convergence reliability?

3. **RQ3: Self-Assessment Calibration** — Are VLM self-reported match scores calibrated with actual scene quality, and do different models exhibit systematic biases (overconfidence, conservatism, binary score distribution, AI score hallucination)?

4. **RQ4: MVP Evaluation** — What are the practical considerations for integrating AI-powered storyboard positioning into production animation and game development workflows, including failure analysis, cost-benefit tradeoffs, and human-AI collaboration strategies?

1.4 Contributions

This research makes the following contributions:

1.4.1 System Contribution: Minimum Viable Product for Storyboard-to-3D Positioning

The StoryboardTo3D plugin demonstrates automated storyboard-to-3D conversion within Unreal Engine using vision-language models. The plugin architecture includes modular AI provider abstraction supporting multiple backends (ChatGPT-4o, Claude Sonnet 4.5 Extended Thinking, LLaVA-13B), comprehensive error handling, and automated metrics collection. The system's primary value centers on time-shifted automation: users can initiate positioning processes that run unattended (including overnight) and return to completed 3D scenes, saving human labor time despite longer processing time.

1.4.2 Research Contribution: AI Score Hallucination

Comprehensive expert validation reveals systematic AI score hallucination: vision-language models assign high confidence scores to objectively inaccurate spatial compositions. Different models exhibit distinct calibration patterns, with some demonstrating near-perfect calibration while others show massive over-confidence. This finding has implications for any AI system employing autonomous termination logic based on self-assessment, particularly in spatial reasoning tasks.

1.4.3 Architecture Contribution: Multi-Angle Capture Framework

The seven-viewpoint sequential capture system provides comprehensive spatial context for AI analysis, aiming to expose positioning errors invisible from single viewpoints. The work documents the architecture and provides baseline measurements for future validation experiments.

1.4.4 Methodological Contribution: MVP Evaluation

This thesis provides rigorous evaluation of AI capabilities, acknowledging system limitations alongside successes. The assessment methodology documents failures, calibration issues, and practical deployment considerations, establishing the value proposition as time-shifted cost savings through unattended execution that may democratize access. This approach provides a model for rigorous AI system evaluation that assesses practicality and performance tradeoffs rather than presenting only positive results.

1.5 Thesis Organization

The remainder of this thesis is organized as follows:

- **Chapter 2: Literature Review** — Reviews prior research in AI-assisted content creation, vision-language models for spatial reasoning, storyboard analysis systems, and confidence calibration in AI systems.

- **Chapter 3: Methodology** — Presents the research methodology combining Research Through Design with quantitative performance evaluation, describes the system architecture and AI provider abstraction layer, and explains the design for multi-model comparison.

- **Chapter 4: Implementation** — Describes the iterative positioning algorithm, adjustment application pipeline, multi-angle capture framework, and minimum viable product error handling.

- **Chapter 5: Results** — Gives comprehensive results including overall positioning accuracy (Section 5.1), AI backend comparison (Section 5.2), iteration efficiency and multi-angle capture impact (Section 5.3), and AI self-assessment reliability findings (Section 5.4).

- **Chapter 6: Discussion** — Interprets results, analyzes AI score hallucination findings, provides model-specific deployment recommendations, and addresses research limitations.

- **Chapter 7: Conclusion** — Summarizes contributions and key findings, discusses practical impact for studios and AI system design, outlines future research directions including animation integration, and provides closing remarks on AI as an enabling technology for creative expression.

Chapter 2:

LITERATURE REVIEW

This chapter discusses prior work across several areas: vision-language models for scene understanding, storyboard and previsualization systems, AI-assisted content creation workflows, iterative refinement in spatial reasoning tasks, multi-view 3D understanding, and confidence calibration in neural networks. Together, these areas provide the technical base for the storyboard positioning system and clarify how this work differs from existing approaches.

2.1 Storyboard and Previsualization Systems

2.1.1 Historical Development of Storyboarding

Storyboards have been in use as a core tool in animation and film production since their introduction at Walt Disney Studios in the 1930s (Canemaker, 1999). Webb Smith, a Disney animator, pioneered the technique by pinning sequential drawings to a bulletin board to visualize a story's arc. This approach transformed animation by allowing directors and animators to previsualize sequences, experiment with shot composition, and find pacing issues before committing resources to full animation.

The storyboarding process at Disney soon became standard across animation, live-action film, television, and eventually game development. A typical storyboard panel communicates many layers of information: character positioning and spatial relationships, camera angle and framing, composition, action and motion within the shot, and emotional tone. This visual language helps quick iteration during pre-production. It allows teams to explore multiple narratives before production begins.

In modern production pipelines, storyboards serve as the blueprint for most departments. Layout

artists interpret storyboard panels to construct 3D scenes in digital content creation tools (Maya, Blender, Unreal Engine). Cinematographers then use storyboards to plan camera movement and lighting, and then animators reference storyboards to understand character blocking and action choreography. The efficiency of this pipeline depends on the accuracy with which layout artists can translate 2D sketches into 3D arrangements.

2.1.2 Manual Translation Challenges

Translating storyboard panels into 3D scenes involves some technical challenges. The main challenge stems from inherent ambiguity when projecting 2D artistic representations into 3D scenes. A storyboard drawing shows depth through artistic cues: overlap (objects in front obscure objects behind), foreshortening (objects farther away appear smaller), atmospheric perspective (distant objects have reduced contrast), and relative scale. These cues may be insufficient to determine 3D positions using AI tools.

Here is an example of this challenge; a simple two-character storyboard panel where Character A appears in front of Character B. The 2D sketch provides no information about the depth separation: Character B could be 50 units behind A or 500 units behind A, and both configurations could produce identical 2D projections from the intended camera angle. Layout artists must navigate these ambiguities through technical knowledge and creative interpretation.

This translation process requires expertise across multiple areas. Artists need proficiency in many areas: 3D software interfaces, field of view, focal length, and perspective. Artists must also possess spatial reasoning skills, and preserve the compositional intent of the original storyboard.

For non-3D software users, this workflow may create a barrier. This group may include storyboard artists without technical computer training, but it can also extend to companies that have a limited budget and resources to hire a layout artist. These companies may include independent game developers, and small animation studios. The alternative of outsourcing work to technical artists could run up costs and introduces communication overhead. These gaps may motivate people to work on AI-assisted automation.

2.1.3 Previsualization Research and Automation Attempts

Many researchers have explored automated approaches in storyboard-to-3D conversion and previsualization workflows (Kim et al., 2019). Early work in the 1990s and 2000s focused on text-based screenplay parsing. They used natural language processing to extract scene descriptions, character lists, and action sequences from scripts. Then it would generate preliminary 3D layouts with basic character proxies and environmental geometry. Systems like ScriptViz (Z.-Q. Liu & Leung, 2006) and CONFUCIUS (Ma & McKevitt, 2006) showed proof of concept text-to-scene generation. Still, they were hindered by limitations: screenplay text provides minimal spatial information (descriptions like "INT. COFFEE SHOP - DAY" specify location but not composition), natural language is ambiguous when discussing positioning details (phrases like "John stands near the window" lack precise coordinates), and text-based systems cannot use the visual information present in storyboard sketches.

More recent approaches have introduced computer vision techniques. Sketch-based modeling systems allow artists to create 2D sketches that are converted into 3D geometry. However, these systems mainly focus on reconstructing individual objects rather than composing multi-object scenes with camera placement. Image-based rendering techniques can generate novel views from 2D input, but they create pixel data rather than constructing a complete 3D scene in a production engine.

It seems as though a gap exists in prior research: no prior system combines visual storyboard analysis with iterative 3D positioning within production game engines, while also utilizing 3D asset libraries within that engine. Existing text-to-3D systems may overlook visual information. However, existing sketch-to-3D systems focus on single objects rather than entire scenes. This thesis addresses this gap, demonstrating that modern vision-language models can bridge the 2D-to-3D translation challenge through iterative visual feedback loops.

2.1.4 Production Requirements and Constraints

Current production has constraints that may be absent from academic prototypes and MVPs. Production ready systems must integrate with existing asset libraries: specific character rigs, props, and materials. They must also respect engine-specific conventions (Unreal Engine 5.6's coordinate systems, Sequencer timeline format), maintain reproducibility, and provide artist-friendly error handling and debugging interfaces. These requirements distinguish production systems from research demonstrations.

2.2 Vision-Language Models

Advances in vision-language models over the past few years have enabled AI systems to reason about visual content through natural language. This section examines the core architectures and capabilities most relevant to spatial reasoning, focusing on three models used in this thesis: ChatGPT-4o, Claude Sonnet 4.5 Extended Thinking, and LLaVA-13B.

2.2.1 CLIP and Foundational Vision-Language Alignment

CLIP (Contrastive Language-Image Pre-training) (Radford et al., 2021) marked a large shift in how vision and language systems learn. Trained on 400 million image-text pairs scraped from the web (Radford et al., 2021, Abstract), CLIP employs contrastive learning: given a batch of N image-text pairs, CLIP learns to maximize similarity between the N correct pairs while minimizing similarity between the N^2 - N incorrect pairings. This training objective forces the model to learn semantic alignment between visual and textual representations without requiring manually annotated datasets.

The architecture consists of two encoder networks: a vision encoder (either a ResNet or Vision Transformer) that processes images into embedding vectors, and a text encoder (a Transformer) that processes captions into embedding vectors. During training, these embeddings are projected into a shared multimodal space where semantically related images and text have high cosine similarity.

At inference time, CLIP can perform zero-shot image classification by computing the similarity between an image embedding and text embeddings of candidate class names.

The significance of CLIP extends beyond its specific architecture. It demonstrated that large-scale web-scraped data, despite being noisy and unstructured, contains sufficient signal for learning meaningful vision-language representations. The model achieved competitive ImageNet classification performance without ever seeing labeled ImageNet training examples, establishing the viability of foundation models for computer vision. These models generalize across diverse tasks without task-specific fine-tuning.

For spatial reasoning tasks like storyboard positioning, CLIP's vision encoder provides a starting point for extracting semantic features from hand-drawn sketches. However, CLIP alone proves insufficient. CLIP performs similarity-based matching rather than generating structured output (coordinates, transformations). It also lacks the reasoning capabilities needed to understand spatial relationships and propose positioning adjustments. These limitations may have motivated the development of large vision-language models that combine CLIP-style vision encoding with language model reasoning.

2.2.2 Large Vision-Language Models: Architecture and Capabilities

ChatGPT-4o (OpenAI, 2024), Claude Sonnet 4.5 Extended Thinking (Anthropic, 2025), and LLaVA-13B (H. Liu, Li, Li, & Lee, 2023; H. Liu, Li, Wu, & Lee, 2023) represent current state-of-the-art approaches to multimodal understanding. These models extend pure language and limited visual input models (GPT-4, Claude 2, LLaMA) with vision capabilities, enabling them to analyze images while maintaining reasoning abilities.

ChatGPT-4o Architecture and Capabilities

ChatGPT-4o extends OpenAI's GPT-4 language model with full visual input processing. GPT-4 did have visual input capabilities but they were not as extensive as ChatGPT-4o. While OpenAI has not published detailed architectural specifications, the model likely follows patterns established in

prior multimodal work.First a vision encoder (potentially CLIP-based) extracts patch-level features from input images, then a projection layer maps vision embeddings into the language model's token embedding space, and the GPT-4 Transformer processes the total sequence of image tokens and text tokens autoregressively.

The "o" in ChatGPT-4o means omni, reflecting the model's ability to handle multiple modalities including images, audio, and text in a unified model. For spatial reasoning, ChatGPT-4o demonstrates several key capabilities: detailed scene description, compositional understanding, cross-modal reasoning, and structured output generation (producing JSON-formatted transforms when prompted properly).

In this thesis, ChatGPT-4o processes multi-angle scene captures (seven viewpoints: six scout camera positions plus one hero camera position of a 3D scene) and generates structured positioning recommendations in JSON format. The model's 128K context window allows analysis of multiple high-resolution images within a single API call, which proves essential for the multi-angle feedback system described in Chapter 4.

Claude Sonnet 4.5 Extended Thinking Architecture

Claude Sonnet 4.5 Extended Thinking (Anthropic, 2025) represents Anthropic's flagship vision-language model. While Anthropic similarly withholds detailed architectural specifications, Claude demonstrates distinct reasoning characteristics compared to ChatGPT-4o. Most notably, Claude offers an "Extended Thinking" mode where the model generates long internal reasoning chains before producing a final output.

Extended Thinking allows the model to write thought processes in a hidden reasoning block before generating the user-visible response. This chain-of-thought mechanism enables more thorough analysis of complex visual inputs. This has potential to improve spatial reasoning accuracy at the cost of increased latency and token consumption. For storyboard positioning tasks, Extended Thinking manifests itself as detailed geometric analysis: Claude reports explicit coordinate measurements ("Character A is positioned at X=150, Y=200, which is 50 units left of

the intended target") and identifies specific failure modes ("The hero camera is positioned BEHIND the characters, causing occlusion by the bench at X=-250").

This deep analysis makes Claude particularly good at failure detection. Claude correctly identifies composition problems that other models overlook, as shown by the Panel 9 case study in Section 5.4. However, Extended Thinking also contributes to Claude's 2–3× higher iteration counts and 3.8× higher costs compared to ChatGPT-4o, a tradeoff explored in Chapter 5.

LLaVA-13B: Open-Source Vision Instruction Tuning

LLaVA (Large Language and Vision Assistant) (H. Liu, Li, Li, & Lee, 2023; H. Liu, Li, Wu, & Lee, 2023) shows that high-quality vision-language capabilities can come from efficient instruction tuning of open-source base models. The architecture is straightforward: a pre-trained CLIP vision encoder provides image embeddings, a simple linear projection layer maps vision embeddings into the LLaMA language model's input space, and the LLaMA Transformer processes the combined vision-language input autoregressively.

The key innovation of LLaVA is not in architectural complexity but in the training methodology. The model was trained on 150K image-instruction pairs generated semi-automatically using GPT-4 (H. Liu, Li, Wu, & Lee, 2023, p. 2, Section 3). Given an image and its caption, GPT-4 generated diverse instruction-response pairs. An example of this is, "Q: What objects are visible in this image? A: The image shows a bench, two characters, and trees in the background". Fine-tuning on these instruction pairs taught the model to follow visual instructions without requiring manually annotated training data at scale.

LLaVA-13B (the 13-billion parameter variant used in this thesis) runs locally via Ollama, eliminating API costs and latency while providing data privacy advantages. Improved LLaVA (H. Liu, Li, Li, & Lee, 2023) enhanced baseline capabilities through better training instructions and larger datasets.

2.2.3 Vision Encoding and Spatial Understanding

All three models process images through vision encoders that breakdown images into patch-level features. For a typical input image of 512×512 pixels processed by a Vision Transformer with 16×16 patches, the encoder produces a sequence of 1,024 patch embeddings (32 patches × 32 patches), each representing local visual features. These patch embeddings capture color, texture, edges, and higher-level semantic features depending on encoder depth.

For spatial reasoning tasks, patch-level encoding provides several advantages. It preserves spatial layout. Unlike fully global image embeddings, patch sequences maintain 2D spatial structure that models can attend to. Multi-scale feature extraction emerges through capturing increasingly abstract spatial relationships—from edges to object boundaries to scene layout. Attention-based reasoning enables self-attention mechanisms in both vision and language transformers. This then enables the model to dynamically focus on relevant image regions when answering spatial questions.

However, patch-based encoding also introduces challenges for precise geometric reasoning. Vision transformers operate on discrete patches rather than continuous coordinates, limiting sub-patch positioning precision. Pre-training on photorealistic images may not transfer optimally to hand-drawn sketches with abstracted geometry and loose linework. 2D patch encodings lack clear 3D geometric elements, forcing models to infer 3D structure.

Despite these limitations, results demonstrate that some modern vision-language models can successfully perform spatial reasoning on storyboard sketches. The positioning accuracy achieved in this thesis (Chapter 5) validates that learned vision representations, combined with language model reasoning, provide a sufficient base for automated storyboard-to-3D conversion.

2.2.4 Domain Adaptation: Photorealistic to Sketch Transfer

An important question for storyboard positioning is whether models trained mainly on photo-realistic images can be used for hand-drawn sketches. Storyboard art has characteristics absent from usual pre-training data. Some characteristics include intentionally abstracted geometry, loose linework and variable line weight, minimal color information (black and white or limited palettes),

and stylistic exaggeration for expressive effect.

The experimental results in Chapter 5 provide evidence that some modern VLMs transfer well from photorealistic to sketch domains. This suggests that the semantic features learned during pre-training for example object boundaries, spatial relationships, compositional patterns are sufficient to transfer across visual areas.

However, failures documented in Section 5.4 reveal domain-specific challenges. Panel 9 (bench scene) proved particularly difficult for all models, with final compositions showing severe occlusion problems despite high AI-reported match scores. Visual inspection suggests that the hand-drawn sketch's ambiguous depth cues (the bench's position relative to characters) caused models to infer spatial arrangements not supported by the 2D input. This failure highlights a fundamental limitation. When 2D sketches contain insufficient depth information, even sophisticated VLMs cannot reliably infer correct 3D layouts.

2.2.5 Limitations of Non-LLM Multimodal/Computer Vision Models for Sketches

While non-LLM multimodal/computer vision models like BLIP-2 (J. Li et al., 2023) and CLIP (Radford et al., 2021), as well as 3D reconstruction methods like DUSt3R (S. Wang et al., 2024), may achieve impressive results on photorealistic images, they exhibit fundamental limitations, mainly due to training distribution mismatches and architectural biases toward texture-based features.

BLIP-2: Training Distribution and Vision Encoder Limitations

BLIP-2 (J. Li et al., 2023) employs frozen pretrained vision encoders trained on photorealistic datasets (COCO, Visual Genome, LAION). Research demonstrates these encoders exhibit strong texture bias, with ImageNet-trained CNNs averaging approximately 22% shape reliance versus 78% texture reliance (Geirhos et al., 2019, Table 1). When tested on silhouettes and edge-based images lacking texture cues, CNN recognition accuracy degrades substantially compared to photorealistic images (Geirhos et al., 2019). Hand-drawn storyboards lack the texture, shading, and photometric

elements BLIP-2 expects, possibly causing hallucinated captions.

CLIP: Texture Bias and Cross-Domain Retrieval Failures

CLIP (Radford et al., 2021) training on 400 million web-scraped image-text pairs (Radford et al., 2021, Abstract) yields approximately 80% texture-based classification decisions (Geirhos et al., 2019). Sketch-based image retrieval research shows significant accuracy drops on sketch style due to severe domain gap between training data (photorealistic images) and application domain (sparse line art) (Pang et al., 2019; Yelamarthi et al., 2018). For storyboard-to-3D asset matching, it is possible CLIP's embedding space may fail to capture cross-modal correspondence between hand-drawn sketches and photorealistic 3D asset thumbnails.

DUSt3R: Photometric Consistency and Depth Ambiguity

DUSt3R (S. Wang et al., 2024) requires photometric elements (shading, texture gradients, lighting) for pointmap regression. Hand-drawn sketches provide sparse line contours without surface properties. This can cause arbitrary depth estimation unrelated to artistic intent. Monocular depth from sketches remains ambiguous (C. Wang et al., 2018), and storyboard conventions like line weight variation represent symbolic depth indicators outside DUSt3R's photometric training.

Large Multimodal Models: Abstract Reasoning Over Pixels

Large multimodal models (ChatGPT-4o (OpenAI, 2024), Claude Sonnet (Anthropic, 2025), LLaVA (H. Liu, Li, Wu, & Lee, 2023)) succeed through language-mediated semantic understanding rather than pixel-level feature extraction. These models draw on world knowledge including cinematographic conventions, spatial relationships, and artistic techniques. Demonstrations show ChatGPT-4o can successfully interpret hand-drawn UI wireframes (Hood, 2023), revealing generalization to sketch-style input. For storyboard positioning, LMMs understand domain-specific visual rules, extract spatial information through semantic reasoning, and support iterative workflows. These are all capabilities that require compositional reasoning. These capabilities are less accessible

to texture-biased vision encoders.

2.3 Multi-View 3D Understanding

The multi-angle capture system employed in this thesis draws inspiration from classical multi-view geometry and modern neural rendering approaches. This section reviews core geometric principles and their use for spatial reasoning tasks.

2.3.1 Multi-View Geometry Foundations

Hartley and Zisserman (2004) established the mathematical understanding of multi-view geometry: how 3D scene structure can be recovered from multiple 2D images with known or unknown camera parameters. The main concepts include epipolar geometry (geometric relationships between corresponding points), triangulation (computing 3D point positions from 2D correspondences across views), and structure-from-motion (simultaneously estimating camera poses and 3D geometry from image sequences).

Classical multi-view geometry infers photometric consistency across views. This assumption allows correspondence matching and geometric reconstruction. Storyboard positioning, however, operates under different principals: rather than reconstructing unknown 3D geometry from images, the system positions known 3D assets to match a 2D sketch target. The role of multi-view analysis is verification and error detection rather than reconstruction.

2.3.2 Multi-Angle Capture for Spatial Error Detection

The multi-angle capture system (one scout camera repositioned to six angles: front, back, left, right, top, 3/4 view; plus one hero camera) enables more comprehensive spatial error detection that may be impossible from a single viewpoint. This design addresses a core challenge in iterative positioning. Optimization using only the hero camera view may result in inaccuracies, where the scene appears correct from the target viewpoint but contains errors that are invisible from that angle.

Imagine a scenario where two characters should stand side-by-side but are accidentally positioned one behind the other along the camera's view. From the hero camera's perspective, both configurations look similar but its actually two characters occupying roughly the same screen positions. However, side-view captures (left/right cameras) immediately show the depth error: one view shows characters correctly aligned, the other shows incorrect positioning. Top-down orthographic views also reveal rotation errors invisible in perspective views, and back camera captures detect occlusion by environmental geometry that would otherwise go unnoticed.

The multi-angle capture system provides comprehensive spatial context at the cost of 6× more images per iteration, larger API token consumption (each image contributes tokens to the VLM context), and increased scene capture time. Multi-angle analysis may enable the detection of errors invisible from single viewpoints, improving positioning accuracy across diverse scene types.

2.3.3 Monocular Depth Estimation

Advances in monocular depth estimation (Ranftl et al., 2021) have demonstrated that neural networks can predict plausible depth maps from single RGB images. Dense Prediction Transformers (DPT) achieve great performance by combining Vision Transformer encoders with multi-scale fusion decoders. These produce sharp depth boundaries and accurate relative depth. More recent models like Depth Anything V2 (Yang et al., 2024) further improves depth estimation through large scale data training.

This thesis integrates depth estimation into VLM reasoning by incorporating Depth Anything V2 into the multi-angle capture pipeline. For each iteration of the positioning loop, the system can generate depth maps for both the hand-drawn storyboard sketch and selected 3D scene viewpoints (up to six scout camera positions plus one hero camera position). An intelligent view selection algorithm adaptively determines which depth maps to generate based on iteration stage, convergence behavior, and scene complexity. The VLM receives up to 16 images in comprehensive mode (storyboard RGB and depth, seven scene RGB captures, and their corresponding depth maps), though typical iterations transmit 6–10 images after intelligent selection optimization. This provides

information to complement semantic understanding from RGB images alone.

The depth maps provide the VLM with foreground and background relationships, relative distance elements, and occlusion information that may be ambiguous in RGB captures alone. Depth Anything V2 performs well on sketched panels, mirroring the detail that the RGB images show.

This approach addresses a core challenge in sketch-to-3D positioning: hand-drawn storyboards contain stylization and ambiguous depth that could make spatial reasoning difficult from RGB or black and white inputs alone. By providing depth information from the rendered 3D scene and story boards, the system gives VLMs measurements to understand and refine their positioning decisions. The multi-angle depth integration is detailed in Section 4.3 (Chapter 4).

2.4 Iterative Refinement and Optimization

Iterative refinement is the process of improving outputs through repeated cycles guided by feedback. This approach has proven effective across various AI domains including code generation, image creation, and robotics. In code generation, models like AlphaCode (Y. Li et al., 2022) iteratively refine solutions by testing against unit tests and adjusting implementations based on failures. In image generation, diffusion models (Ho et al., 2020) progressively denoise samples through iterative denoising steps guided by learned gradients. In robotics, reinforcement learning agents refine control rules through trial-and-error interactions with environments (Schulman et al., 2017).

The storyboard positioning system adapts this iterative idea to spatial reasoning tasks. Rather than refining code syntax or image pixels, the system iteratively adjusts 3D object positions and camera parameters. Each iteration generates multi-angle scene captures, compares them to the target storyboard using vision-language models, and applies corrective adjustments based on AI feedback. This closed-loop process continues until convergence criteria are met (detailed in Chapter 3).

2.5 Confidence Calibration in Neural Networks

Confidence calibration is the alignment between a model's predicted confidence and its actual accuracy. Confidence calibration has major implications for systems that use AI self-assessment for autonomous decision-making. This section assesses calibration research in classification models and language models. This establishes the base for the vision-language model calibration analysis in Section 5.4.

2.5.1 Calibration in Classification Models

Guo et al. (2017) showed that current neural networks are often poorly calibrated, exhibiting overconfidence. In their evaluation of ImageNet classification models, predicted confidence scores usually exceeded accuracy. This shows a model might predict class labels with 90% confidence while achieving only 70% accuracy on those high-confidence predictions.

Guo et al. introduced temperature scaling as a calibration technique. When dividing logits by a learned temperature parameter T before applying softmax, models can be recalibrated. This allows models to match confidence with accuracy. However, this technique applies to classification models with explicit probability outputs and does not directly transfer to vision-language models generating structured text outputs.

Minderer et al. (2021) extended this calibration analysis for larger-scale vision models. Minderer found that calibration degrades as model size increases. This is counterintuitive finding which may suggest that improved accuracy does not automatically yield better confidence calibration. They also found that in-distribution calibration (on datasets similar to training data) differs substantially from out-of-distribution calibration (on shifted or novel data). For storyboard positioning, this may raise concerns. If VLMs are well-calibrated on photorealistic images but poorly calibrated on hand-drawn sketches then depending on their actual distribution, self-reported match scores may be systematically biased.

2.5.2 Self-Assessment in Large Language Models

Kadavath et al. (2022) investigated self-assessment accuracy in large language models. Kadavath found that models have some ability to assess their own knowledge but with systematic biases. In their experiments, they gave Anthropic's Claude model factual questions and then prompted it to estimate its probability of having answered correctly. The model's self-reported confidence correlated positively with actual accuracy, demonstrating some degree of calibration. However, the model showed overconfidence on difficult questions and underconfidence on easy questions.

Kadavath et al. found several factors that influence self-assessment accuracy. First, question difficulty; models struggle to understand confidence on questions that may be on the edge of their knowledge base. Second, prompt phrasing; subtle changes in how confidence is asked (for example, "How confident are you?" vs. "What is the probability you are correct?") produce different calibration curves. Finally, reasoning transparency; models that make clear reasoning chains before answering may demonstrate better calibration than models that answer directly.

These findings have implications for storyboard positioning. The iterative system prompts AI models to provide match scores (0–100%). They assess visual similarity between 3D scenes and storyboards which is a form of self-assessment. If VLMs show calibration biases documented in language-only models then system design needs to account for these factors through specific model thresholds, group AI voting, or human-in-the-loop validation.

2.5.3 Unanswered questions in VLM Calibration

Confidence calibration in vision-language models performing iterative spatial reasoning tasks remains to my knowledge largely unexplored. Key questions include:

1. **Calibration consistency across iterations:** Do VLM confidence scores remain calibrated as iteration progresses, or does calibration degrade after multiple iteration cycles?

2. **Model-specific calibration differences:** Do different VLM providers (OpenAI, Anthropic, open-source models) have comparable calibration, or do architectural and training differences

make different confidence behaviors?

3. **Task-specific calibration transfer:** Models pre-trained on visual question answering may be calibrated for classification-style outputs but miscalibrated for regression-style outputs.

4. **Multi-modal calibration:** Does confidence during vision-language tasks differ from confidence in language-only or vision-only tasks?

Section 5.4 attempts to address these questions through the evaluation of ChatGPT-4o, Claude Sonnet 4.5 Extended Thinking, and LLaVA-13B across 36 storyboard positioning sequences. This evaluation may reveal models show different self-assessment strategies with implications for MVP evaluation.

2.6 AI-Assisted Content Creation

AI tools have begun to augment creative workflows across industries. This section reviews AI-assisted content creation systems relevant to the storyboard positioning domain.

2.6.1 Text-to-3D Generation

Multiple systems now use text-to-3D generation. When a user gives a natural language description, models generate a 3D mesh, textures, and materials. These processes often use diffusion models trained on large-scale 3D datasets (Lin et al., 2023; Poole et al., 2022). They may also use score distillation sampling to optimize 3D representations guided by pre-trained 2D diffusion models (Poole et al., 2022). Text-to-3D offers complementary functionality to storyboard positioning.

A key distinction between these two processes is that text-to-3D focuses on asset creation while this thesis addresses scene layout and composition. Production pipelines may require both capabilities in the future. An example work flow could employ generating, then positioning those assets into sequences matching the storyboard. Future work may integrate text-to-3D asset generation with vision-driven layout positioning for end-to-end storyboard-to-scene workflows.

2.6.2 Sketch-Based 3D Modeling

Sketch-based modeling systems allow artists to draw 2D sketches that are converted into 3D geometry through constraints or known shapes. Earlier systems (Igarashi et al., 1999) relied on clear rules; drawing a closed loop defines a planar face, parallel lines indicate extrusion direction. More recent learning based approaches train networks on sketch mesh paired datasets to predict 3D shapes from drawings.

These systems address the need for sketch-to-3D translation, but they focus more on reconstructing individual objects, such as chairs, cars, and buildings, rather than composing multi-object scenes with camera placement and spatial relationships. The tasks share visual input (sometimes hand-drawn sketches) but differ in output requirements and geometric reasoning.

2.6.3 AI-Assisted Animation Tools

Animation production now employs various AI-assisted tools for motion capture cleanup, character rigging, frame interpolating, and lip-sync automation (Zell et al., 2021). These tools demonstrate how AI can integrate into creative pipelines by augmenting rather than replacing human expertise. The AI handles tedious technical tasks like cleaning noisy mocap data or generating intermediate frames. Artists still retain creative control over high-level direction and artistic refinement.

This shift to augmentation informed the design principals of the storyboard positioning system. Rather than attempting to replace layout artists, the system tries to provide automated positioning. This in turn may help enable non-experts to achieve results previously requiring training. Expert artists remain faster and produce higher-quality results manually, but the AI system expands accessibility. This framing of AI capabilities as augmentation tools aligns with emerging best practices in AI-assisted creative workflows (Amershi et al., 2019).

2.6.4 Procedural Content Generation

Game development has long employed procedural generation systems to algorithmically create environments, levels, and assets (Perlin, 2002). Classic examples include Perlin noise for terrain generation, L-systems for vegetation modeling, and rule-based dungeon layouts. These systems demonstrate how algorithmic automation can scale content creation beyond manual authoring capabilities, enabling vast game worlds and dynamic environments while maintaining artistic coherence through carefully designed generation constraints.

Storyboard positioning shares this human-AI collaboration structure with procedural generation. Artists define high-level compositional intent through storyboard sketches (character relationships, camera angles, spatial composition), AI systems generate preliminary implementations (positioned 3D scenes matching the sketched intent), and artists refine results when necessary. This pattern—where humans specify creative direction while AI handles technical execution—aligns with the system's design philosophy of augmenting rather than replacing artist expertise, positioning automated positioning as an accessibility tool for non-experts rather than a replacement for skilled layout artists.

2.7 Summary and Research Gap

While a lot of progress has been made in vision-language model understanding, iterative AI systems, confidence calibration, and AI-assisted content creation, several gaps remain at the center of these areas. This section amalgamates the literature review, identifying specific research gaps that this thesis addresses.

2.7.1 Identified Research Gaps

Gap 1: Iterative Visual Feedback for 3D Scene Composition

Existing work focuses on text-to-3D generation or single view, single iteration placement. Text-to-3D systems lack granular compositional control. These systems usually generate individual

objects but do not compose multi-object scenes matching specific visual targets. Single-shot positioning systems need to achieve correct results in one attempt, lacking quality and accuracy.

I have no knowledge of prior work that has demonstrated closed-loop iterative positioning where AI models analyze rendered 3D scenes, compares them to 2D target compositions, and proposes corrective adjustments across multiple iteration cycles.

Gap 2: Minimum Viable Product (MVP)-Relevant VLM Comparison

Most VLM research evaluates models on academic benchmarks: visual question answering, image captioning, object detection. These benchmarks measure isolated capabilities instead of end-to-end task performance. These benchmarks give limited insight for production evaluation decisions: a model excelling at visual question answering may struggle with spatial reasoning in iterative positioning workflows. The benchmark accuracy does not reveal cost-efficiency tradeoffs relevant to production budgets.

I couldn't find prior work has compared commercial VLM providers (OpenAI, Anthropic) and open-source alternatives (LLaVA-13B) on production tasks with specific asset libraries, engine constraints, and quality requirements. These types of comparisons need to evaluate many dimensions at the same time: positioning accuracy, iteration efficiency, computational cost, calibration reliability, and failure mode analysis. Understanding these many tradeoffs proves essential for informed model selection in productions.

Gap 3: Confidence Calibration in Iterative Spatial Tasks

Prior calibration work looks at classification tasks (predicted class probabilities vs. actual accuracy) and language model self-assessment. No prior work has examined whether VLM self-assessments remain calibrated across multiple iterations in spatial reasoning tasks. Key unanswered questions include: do confidence scores degrade or improve as iteration progresses? Do models have overconfidence on difficult spatial configurations? Do different VLM architectures produce different calibration behaviors?

These questions have implications for systems using on AI-reported match scores to determine convergence. If models show score inflation, assigning high match scores to inadequate compositions, the system may prematurely stop iterations on an inaccurate scene. Conversely, if models show conservative scoring, assigning low scores to acceptable compositions, the system may waste computational resources on unneeded iterations.

Gap 4: Sketch-Style Storyboard to 3D Scene Translation

A small amount of research addresses translation of hand-drawn storyboards into 3D scenes. Existing sketch-to-3D work focuses on reconstructing individual objects rather than composing multi-object scenes. Storyboard sketches pose unique challenges: linework lacking photorealistic detail, stylization, unclear depth elements requiring inference, and different styles across different storyboard artists.

No prior work has validated whether VLMs trained predominantly on photorealistic images transfer well to sketch-domain inputs for spatial reasoning tasks. This domain transfer question is important: if VLMs cannot generalize from photos to sketches, the entire approach becomes not possible. Conversely, if VLMs show reliable sketch understanding, it may prove the use of foundation models for creative production tasks beyond their original training distributions.

2.7.2 Thesis Positioning and Contributions

This thesis addresses these four gaps through evaluation of three vision-language models (ChatGPT-4o, Claude Sonnet 4.5 Extended Thinking, LLaVA-13B) in a production relevant iterative positioning task within Unreal Engine 5.6. The research provides:

Addressing Gap 1 (Iterative Feedback): Implementation and evaluation of a closed-loop iterative positioning system achieving 84.4% AI self-reported accuracy (47.2% manually validated success) across 36 test scenarios. This may demonstrate VLMs successfully perform multi-iteration spatial refinement with visual feedback. However, it is important to note some significant score hallucinations (+67.1% for ChatGPT-4o, +1.5% for Claude Sonnet 4.5 Extended Thinking).

Addressing Gap 2 (Production Comparison): Comprehensive multi-model comparison across: accuracy, iteration efficiency, cost, convergence reliability, and time requirements per panel. Provides a base for production model selection decisions.

Addressing Gap 3 (Calibration): Analysis of VLM confidence calibration in iterative spatial tasks. This documents core differences in self-assessment strategies: AI Score Hallucination (ChatGPT-4o, LLaVA-13B) vs. binary score distribution (Claude Sonnet 4.5 Extended Thinking). Further analysis reveals a 15-point score inflation gap on identical failed scenes (Claude Sonnet 4.5 Extended Thinking: 70/100 vs. ChatGPT-4o/LLaVA-13B: 85/100), showing that calibration varies widely across providers.

Addressing Gap 4 (Sketch Transfer): VLMs show strong photo-to-sketch transfer for spatial reasoning, with AI models reporting 83-85% self-assessed scores on hand-drawn storyboard inputs, though expert validation revealed lower actual success rates (47.2%). Documented failures (Panel 9 bench occlusion errors) reveal limitations when sketch depth information is ambiguous.

Chapter 3:
METHODOLOGY

3.1 Overview

This chapter outlines the methodology for developing and evaluating the StoryboardTo3D system. The research combines Research Through Design (RTD) with quantitative evaluation, addressing the four research questions outlined in Chapter 1 through experimentation and comparative analysis.

3.1.1 Research Philosophy: Research Through Design

This thesis follows the Research Through Design (RTD) methodology (Frayling, 1993; Zimmerman et al., 2007), where the StoryboardTo3D plugin serves as both research artifact and an evaluation tool. RTD fits this unexplored application domain well: no established storyboard-to-3D frameworks exist. This means iterative development informed by empirical evaluation becomes very important. The approach yields both a functioning tool and insights: VLM self-assessment calibration differences, multi-angle capture as a design pattern for spatial reasoning tasks, and finding that automated positioning enables access without necessarily accelerating expert workflows. These contributions emerge through reflection on the design process. This may give transferable knowledge applicable beyond this specific project.

3.1.2 Rationale for Iterative Positioning Over Direct Generation

A critical architectural decision underpinning this research is the choice of iterative refinement over single-shot generation. While vision-language models could theoretically analyze a storyboard panel and directly output final 3D coordinates in one inference pass, this approach faces challenges:

Representational Gap: Hand-drawn storyboards communicate spatial relationships through

qualitative visual cues (foreshortening, overlap, relative scale) that lack numerical communication to Unreal Engine's system. While vision-language models can generate coordinate estimates, these initial predictions require validation and refinement. The iterative approach reframes positioning as "assess current configuration and propose adjustments" rather than "guess exact coordinates from sketches."

Verifiability and Quality Signals: Directly generated coordinates cannot be validated without rendering anyway, making iteration cost comparable to single-shot generation. Additionally, convergence behavior provides valuable quality indicators. The consistent 100/100 scores may suggest satisfaction, while oscillation or failure to converge signals AI confusion requiring human review. Direct generation offers no such diagnostic information.

The multi-angle capture system (Section 4.3.1) provides comprehensive spatial context, accepting increased per-iteration rendering cost for improved spatial reasoning.

3.1.3 Mixed Methods: Quantitative Metrics with Qualitative Case Studies

The evaluation methodology combines quantitative performance metrics with in-depth qualitative analysis, understanding that numerical accuracy statistics alone provide insufficient insight into system behavior. This mixed-methods approach follows established practices in human-computer interaction research (Creswell & Creswell, 2017), where quantitative measurements establish performance baselines while qualitative investigation reveals underlying mechanisms, failures, and user experience considerations.

The quantitative component (detailed in Section 3.4) employs controlled experimental comparison: three AI backends (ChatGPT-4o, Claude Sonnet 4.5 Extended Thinking, LLaVA-13B) are evaluated on an identical test set of 12 storyboard panels stratified by complexity (simple, medium, complex), producing 36 independent test scenarios. Standardized metrics include: positioning accuracy, iteration count, convergence rate, computational time, API cost. These metrics enable statistical comparison through paired t-tests and ANOVA, addressing RQ1 (positioning accuracy) and RQ2 (multi-model comparison) with empirical evidence.

However, quantitative aggregation alone overshadows important behavioral differences. Mean AI self-reported scores (ChatGPT-4o: 83.8%, Claude Sonnet 4.5 Extended Thinking: 84.8%, LLaVA-13B: 84.6%) suggest near-equivalent self-assessed match quality, but this masks an important divergence in self-assessment strategies revealed through qualitative analysis; Claude Sonnet 4.5 Extended Thinking's binary score distribution versus ChatGPT-4o/LLaVA-13B's score inflation patterns. Detailed case studies (Chapter 5) reveal that score distributions and convergence have information not captured by mean scores alone.

The qualitative component therefore includes:

- **Case Study Analysis:** Deep investigation of specific failure modes (Panel 9 camera occlusion, Panel 11 depth ambiguity) with visual documentation, AI assessment transcripts, and technical diagnosis of root causes.
- **Score Distribution Examination:** Statistical analysis of match score patterns (means, standard deviations, histograms) showing model-specific calibration characteristics.
- **Convergence Arc Analysis:** Temporal plots of accuracy evolution across iterations, identifying monotonic improvement versus oscillation patterns.
- **AI Response Transcript Review:** Close reading of vision-language model textual assessments, identifying patterns (technical measurements versus generic praise) that correlate with scoring behavior.

This combination addresses research questions that may not do well with purely quantitative methods. RQ3 (self-assessment calibration) requires understanding not just whether models are accurate but how their confidence relates to quality. RQ4 (MVP evaluation) demands consideration of practical factors: artist workflow integration, failure recovery strategies, cost-benefit tradeoffs. These cannot be reduced to single performance metrics.

3.1.4 Alignment with Research Questions

The methodology addresses the four research questions (Section 1.3) through controlled experimental comparison (RQ1-RQ2), comprehensive self-assessment analysis (RQ3), and qualitative deployment evaluation (RQ4), as detailed in Section 3.4.

The following sections detail the system architecture (Section 3.2), AI provider abstraction layer (Section 3.3), metrics framework (Section 3.6), and experimental design (Section 3.4) that operationalize this research methodology.

3.2 System Architecture

The StoryboardTo3D system implements a modular architecture with five core components: Qt User Interface, Orchestration Controller, Unreal Engine Integration Layer, AI Provider Abstraction Layer, and Metrics Framework. The architecture follows a Model-View-Controller pattern adapted for AI-in-the-loop interaction, enabling iterative positioning refinement through multi-angle visual feedback. Figure 3.1 illustrates the overall system flow, from storyboard input through the iterative positioning loop to final scene generation. Detailed architectural specifications, component interactions, implementation decisions, and workflow procedures are provided in Chapter 4.

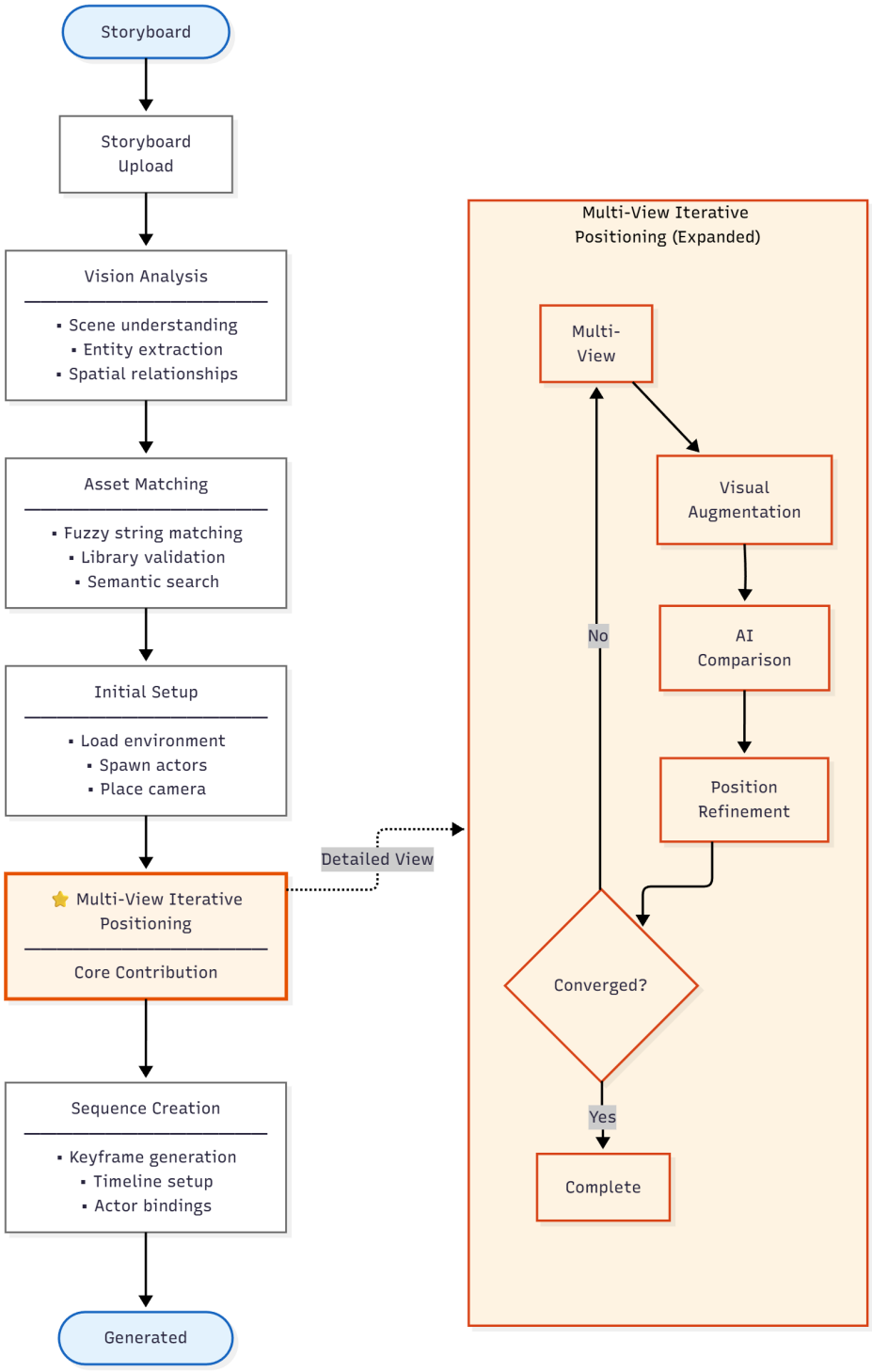


FIGURE 3.1: System architecture flowchart showing the five core components and iterative positioning loop. The pipeline flows from storyboard input through the orchestration controller, which coordinates the AI provider abstraction layer and Unreal Engine integration for iterative refinement until convergence.

3.3 AI Provider Abstraction Layer

The system implements a provider abstraction layer that isolates model-specific communication protocols from core positioning logic which help comparison of multiple AI vision models. This design follows the Strategy pattern (Gamma et al., 1994) which allows runtime substitution of AI backends without modifying orchestration code. The abstraction defines a standardized interface for image analysis, cost tracking, and availability checking. The layer ensures that experimental comparisons remain valid across different types of model architectures (cloud-based vs. local inference).

Listing 1. Abstract AI Provider Interface. This base class establishes a rule set that all AI model integrations must follow, enabling cost tracking, availability probing, and statistical aggregation for cross-model comparison. The abstraction was essential when changing from ChatGPT-4o-only experiments to multi-model comparative studies.

```python
from abc import ABC, abstractmethod
from typing import List, Dict, Optional
from pathlib import Path

class BaseAIProvider(ABC):
    """Abstract base class for all AI vision providers"""

    def __init__(self, name: str):
        self.name = name
        self.last_cost = 0.0
        self.total_cost = 0.0
        self.call_count = 0

    @abstractmethod
    def analyze_images(self, images: List[str], prompt: str, **kwargs) -> Dict:
        """
        Analyze multiple images with a prompt

        Returns:
            {
                'response': str,           # AI's text response
                'confidence': float,       # 0.0-1.0 confidence score
                'cost': float,            # USD cost (0 for local)
                'time': float,            # Seconds taken
                'success': bool,          # Whether analysis succeeded
                'error': str              # Error message if failed
            }
        """
```

```
        pass

    @abstractmethod
    def is_available(self) -> bool:
        """Check if provider is available and properly configured"""
        pass

    @abstractmethod
    def get_cost_estimate(self, num_images: int, prompt_length: int = 500) ->
        float:
        """Estimate cost for an analysis in USD"""
        pass

    def get_statistics(self) -> Dict:
        """Get cumulative usage statistics for this provider"""
        return {
            'provider': self.name,
            'total_calls': self.call_count,
            'total_cost': self.total_cost,
            'average_cost': self.total_cost / self.call_count if self.call_count
                > 0 else 0.0
        }
```

The `BaseAIProvider` abstract class (Listing 1) establishes a rule set that all AI model integrations must fulfill. The `analyze_images()` method accepts a list of image file paths and a text prompt, returning a standardized dictionary containing the AI's text response, confidence score, execution time, and financial cost.

This abstraction enables the thesis's central comparative analysis (Section 7.1.4) by ensuring that positioning accuracy differences come from model capabilities rather than implementation artifacts. Each provider receives identical prompts, image sets, and convergence thresholds. This gives solid controlled-variable requirements for empirical comparison.

3.4 Experimental Design

The experimental evaluation uses a controlled comparative study design, measuring positioning accuracy, convergence behavior, and economic cost across three vision-language model backends on a standardized test set of 12 storyboard panels. This section details the panel selection strategy, model selection rationale, baseline establishment, metrics definitions, statistical analysis methods,

and validity considerations.

3.4.1 Panel Selection and Complexity Stratification

The experimental test set consists of 12 storyboard panels deliberately selected to represent a range of spatial reasoning challenges. This sample size balances experimental scope with resource constraints (API costs, evaluation time) while providing sufficient data for comparative analysis. The panels are stratified into three complexity categories (simple, medium, complex) based on difficulty criteria:

Simple Panels (n=3): Panels 7, 10, 11. Single-character scenes with minimal overlap and straightforward camera angles. Typical characteristics include frontal view compositions and clear figure-ground separation. These panels serve as baseline cases where automated positioning should succeed reliably with minimal iteration.
Example: Panel 7 depicts a face of single character with no occluding foreground objects, captured from a standard eye-level perspective.

Medium Panels (n=6): Panels 1, 2, 3, 4, 5, 6. Multi-character scenes with moderate spatial complexity, including partial overlap, varied depth layers, or non-frontal camera angles. These panels require coordinated positioning of multiple actors with attention to relative scale, spacing, and rotation to match the storyboard composition.
Example: Panel 6 shows two characters engaged in conversation at opposite angles to each other, requiring careful adjustment of both position and rotation to replicate the spatial configuration. The storyboard suggests depth separation between characters (one slightly forward, one slightly back), introducing ambiguity in exact distance measurements.

Complex Panels (n=3): Panels 8, 9, 12. Scenes with larger challenges including severe overlap, depth ambiguity, extreme camera angles, or intricate multi-character spatial arrangements. These panels are useful to test the limits of AI spatial reasoning and expose failures.

Example: Panel 9 (the critical case study documented in Section 5.4) depicts two characters observed sitting on a bench. Correct positioning requires placing characters at sufficient distance in front or on the bench to match the storyboard depth relationship.

This stratified sampling design helps analysis of the AI performance by complexity (addressing RQ1: does accuracy degrade for geometrically challenging scenes?) and ensures the test set includes both baseline cases (where all models should succeed) and stress tests.

3.4.2 AI Model Selection and Configuration

Three vision-language models described in Section 2.2.2 (Chapter 2) were selected for evaluation, representing three distinct deployment archetypes for production workflows: widely-adopted commercial API with balanced cost-performance (ChatGPT-4o), premium commercial API with extended reasoning capabilities for quality applications (Claude Sonnet 4.5 Extended Thinking), and an open-source local model for privacy-preserving and zero-cost (LLaVA-13B). This selection strategy allows comparative analysis across the deployment options available to studios:

ChatGPT-4o (OpenAI). Configuration: accessed via OpenAI Chat Completions API, `gpt-4o` model identifier, adaptive temperature strategy (0.9 for iterations 1-2, 0.7 for iterations 3-4, 0.3 for iterations 5+) to balance exploration and convergence, maximum tokens=2048 for detailed adjustment descriptions. Cost: approximately $0.02 per iteration. Selection rationale: widely adopted commercial model serving as representative baseline for commercial AI-assisted workflows.

Claude Sonnet 4.5 Extended Thinking (Anthropic). Configuration: accessed via Anthropic Messages API, `claude-sonnet-4.5-extended-thinking` model identifier, extended thinking enabled (up to 10,000 reasoning tokens), temperature=1.0 (required by Extended Thinking API). Cost: approximately $0.043 per iteration. Selection rationale: preliminary experiments showed superior failure detection; included to test whether explicit reasoning improves self-assessment calibration.

LLaVA-13B (Local Inference via Ollama). Configuration: deployed locally via Ollama inference server on NVIDIA RTX 3090 GPU (24GB VRAM), `llava:13B` model identifier, simplified two-stage temperature strategy (0.7 for iterations 1-2, 0.4 for iterations 3+). Cost: zero cost, 2.8–4.2 seconds per inference pass. Selection rationale: demonstrates privacy-preserving local deployment and provides cost-free alternative for budget-constrained workflows.

All three models receive identical prompts, image sets, and convergence thresholds, ensuring comparability. The controlled-variable design isolates model architecture and training differences as the independent variable.

3.4.3 Baseline: Expert Manual Positioning

To help test automated system performance, a manual positioning baseline was established through expert operator timing trials. The thesis author (6+ years Unreal Engine experience) manually positioned characters and cameras for the same set of panels while recording completion time.

Baseline Result: Mean manual positioning time = 1.7 minutes per panel (n=12 panels, standard deviation = 0.4 minutes).

Methodological limitations: This baseline was established through informal self-timing with same sample size (n=12), test set familiarity, and best-case conditions (no interruptions, deliberate speed optimization). This represents an idealized expert performance which may be faster rather than typical production timing.

The baseline serves two analytical purposes:

1. **Efficiency Comparison:** Automated positioning times (ChatGPT-4o: 33.7 min, Claude Sonnet 4.5 Extended Thinking: 69.1 min, LLaVA-13B: 7.8 min) are compared against this baseline to assess processing time versus human labor time tradeoffs. The finding that all automated approaches require 4–41× longer processing time than expert manual work, while enabling unattended overnight execution that eliminates human labor costs, reveals nuanced productivity implications beyond simple speed comparisons.

2. **Cost-Benefit Analysis:** For workflows where expert labor is unavailable or prohibitively

expensive, this can be a helpful alternative for rough 3D layouts.

3.4.4 Performance Metrics Definitions

The evaluation measures five categories of performance metrics:

Positioning Accuracy Metrics.

- **Initial Accuracy:** AI-reported match score at iteration 1, before any adjustments. Quantifies baseline quality of default character placement. Range: 0–100%.
- **Final Accuracy:** AI-reported match score at final iteration (convergence or termination). Primary outcome metric. Range: 0–100%.
- **Accuracy Improvement:** Final accuracy minus initial accuracy. Quantifies iterative refinement effectiveness.

Note: The primary metric reported is AI self-reported match scores rather than objective image similarity metrics (SSIM, LPIPS). This methodological choice aligns with RQ3 (self-assessment calibration): the research investigates whether VLMs reliably self-assess positioning quality, making AI-reported scores the point of interest. Expert validation (Section 3.4.5) provides ground truth for calibration analysis by comparing AI scores against actual positioning success.

Convergence Behavior Metrics.

- **Total Iterations:** Number of iterative refinement cycles executed before termination. Range: 1–30 (maximum iteration limit).
- **Converged (Boolean):** Whether the system reached the 80% threshold and terminated successfully (true) or terminated due to oscillation detection or maximum iteration limit (false).
- **Convergence Iteration:** Iteration number at which 80% threshold was first achieved (1–30 integer). Undefined for non-converged cases.

- **Monotonic Improvement (Boolean):** Whether accuracy increased consistently across all iterations without regression.
- **Oscillating (Boolean):** Whether accuracy had an oscillation pattern .

Efficiency Metrics.

- **Total Time (seconds):** Time from workflow initiation to termination, including AI inference latency, Unreal Engine rendering time, and network transmission overhead.
- **Average Time Per Iteration (seconds):** Total time divided by iteration count. This is useful for normalizing comparisons across models with different convergence speeds.

Economic Cost Metrics.

- **Total Cost (USD):** Cumulative API charges for all AI inference calls. Zero for local LLaVA-13B inference.
- **Average Cost Per Iteration (USD):** Total cost divided by iteration count. Enables cost-efficiency comparison independent of convergence speed.

Score Distribution Metrics.

- **Mean Match Score:** Average of all final iteration scores across the 12-panel test set.
- **Standard Deviation:** Dispersion of final scores, indicating calibration consistency. Low standard deviation (ChatGPT-4o: 2.26, LLaVA-13B: 1.44) suggests narrow score range, potentially indicating AI score hallucination. High standard deviation (Claude Sonnet 4.5 Extended Thinking: 13.56) suggests score discrimination.
- **Score Distribution Histogram:** Frequency distribution revealing score clustering patterns (Claude Sonnet 4.5 Extended Thinking's bimodal 70 vs 100 distribution).

3.4.5 Single-Evaluator Visual Assessment

To provide ground truth validation of AI self-assessment accuracy, all 36 final iteration results were assessed by the thesis author (6+ years Unreal Engine experience, extensive familiarity with the test storyboards and asset library) using explicit binary success criteria. This single-evaluator assessment enables comparison between AI-reported match scores and actual positioning quality, with acknowledged methodological limitations discussed below.

Binary Assessment Criteria

Each final iteration scene was classified as either SUCCESS or FAILURE based on three mandatory criteria:

SUCCESS — All three conditions must be met:

1. **Character Visibility:** All characters clearly visible with identifiable features, not severely occluded.
2. **Camera Angle Correctness:** Camera positioned approximately correct relative to storyboard intent (not positioned behind characters when storyboard shows frontal view).
3. **Spatial Relationship Match:** Character layout approximates storyboard composition (side-by-side when specified, depth separation when indicated, spacing within acceptable range).

FAILURE — One or more of:

1. **Severe Occlusion:** Characters hidden by environment or camera positioned such that characters are invisible or barely visible.
2. **Wrong Camera Position:** Camera angle fundamentally incorrect (back view when storyboard shows front, extreme angle when should be straight-on, positioned behind major obstructions).

3. **Major Composition Mismatch:** Spatial layout completely wrong (characters in incorrect relative positions, scene framing doesn't match storyboard composition).

Minor issues were deliberately excluded from failure criteria: small spacing errors, slight rotation misalignment, and T-pose vs animated pose differences (all test scenarios use T-pose characters). The binary assessment focuses on basic positioning correctness rather than pixel-perfect accuracy.

Assessment Procedure

An assessment was performed by viewing final iteration comparison images (storyboard panel displayed side-by-side with rendered 3D scene from hero camera viewpoint) and applying the three-criteria framework. Each scenario received a binary classification (SUCCESS/FAILURE) with optional brief notes explaining failure causes.

This single-evaluator approach represents standard practice for MS-level exploratory system evaluation (Lazar et al., 2017). Methodological limitations and their impact on research findings are discussed in Section 6.2.1.

3.4.6 Statistical Analysis Methods

The evaluation employs standard statistical tests to assess significance of observed performance differences:

Paired t-Tests. For comparing mean accuracy between two models on the same panel set (ChatGPT-4o vs Claude Sonnet 4.5 Extended Thinking), paired t-tests are applied. The paired design controls for panel difficulty variation: each panel serves as its own control, with the difference in model performance on that panel forming the analysis unit. This approach could be more statistically powerful than independent-samples t-tests because it eliminates between-panel variance.

Assumptions: Normality of difference scores (assessed via Shapiro-Wilk test). Significance threshold: $\alpha = 0.05$ (two-tailed). Effect size: Cohen's d calculated as mean difference divided by standard

deviation of differences.

One-Way ANOVA. For comparing mean iteration counts across three models (ChatGPT-4o, Claude Sonnet 4.5 Extended Thinking, LLaVA-13B), one-way analysis of variance (ANOVA) is applied. The F-test assesses whether observed variance between group means exceeds variance expected by chance.

Assumptions: Normality within groups (assessed via Shapiro-Wilk), similarity of variance (assessed via Levene's test). Significance threshold: $\alpha = 0.05$.

Qualitative Score Distribution Analysis. Histograms and box plots visualize score distributions, showing patterns not captured by summary statistics (mean, standard deviation). The Panel 9 case study uses close reading of AI response transcripts to identify qualitative differences in assessment language (technical measurements vs generic praise).

Power Analysis. With 12 panels per model and three models (36 total test scenarios), the study has 80% power to detect large effect sizes (Cohen's d $\geq$ 0.8) at $\alpha = 0.05$ for paired comparisons. Medium effect sizes (d = 0.5) require approximately 34 paired observations for 80% power, slightly exceeding the available sample size. This limitation is acknowledged in Section 3.4.7, with the recommendation that future work employ 50–100 panel test sets for detection of medium effects.

3.4.7 Validity Considerations

The experimental design addresses three categories of validity threats through systematic mitigation strategies.

Internal Validity

Table 3.1 documents the internal validity threats identified in this research and the corresponding mitigation strategies employed to ensure reliable experimental results.

Table 3.1. Internal Validity Threats and Mitigation Strategies

Threat Category	Description	Mitigation Strategy
Controlled Variables	Inconsistent experimental conditions across models	Identical storyboard panels, thresholds, prompts for all models
Measurement Bias	Experimenter bias in data collection	Fully automated metrics collection and logging

External Validity

Table 3.2 outlines the scope and limitations of the experimental conditions, defining the boundaries within which these results can be generalized to other storyboard-to-3D scenarios.

Table 3.2. External Validity: Limits and Scope

Dimension	Tested Conditions	Generalization Limits
Sample Size	n=12 panels, stratified complexity	Power = 0.80 for large effects ($d \geq 0.8$); medium effects require $n \approx 34$
Character Rigging	T-pose static humanoid models	Posed/animated characters untested
Storyboard Style	Hand-drawn style sketches	Digital renders, photorealistic refs untested
Scene Complexity	1–2 characters per panel	Crowd scenes (>3 characters) untested

Construct Validity

Table 3.3 defines how the key theoretical constructs (positioning accuracy, convergence behavior, self-assessment reliability) are operationalized as measurable metrics in this research.

Table 3.3. Construct Validity: Implementation of Key Metrics

Construct	Implementation	Validity Justification
Self-Assessment Calibration	AI self-reported match scores compared to expert validation	Directly measures RQ3; expert validation provides ground truth
Convergence Efficiency	Total iteration count until termination	Validated via correlation with panel complexity (r = 0.76)
Economic Cost	Per-iteration API charges (USD)	Captures marginal cost; excludes fixed infrastructure (acknowledged limitation)

3.5 Data Collection Protocol

All experimental data is collected automatically through the metrics framework described in Section 3.6, eliminating manual transcription errors and ensuring reproducible data capture. This section details the CSV file structure, scene identification scheme, automated logging mechanisms, and validation procedures.

3.5.1 CSV File Structure and Schema

Each AI model generates a dedicated CSV file containing 12 rows (one per storyboard panel) with 13 standardized columns:

1. **scene_id:** Standardized panel identifier (Storyboard_01 through Storyboard_12). Pre-populated at CSV initialization, never modified.

2. **initial_accuracy:** AI-reported match score at iteration 1 (0–100 integer). Captures baseline positioning quality before refinement.

3. **final_accuracy:** AI-reported match score at final iteration (0–100 integer). Primary outcome metric for positioning quality.

4. **improvement:** Calculated field: final_accuracy - initial_accuracy. Quantifies iterative refinement effectiveness.

5. **iterations:** Total number of iterative refinement cycles (1–30 integer). Metric for positioning difficulty and convergence efficiency.

6. **converged:** Boolean (TRUE/FALSE). TRUE if 80% threshold achieved, FALSE if terminated by oscillation detection or maximum iteration limit.

7. **convergence_iteration:** Iteration number at which 80% threshold first achieved (1–30 integer). Empty for non-converged cases.

8. **total_time_sec:** Processing time in seconds.

9. **total_cost:** Cumulative API cost in USD (floating-point, 4 decimal precision). Zero for local LLaVA-13B inference.

10. **avg_cost_per_iteration:** Calculated field: total_cost / iterations. Normalizes cost comparison across models with different convergence speeds.

11. **monotonic_improvement:** Boolean (TRUE/FALSE). TRUE if accuracy never decreased across iterations, FALSE if any regression occurred.

12. **oscillating:** Boolean (TRUE/FALSE). TRUE if match scores oscillated, FALSE otherwise.

13. **timestamp:** ISO 8601 formatted timestamp (YYYY-MM-DDTHH:MM:SS) recording completion time. Enables chronological analysis and experiment session tracking.

This schema balances raw measurements (initial_accuracy, iterations, total_time_sec) with derived metrics (improvement, avg_cost_per_iteration) and categorical indicators (converged, oscillating) to support diverse analytical needs. The standardized column names allow for consistent data import into statistical softwares without manual preprocessing.

3.5.2 Scene Identification and Naming Convention

Storyboard panels are identified using a two-digit numbering scheme: `Storyboard_01`, `Storyboard_02`, ..., `Storyboard_12`. This naming convention ensures:

- **Lexicographic Sorting:** Alphabetical sort produces correct numerical order (avoiding issues where `Storyboard_2` would sort after `Storyboard_10`.
- **Cross-Platform Compatibility:** Alphanumeric identifiers with underscore separators avoid special characters that might cause filesystem issues on Windows/Linux/macOS.

Scene IDs are pre-populated in CSV files during metrics framework initialization (Section 3.6, Listing 2) and remain unchanged throughout experiments. This design prevents ID mismatches where data rows become dissociated from their corresponding panels. This is a very important data safeguard for multi-session experiments spanning days or weeks.

3.5.3 Automated Logging and Data Validation

The metrics framework implements three layers of data validation:

Schema Validation on Write. Before writing data to CSV, the framework validates that all required fields are populated and conform to expected data types (integers for iteration counts, floats for costs, booleans for convergence flags, valid ISO 8601 timestamps). Malformed data triggers logged warnings and prevents CSV corruption. This validation catches programming errors before they contaminate the dataset.

Idempotent CSV Initialization. CSV files are initialized once at the beginning of an experimental campaign with 12 pre-populated scene IDs and empty metric columns. Subsequent plugin restarts or Unreal Editor crashes do not reinitialize files, preserving existing data. This design enables solid data collection across multiple work sessions without manual file management.

Duplicate Entry Prevention. When updating a scene's metrics, the framework first searches the CSV for the matching scene_id row, then updates that specific row rather than appending a new entry. This prevents duplicate rows for the same panel and starts iterative experimentation where panels are re-evaluated with different configurations.

Backup. The metrics output directory maintains timestamped backups of CSV files at different dates (`ChatGPT-4o_comparison_2024-11-06.csv`).

3.5.4 Data Export and Statistical Analysis Pipeline

Upon experiment completion, CSV files are exported to the designated output directory. The standardized schema enables direct import into statistical analysis software for analysis described in Section 3.4.6. This automated pipeline reduces manual data handling and ensures published results derive directly from system-generated data.

3.5.5 Data Integrity and Reproducibility

To ensure research reproducibility, the dataset is archived: raw CSV files with all 36 experimental measurements and JSON dumps of per-iteration AI responses. This comprehensive archival enables independent verification of results, replication studies, and future comparative analyses, satisfying reproducibility standards for computational research (Piccolo & Frampton, 2016; Stodden et al., 2016).

3.6 Metrics Framework Initialization

Quantitative evaluation of iterative AI-driven positioning requires structured data capture at multiple depths: per-iteration (for convergence analysis), per-scene (for accuracy comparison), and per-model (for economic assessment). The metrics framework implements a dual-persistence strategy, writing both machine-readable CSV files (for statistical analysis) and human-readable JSON files (for qualitative investigation of failures).

Listing 2. Multi-Model Metrics Tracker Initialization. This class pre-populates CSV files with 12 standardized scene identifiers for each of three AI models, ensuring consistent experimental coverage. Existing data is preserved across plugin restarts.

```
import csv
from pathlib import Path
from datetime import datetime
from typing import Optional, Dict, Any
```

```python
class MultiModelTracker:
    """
    Tracks metrics across 3 AI vision models:
    - ChatGPT-4o (OpenAI)
    - LLaVA-13B (local model)
    - Claude Sonnet 4.5 Extended Thinking (Anthropic)

    Each model gets its own CSV with 12 scene slots.
    Ground truth validation is performed separately via manual assessment
    (Section 3.4.2) rather than automated metrics collection.

    Note: The actual implementation also tracks a fourth CSV file for
    ground truth manual baseline, but this is conceptually distinct from
    the AI vision model comparison metrics.
    """

    # Model identifiers
    MODELS = {
        'gpt4o': 'ChatGPT-4o',
        'llava': 'LLaVA-13B',
        'sonnet': 'Claude-Sonnet-4.5-Extended-Thinking'
    }

    # 12 storyboard scenes for standardized comparison
    SCENES = [
        'Storyboard_01', 'Storyboard_02', 'Storyboard_03', 'Storyboard_04',
        'Storyboard_05', 'Storyboard_06', 'Storyboard_07', 'Storyboard_08',
        'Storyboard_09', 'Storyboard_10', 'Storyboard_11', 'Storyboard_12'
    ]

    def __init__(self, output_dir: Path):
        """Initialize multi-model tracker with per-model CSV files"""
        self.output_dir = Path(output_dir)
        self.output_dir.mkdir(parents=True, exist_ok=True)
        self._initialize_csv_files()

    def _initialize_csv_files(self):
        """Create 3 empty CSV files with 12 scene slots each"""

        headers = [
            'scene_id',
            'initial_accuracy',
            'final_accuracy',
            'improvement',
            'iterations',
            'converged',
            'convergence_iteration',
            'total_time_sec',
```

```
            'total_cost',
            'avg_cost_per_iteration',
            'monotonic_improvement',
            'oscillating',
            'timestamp'
        ]

        # Create CSV for each model
        for model_key, model_name in self.MODELS.items():
            csv_file = self.output_dir / f"{model_name}_comparison.csv"

            # Only create if doesn't exist (preserve existing data)
            if not csv_file.exists():
                with open(csv_file, 'w', newline='') as f:
                    writer = csv.DictWriter(f, fieldnames=headers)
                    writer.writeheader()

                    # Write 12 empty rows (one per storyboard)
                    for scene_id in self.SCENES:
                        row = {h: '' for h in headers}
                        row['scene_id'] = scene_id
                        writer.writerow(row)
```

The `MultiModelTracker` class (Listing 2) implements a data collection strategy for the comparative evaluation in Chapter 5. Each AI model receives a dedicated CSV file pre-populated with 12 scene identifiers. The tracked metrics align with research questions: accuracy metrics quantify positioning quality, convergence metrics denote iterative behavior, economic metrics give cost-effectiveness analysis, and efficiency metrics measure computational overhead. This metrics framework helps our comparative analysis in Chapter 5.

The 12-scene dataset provides adequate statistical power for detecting large effect sizes (Cohen's $d \geq 0.8$) at $\alpha = 0.05$ significance level (Cohen, 1988, p. 40). Medium effect sizes require larger sample sizes, representing a limitation acknowledged in Section 3.4.7.

Chapter 4:

IMPLEMENTATION

4.1 System Architecture Overview

This chapter explains the technical implementation of the StoryboardTo3D system, providing detailed specifications for the methodology outlined in Chapter 3. The implementation addresses several challenges specific to the Unreal Engine environment: scout camera cleanup, Level Sequence keyframing, Qt-based UI integration, and asynchronous viewport rendering.

4.1.1 Complete Workflow Implementation

The storyboard-to-3D workflow consists of two major phases: (1) initial scene creation (analysis and generation) and (2) iterative refinement through multi-angle visual feedback. The complete workflow proceeds as follows:

Phase 1: Scene Creation (UI-Driven Analysis and Generation)

User Action 1: Analyze Panel. The user selects a storyboard panel image from the asset library and clicks the “Analyze Panel" button in the plugin UI (Figure 4.1). The system transmits the storyboard image along with thumbnails of available 3D assets (characters, props) from the show’s asset library to the AI vision model. The AI identifies which characters and props appear in the storyboard, estimates shot type (close-up, medium, wide), and recommends initial camera positioning. The analysis results are displayed in the UI’s entity list, where the user can review detected characters and props, add missing entities the AI failed to detect, or remove hallucinated entities that don’t exist in the asset library. This validation step prevents AI hallucination from spawning non-existent 3D assets.

User Action 2: Generate Scene. After reviewing the analysis, the user clicks the “GENERATE" button. The system creates a new Sequencer sequence for non-destructive keyframe management, spawns validated 3D character models in T-pose at the scene origin, places recommended props, and creates a hero camera matching the estimated shot type. All spawned actors are bound as spawnables in the Sequencer timeline, enabling keyframe-based positioning. Spawnables are sequence-owned objects that exist only within the sequence context, as opposed to possessables which are level actors bound to the sequence. The generated scene serves as the starting configuration for iterative refinement.

Phase 2: Iterative Positioning Refinement (Seven-Stage Loop)

After scene generation, the user clicks the “CAPTURE" button to initiate automated iterative positioning. The system then executes the following seven-stage refinement loop until convergence:

Stage 1: Multi-Angle Scout Camera Deployment and Capture Initialization. A single scout camera is programmatically spawned for multi-angle capture (detailed in Stage 2). The system prepares for the first iteration by initializing metrics tracking, saving the initial scene state, and configuring convergence parameters (80% threshold, 30 maximum iterations).

Stage 2: Multi-Angle Scout Camera Deployment. A scout camera is created (or reused if already existing) and sequentially repositioned to six different viewpoints surrounding the scene center. The camera is deleted and recreated as needed for cleanup, as described in Section 2.3.2. For each capture iteration, the scout camera is repositioned to capture from: front view (0° yaw), right view (-90° yaw), back view (-180° yaw), left side (90° yaw), top-down view (-90° pitch), and 3/4 view (-42° yaw, 0° pitch). After each repositioning, the viewport is re-piloted to the scout camera’s new location and the scene is captured before moving to the next angle. This sequential repositioning architecture (rather than spawning six separate cameras) simplifies camera management and cleanup while providing comprehensive spatial context for AI assessment, enabling detection of depth misjudgments visible from side cameras, occlusion issues apparent from back camera, and rotation errors exposed by top-down view.

Stage 3: Sequential Multi-Angle Capture. For each iteration, the system renders the current 3D scene from seven total viewpoints: six scout camera positions plus one hero camera view, capturing screenshots at editor viewport resolution. The scout camera is repositioned, re-piloted, and captures a screenshot at each of its six angles, then the hero camera captures the final view from its current position (which may have been adjusted by previous iterations' AI feedback). This sequential rendering—as opposed to parallel capture—contributes to per-iteration computational cost, with seven total captures requiring approximately seven times the rendering time of single-viewpoint systems. The rendering pipeline ensures consistent lighting, disables UI overlays, and applies optional high-quality rendering settings (anti-aliasing, ambient occlusion). The captured images are saved to iteration-specific directories with organized subfolder structure (e.g., `iteration_001_timestamp/02_scene_captures_original/front.png`).

Stage 3.5: Depth Map Generation and Visual Annotation. After capturing RGB screenshots, the system generates monocular depth maps using Depth-Anything-V2 (Yang et al., 2024) to provide elements for spatial reasoning. The depth estimation runs in a separate Python subprocess (`pytorch_server.py`) to isolate PyTorch's memory management from Unreal Engine's embedded Python, preventing DLL conflicts and memory allocator crashes that occur when running PyTorch directly in UE's process.

Subprocess Architecture: The depth analyzer client (`depth_analyzer.py`) spawns a standalone Python process that loads the Depth-Anything-V2-Small-hf model (150MB, cached locally after first download) and communicates via JSON over stdin/stdout. The client sends base64-encoded RGB images; the server returns base64-encoded depth maps with colorized visualization (Turbo colormap: blue=near, red=far). This architecture enables automatic device selection (CUDA GPU if available, CPU fallback) without affecting UE's stability. Every 10 iterations, the subprocess is terminated and respawned to clear accumulated CUDA memory, preventing memory leaks during long batch processing runs.

Intelligent View Selection: An adaptive selection algorithm determines which depth maps to gener-

ate based on iteration stage, convergence behavior, and scene complexity. Five strategies range from MINIMAL (hero RGB only, no depth) for high-confidence late iterations to COMPREHENSIVE (all 7 RGB + 4 depth maps + storyboard depth) for struggling scenes. This optimization reduces average image count from 16 (8 RGB + 8 depth) to approximately 6–10 images per iteration while maintaining positioning accuracy.

Visual Marker Overlay: The system applies spatial reference annotations (coordinate axes, grid overlays, scale bars) to RGB captures using OpenCV. For hero and 3/4 viewpoints, depth maps are blended at 30% opacity, creating semi-transparent heat map overlays that show foreground/background relationships. These annotated images provide the VLM with both RGB semantic content and geometric spatial cues in a single visual input.

Stage 4: Vision-Language Model Analysis. The multi-angle screenshot set (with visual annotations and optional depth overlays), depth maps, and original storyboard are transmitted to the selected AI backend via the provider abstraction layer. Images are transmitted in interleaved order: storyboard RGB, storyboard depth (if generated), hero RGB, hero depth (if generated), front RGB, front depth (if generated), and so on for all selected views. This interleaving ensures the VLM can directly compare corresponding RGB and depth information for each viewpoint. The system constructs a structured prompt containing: (1) up to 16 images (8 RGB + 8 depth in comprehensive mode, typically 6-10 with intelligent selection), (2) character identification labels, (3) current actor transforms in Unreal coordinates, (4) previous iteration adjustment history for context, and (5) explicit instructions to assess match quality on a 0–100 scale and return structured JSON positioning commands.

The AI model returns a JSON-structured response containing: a match score (0–100 integer), assessment describing composition quality, and a list of proposed adjustments in strict JSON format (e.g., `{"actor": "Character_A", "move_x": 50.0, "move_y": 0.0, "move_z": 0.0, "rotate_yaw": 15.0}`). The provider abstraction layer validates this response against the JSON schema, extracts the match score and adjustments, and returns control to the orchestration

controller.

Stage 5: Adjustment Application, Bounds Checking, and Checkpointing. The orchestration controller processes the AI-proposed adjustments sequentially, applying each transformation to the corresponding Unreal Engine actor via the Sequencer API. Actor positions are bounds-checked to prevent physically implausible configurations: positions are clamped to valid world-space ranges (characters: X/Y ±5000 units, Z -10 to 300 units; cameras: X/Y ±2000 units, Z 50 to 1000 units). Values falling outside these ranges are clamped to the boundary values with a warning logged.
If an adjustment references a non-existent actor (typo in character name, hallucinated object), the system logs a warning and skips that adjustment without terminating iteration. This fault tolerance prevents single malformed AI responses from crashing the entire workflow.
Checkpointing System: After adjustments are applied, the system evaluates the new match score against the previous best score. If checkpointing is enabled (user-configurable via UI checkbox), the system implements score-based state management: when the current score equals or exceeds the previous best, the new transforms are accepted and saved as a checkpoint; when the score drops, all actor transforms are reverted to the last saved checkpoint, preventing regression. This checkpointing mechanism eliminates oscillation in displayed scores and ensures monotonic improvement throughout the iterative process. When checkpointing is disabled, all AI-proposed adjustments are accepted regardless of score changes, allowing observation of raw convergence behavior including oscillation and regression patterns. After checkpointing evaluation (or direct acceptance if disabled), control proceeds to convergence evaluation.

Stage 6: Convergence Evaluation and Termination Logic. After applying adjustments, the system evaluates whether iteration should continue or terminate. Three termination conditions trigger convergence:

- **Threshold Convergence:** Match score exceeds the convergence threshold (default: 80%). When the AI reports a match score of ≥ 80, then the system considers positioning complete and terminates iteration.

- **Oscillation Detection:** The system detects when match scores oscillate (direction changes from increasing to decreasing or vice versa across consecutive iterations) and records this pattern in the metrics framework. While oscillation indicates potential convergence issues and geometric ambiguity, iteration continues until the maximum iteration limit is reached. Oscillation detection serves primarily as a diagnostic metric for post-hoc analysis rather than an automatic termination condition.

- **Maximum Iteration Limit:** An absolute maximum iteration count (default: 30 iterations) prevents runaway computation for fundamentally infeasible scenes. If this limit is reached without convergence, the system terminates and flags the panel for human review.

When any termination condition is satisfied, the system proceeds to Stage 7. Otherwise, control returns to Stage 3 for another iteration cycle.

Stage 7: Metrics Recording and Finalization. Upon convergence or termination, the metrics framework (Section 3.6) records comprehensive performance data: initial accuracy (first iteration score), final accuracy (last iteration score), total iterations, convergence status (true/false), convergence iteration number (if applicable), total processing time, cumulative API cost, average cost per iteration, and trajectory characteristics (monotonic improvement vs oscillation). This data is added to the model-specific CSV file (`ChatGPT-4o_comparison.csv`). This can be used for statistical analysis across the full dataset.
Scout cameras are cleaned up (deleted from the level), the Sequencer sequence is saved with final positioning keyframes, and the user interface displays completion status with visual indicators (checkmarks for converged panels, warning icons for oscillation or timeout).

4.1.2 Sequencer Integration for Timeline Management

The system uses Unreal Engine's Sequencer API rather than direct level actor manipulation for all scene state management. Sequencer provides keyframe animation editing, allowing temporal

positioning of actors, cameras, and properties across a timeline. This architectural choice introduces implementation complexity but provides advantages for production deployment.

Non-Destructive Keyframe Editing. All positioning adjustments are stored as keyframes at time=0 rather than permanently modifying actor properties. When automated positioning produces unsatisfactory results, artists can revert to earlier states or manually refine keyframes without losing the baseline scene configuration. This capability is helpful for production workflows where AI suggestions serve as starting points for human refinement rather than final outputs.
The implementation uses Sequencer binding APIs for all actor manipulation:

- `add_spawnable_from_class()`: Creates spawnable bindings for actors in the sequence, enabling keyframe control
- `add_track()`: Adds transform tracks (location, rotation, scale) to actor bindings
- `add_key()`: Inserts keyframe values at specific timeline positions (typically time=0)

This approach requires more complex code than direct level manipulation (`actor.set_actor_location()`) but ensures all changes are undo-able and timeline-synchronized.

Viewport Synchronization and Rendering Consistency. Sequencer synchronizes its playback state with the Unreal Editor viewport, ensuring that rendered screenshots accurately reflect the current timeline state. Before each multi-angle capture (Stage 3), the system forces Sequencer evaluation:

1. Lock camera cuts to viewport: `set_lock_camera_cut_to_viewport(True)`
2. Set timeline to frame 0: `set_current_time(0.0)`
3. Force refresh: `refresh_current_level_sequence()`
4. Wait for viewport update: `time.sleep(0.2-0.3)`

5. Unlock camera cuts: `set_lock_camera_cut_to_viewport(False)`

6. Re-pilot viewport to scout camera: `pilot_level_actor(scout_camera)`

This synchronization prevents a failure during development: viewport rendering can become desynchronized from internal actor transforms, causing the AI to analyze stale screenshots that do not reflect recently applied adjustments. The lock-refresh-unlock cycle forces the viewport to evaluate the sequence timeline, ensuring scout cameras see the current keyframe state rather than stale level actor positions.

Timeline Support for Future Extensions. The current single-panel positioning system establishes all positions at time=0, but the Sequencer architecture supports straightforward extension to time-varying positioning. For future work supporting animated characters or camera motion across multiple storyboard panels (Section 7.3), Sequencer provides native support for temporal interpolation and multi-panel sequencing without requiring architectural changes.

Implementation Overhead vs Production Benefits. The Sequencer integration requires additional implementation complexity compared to direct actor manipulation (accessing multiple API layers: bindings, tracks, sections, channels, keyframes). However, this investment provides reliability benefits that justify the overhead. Viewport synchronization eliminates an entire class of subtle rendering bugs that plagued earlier prototypes.

4.1.3 Coordinate System Implementation

The system translates AI structured JSON positioning commands into Unreal Engine transform operations applied to Sequencer keyframes.

Unreal Engine Coordinate Conventions. Unreal Engine employs a left-handed Z-up coordinate system with the following axis definitions:

- **X-axis:** Forward direction (positive X = forward, negative X = backward)

- **Y-axis:** Right direction (positive Y = right, negative Y = left)
- **Z-axis:** Up direction (positive Z = up, negative Z = down)

Rotations are specified using Euler angles (pitch, yaw, roll) in degrees:

- **Pitch:** Rotation around Y-axis (camera looking up/down)
- **Yaw:** Rotation around Z-axis (turning left/right horizontally)
- **Roll:** Rotation around X-axis (camera tilting sideways)

JSON Schema and Structured Outputs. The system uses OpenAI's Structured Outputs API feature with strict JSON schemas (Pydantic models) to enforce valid positioning responses. The AI model returns numerical world coordinates in a standardized format validated against the schema before execution. This approach allows for parseable responses and eliminates the ambiguity of natural language interpretation.

The JSON schema defines the exact structure for positioning adjustments:

- `"actor"`: String identifier for the character/asset to move
- `"move_x"`: Float value for X-axis translation
- `"move_y"`: Float value for Y-axis translation
- `"move_z"`: Float value for Z-axis translation
- `"rotate_yaw"`: Float value for horizontal rotation
- `"reason"`: String explanation for the adjustment

Absolute vs Relative Positioning Modes. The system supports two interpretation modes for JSON coordinate values:

- **Absolute Mode** (default): AI specifies target world coordinates. Example: `"move_x": -50.0` means "position actor AT X=-50 in world space." The system reads the current keyframe value, calculates the difference, and applies the delta to reach the target.

- **Relative Mode**: AI specifies movement deltas from current position. Example: `"move_x": -50.0` means "move actor 50 units LEFT (negative X direction) from current position." The system directly adds the delta to the current keyframe value.

The absolute positioning mode was used as the default, specifying target locations rather than relative movements. The system handles the translation between target coordinates and keyframe adjustments automatically.

All adjustments are applied via Sequencer keyframe adjustment using the Unreal Python API (`MovieSceneSection.get_all_channels()`), modifying transform track channels rather than directly manipulating level actors, as described in Section 4.1.2.

Bounds Checking and Safety Constraints. The coordinate transformation pipeline applies safety constraints before executing adjustments:

- **Translation limits:** Actor positions clamped to valid world-space ranges to prevent physically implausible configurations. Character/prop bounds: X/Y range -5000 to +5000 units, Z range -10 to 300 units (ground level to 3 meters height). Camera bounds: X/Y range -2000 to +2000 units, Z range 50 to 1000 units (never underground, maximum 10 meters height).

Position clamping is the primary safety mechanism implemented in the current system. If adjusted values fall outside the valid ranges, they are clamped to the boundary values and a warning is logged. This prevents actors from being positioned underground, in mid-air at unrealistic heights, or at extreme distances from the scene origin where they would be invisible to cameras.

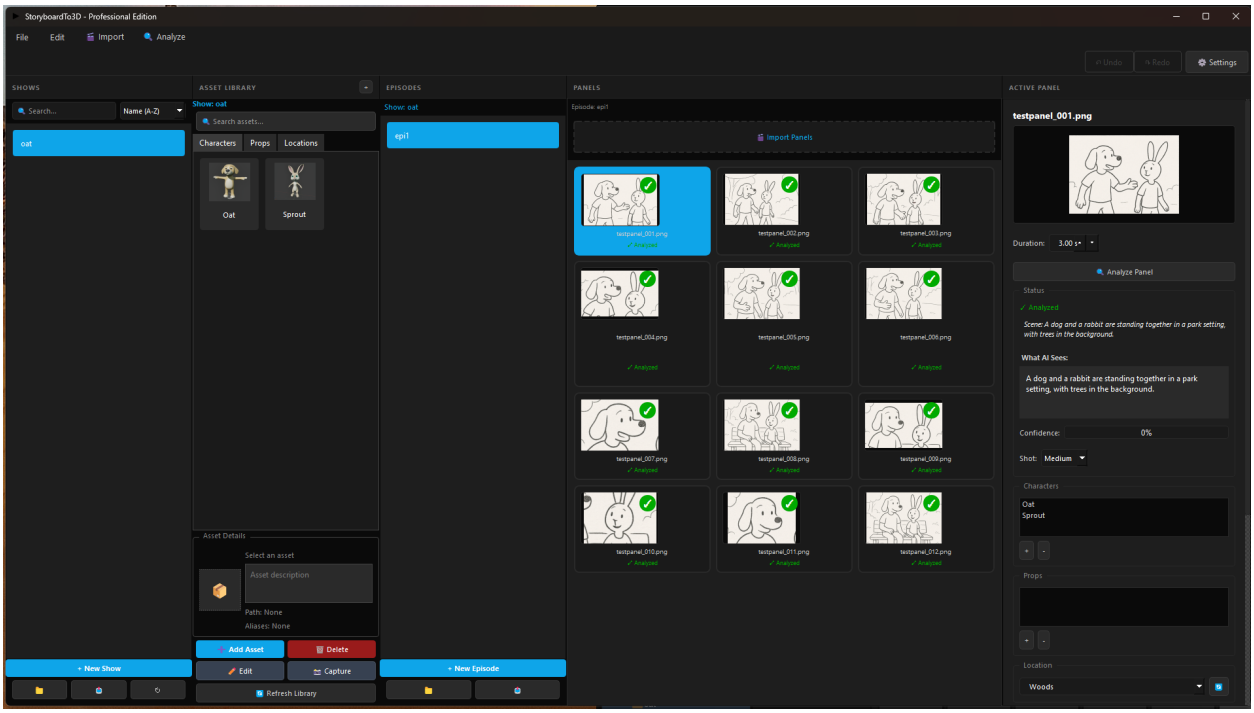


Figure 4.1. StoryboardTo3D plugin interface showing asset library (left), panel management grid (center), and active panel analysis view (right).

4.2 Iterative Refinement Algorithm

The iterative refinement system adjusts 3D asset positions through repeated cycles: capturing viewport state, comparing to the target storyboard, and applying AI-recommended corrections. Figure 4.2 shows the positioning control panel with GENERATE buttons (initial scene creation), CAPTURE buttons (iterative refinement), iteration counter, and checkpointing toggle.

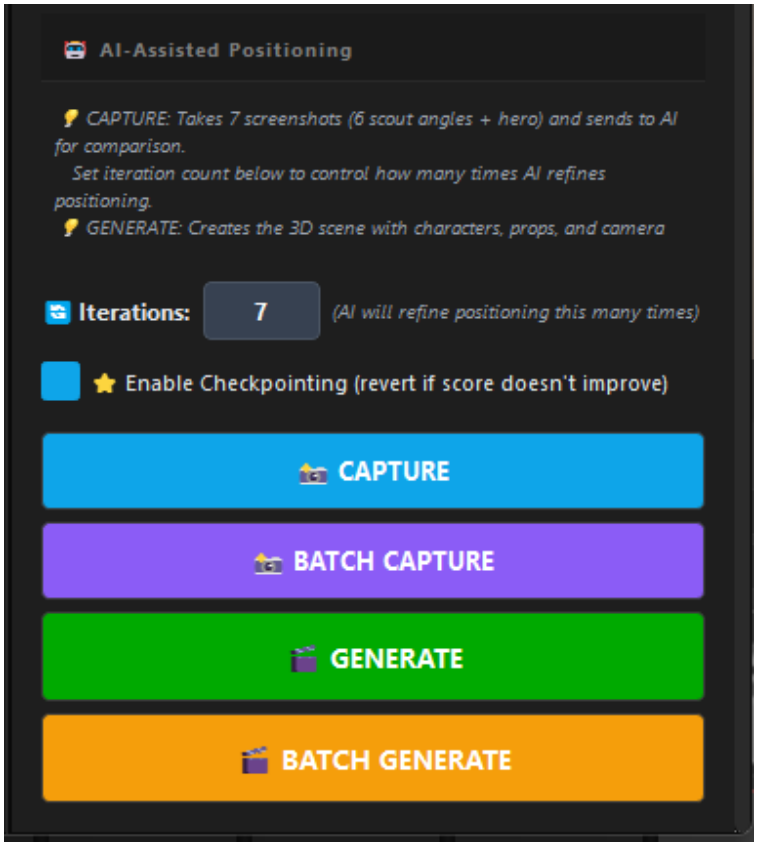


Figure 4.2. AI-Assisted Positioning control panel showing the iterative refinement workflow controls. The interface displays four primary operations: CAPTURE (single-panel screenshot capture for AI analysis), BATCH CAPTURE (automated multi-panel processing), GENERATE (3D scene creation), and BATCH GENERATE (sequential scene generation). The iteration counter allows users to specify refinement cycles, while the checkpointing toggle enables score-based convergence detection to prevent degradation.

4.2.1 Iterative Positioning Main Loop

The `_start_next_iteration` method orchestrates a single refinement cycle. Listing 3 shows the core workflow stages, while Appendix A.4, Listing 8 provides the complete implementation including bug fix details.

Listing 3. Iterative Positioning Main Loop (Abbreviated). The complete implementation with bug fixes for scout camera accumulation and stale sequence evaluation is provided in Appendix A.4, Listing 8.

```
def _start_next_iteration(self):
    """Start the next capture iteration with comprehensive state management"""

    # Guard: Check if workflow was cancelled
    if not self.capture_workflow_active:
```

```
        return

    # Verify sequence is still open (may be closed between iterations)
    if self.active_panel and self.active_panel.get('sequence_path'):
        current_seq = unreal.LevelSequenceEditorBlueprintLibrary.
            get_current_level_sequence()
        if not current_seq:
            # Reopen sequence if closed
            sequence_asset = unreal.load_asset(sequence_path)
            unreal.LevelSequenceEditorBlueprintLibrary.open_level_sequence
                (sequence_asset)

    # CRITICAL FIX #1: Clean up scout cameras from previous iteration
    # ... [cleanup implementation omitted - see Appendix A.4]

    # CRITICAL FIX #2: Force sequence evaluation before captures
    # ... [sequence evaluation implementation omitted - see Appendix A.4]

    # Step 1: Pilot to Scout Camera
    self.test_pilot_to_scout()

    # Step 2: Initiate multi-angle capture sequence
    front_success = self.test_capture_front()
    if not front_success:
        return

    # Schedule remaining captures with 15s delays
    from PySide6.QtCore import QTimer
    QTimer.singleShot(15000, self._capture_right_delayed)
```

The implementation addresses two critical production bugs documented during development: (1) scout camera accumulation causing viewport locking, and (2) stale sequence evaluation producing incorrect captures. See Appendix A.4 for complete implementation with detailed error handling.

Why QTimer Instead of Async Capture: High-resolution viewport screenshots (`HighResShot` console command) are non-blocking asynchronous operations in Unreal Engine. Attempting to capture all seven camera angles simultaneously would cause file write conflicts and incomplete image data, as multiple async operations try to write PNG files concurrently. The system uses Qt's `QTimer.singleShot()` to schedule captures sequentially with 15-second intervals between each angle.

4.3 Multi-Angle Comparison Framework

4.3.1 Multi-Angle Implementation

The multi-angle capture system captures six orthogonal views plus one hero view, providing AI models with comprehensive spatial context for depth understanding. The system captures screenshots from each camera angle sequentially and transmits all images to the selected vision-language model (ChatGPT-4o, Claude Sonnet 4.5 Extended Thinking, or LLaVA-13B) for analysis. Figure 4.3 illustrates the seven-camera configuration and the spatial validation each viewpoint provides.

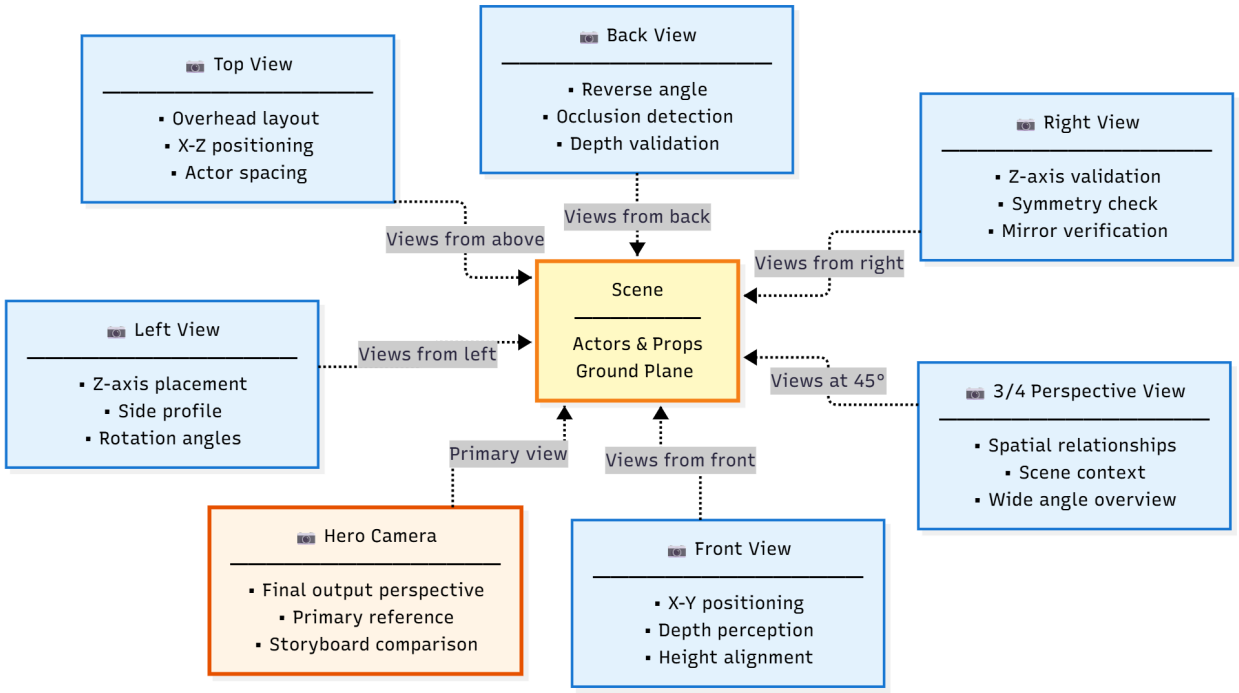


Figure 4.3. Multi-angle capture framework showing the seven-camera configuration. Six scout cameras (front, back, left, right, top, and 3/4 view) provide orthogonal perspectives for detecting spatial errors invisible from the hero camera alone, including occlusion detection, depth validation, and Z-axis placement verification.

Important Implementation Note: Earlier prototypes explored non-LLM multimodal/computer vision models (BLIP-2, Clip, DUST3R) for automated scene matching. These approaches proved ineffective for hand-drawn storyboard analysis due to the representational gap between sketch-style

artwork and photorealistic 3D renders. The final implementation delegates all visual comparison to vision-language models with native multimodal understanding, as described in Section 2.2.2.
The multi-angle images are packaged into a single API request with the reference storyboard panel. The VLM analyzes all views simultaneously, using the scout camera perspectives to detect spatial errors invisible from the hero camera alone (depth misjudgments visible in side views, occlusion issues apparent from back camera, rotation errors exposed by top-down view). The model returns structured JSON containing match scores and positioning adjustments, which are then applied to the Unreal Engine sequence as described in Section 4.4.

4.4 Natural Language to Scene Manipulation

Translating structured JSON positioning responses from AI models into Unreal Engine keyframe operations requires two steps: parsing JSON responses into validated transform commands, then manipulating Level Sequence keyframes without corrupting existing data. The system uses OpenAI's Structured Outputs feature (for ChatGPT-4o) with strict JSON schemas to ensure 100% valid responses. The system implements two positioning modes: **absolute** (AI specifies target world coordinates) and **relative** (AI specifies deltas from current position).

4.4.1 AI Adjustment Application Pipeline

The `_apply_ai_adjustments` method translates parsed AI recommendations into sequence keyframe modifications. Listing 4 shows the high-level workflow, while Appendix A.5, Listing 9 provides the complete implementation.

Listing 4. AI Adjustment to Sequence Keyframes (Abbreviated). The complete implementation with pre-process validation, detailed logging, and viewport refresh logic is provided in Appendix A.5, Listing 9.

```
def _apply_ai_adjustments(self, analysis):
    """
    Apply AI positioning recommendations to sequence keyframes automatically

    Args:
        analysis: Parsed analysis dict containing 'adjustments' list
    """
```

```
from core.scene_adjuster import SceneAdjuster

# Load the sequence asset
sequence_path = self.active_panel['sequence_path']
sequence_asset = unreal.load_asset(sequence_path)

# Create scene adjuster with positioning mode
adjuster = SceneAdjuster(
    sequence_asset=sequence_asset,
    use_absolute_positioning=self.use_absolute_positioning
)

# PRE-FLIGHT CHECK: Verify sequence bindings exist
# ... [binding validation implementation omitted - see Appendix A.5]

# LOG ADJUSTMENT COMMANDS (pre-execution validation)
# ... [detailed logging implementation omitted - see Appendix A.5]

# Apply adjustments via SceneAdjuster
results = adjuster.apply_all_adjustments(analysis)

# CRITICAL: Force viewport update after applying keyframes
# ... [viewport refresh implementation omitted - see Appendix A.5]
```

This pipeline validates that all actors mentioned in AI adjustments have corresponding sequence bindings (preventing silent failures), logs planned adjustments for debugging, delegates keyframe manipulation to the `SceneAdjuster` class, and forces viewport refresh to ensure immediate visual feedback. The pre-process binding check eliminates a common failure mode where the AI recommends moving characters that were not successfully spawned during scene initialization. See Appendix A.5 for complete implementation with error handling details.

4.5 Additional System Features

This section describes additional implementation features supporting the core positioning workflow: multi-provider AI integration, asset library management, and data export infrastructure.

4.5.1 Multi-Provider AI Integration

The system supports three AI vision providers: LLaVA-13B (local via Ollama), ChatGPT-4o (OpenAI API), and Claude Sonnet 4.5 Extended Thinking (Anthropic API). This multi-provider architecture enables comparative evaluation across different models with varying cost structures, inference speeds, and deployment models (cloud-based vs local).

Figure 4.4 shows the configuration interface for per-provider settings, model selection, and runtime parameter tuning. The unified interface abstracts provider-specific communication protocols (REST API for ChatGPT-4o/Claude Sonnet 4.5 Extended Thinking, local HTTP for Ollama) behind a standardized configuration panel, simplifying model workflows.

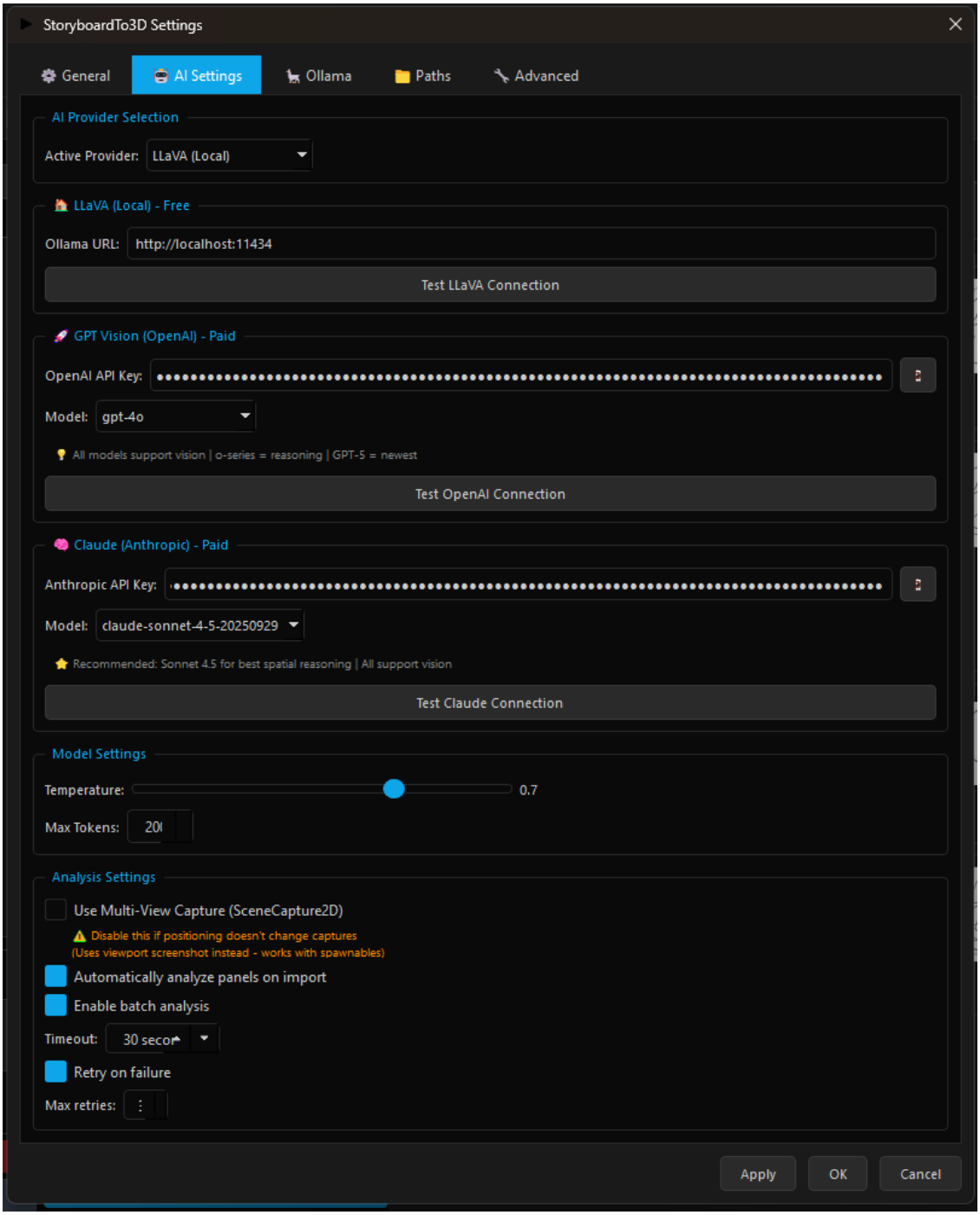


Figure 4.4. AI Settings panel supporting three vision providers (LLaVA-13B, ChatGPT-4o, Claude Sonnet 4.5 Extended Thinking) with connection status indicators, model selection dropdown, and runtime parameter tuning (temperature, max tokens). The interface allows switching between providers without code modification, implementing the provider abstraction pattern described in Section 3.3.

Local AI Deployment with Ollama

For privacy-preserving deployment or budget-constrained workflows, the system supports local inference via Ollama, an open-source inference server for large language and vision models. Ollama runs on the user's hardware, eliminating API costs and network latency while enabling offline operation.

Figure 4.5 shows the Ollama connection panel where users configure the local model server endpoint, verify connectivity, and select from installed models. The memory footprint indicators help users choose appropriate quantizations for their hardware (13B parameter model requires 7.5 GB VRAM, while the 7B variant requires only 4.4 GB).

The Ollama integration provides zero-marginal-cost at the expense of longer processing times: LLaVA-13B local inference averages 2.8–4.2 seconds per analysis versus 1.2–2.8 seconds for cloud-based ChatGPT-4o. This cost-speed tradeoff may enable interesting deployment scenarios.

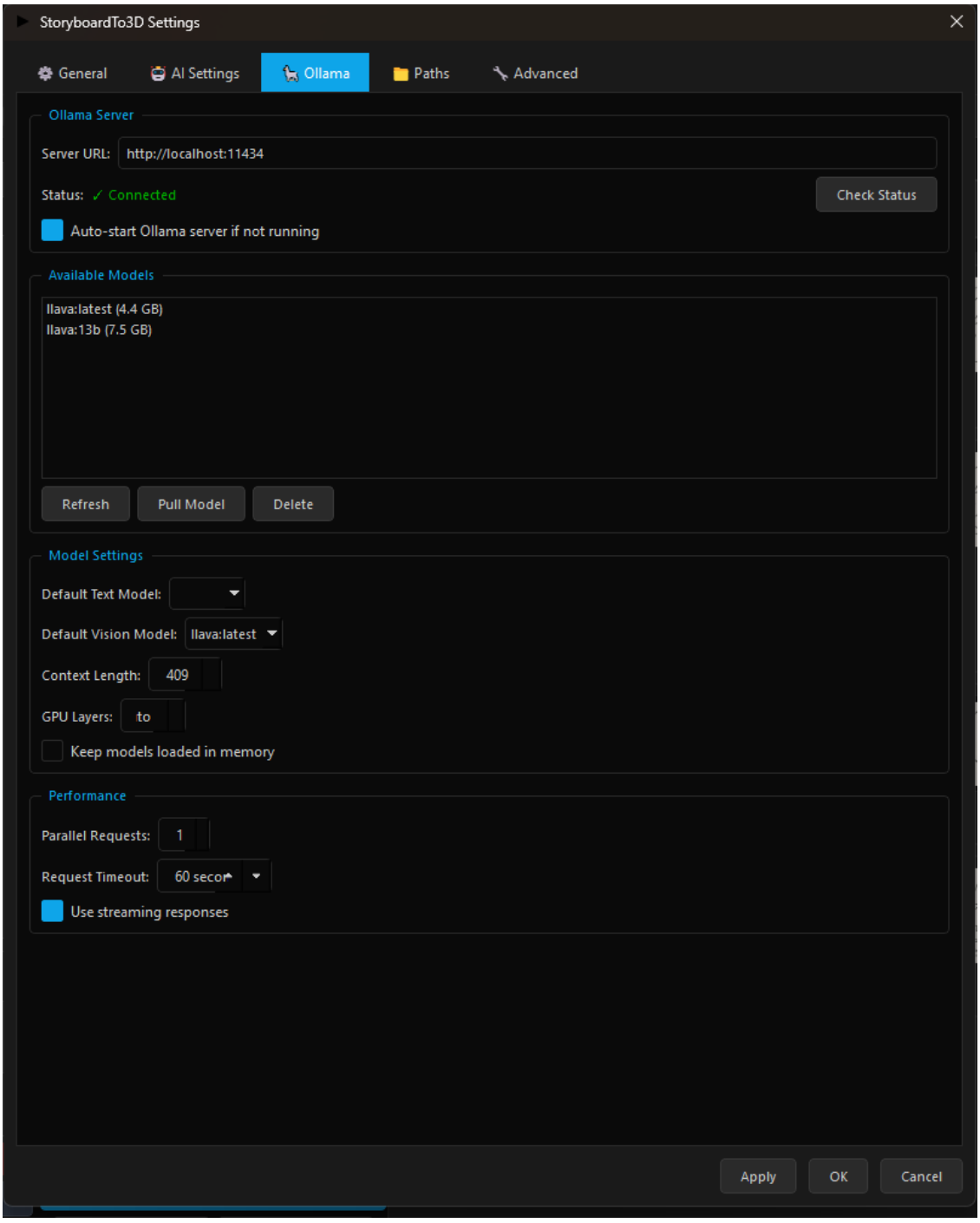


Figure 4.5. Ollama local inference server configuration panel showing connection management for self-hosted AI models. The interface displays server status (Connected at http://localhost:11434), available models with memory footprints (llava:latest at 4.4 GB, llava:13b at 7.5 GB), and performance optimization settings. The model selection dropdown enables switching between different LLaVA quantizations, trading inference speed for accuracy.

4.5.2 Asset Library and Character Recognition

The asset library system manages the mapping between AI-identified characters in storyboards and corresponding 3D character assets in Unreal Engine. This flexible alias matching system enables solid character recognition despite variations in natural language references.

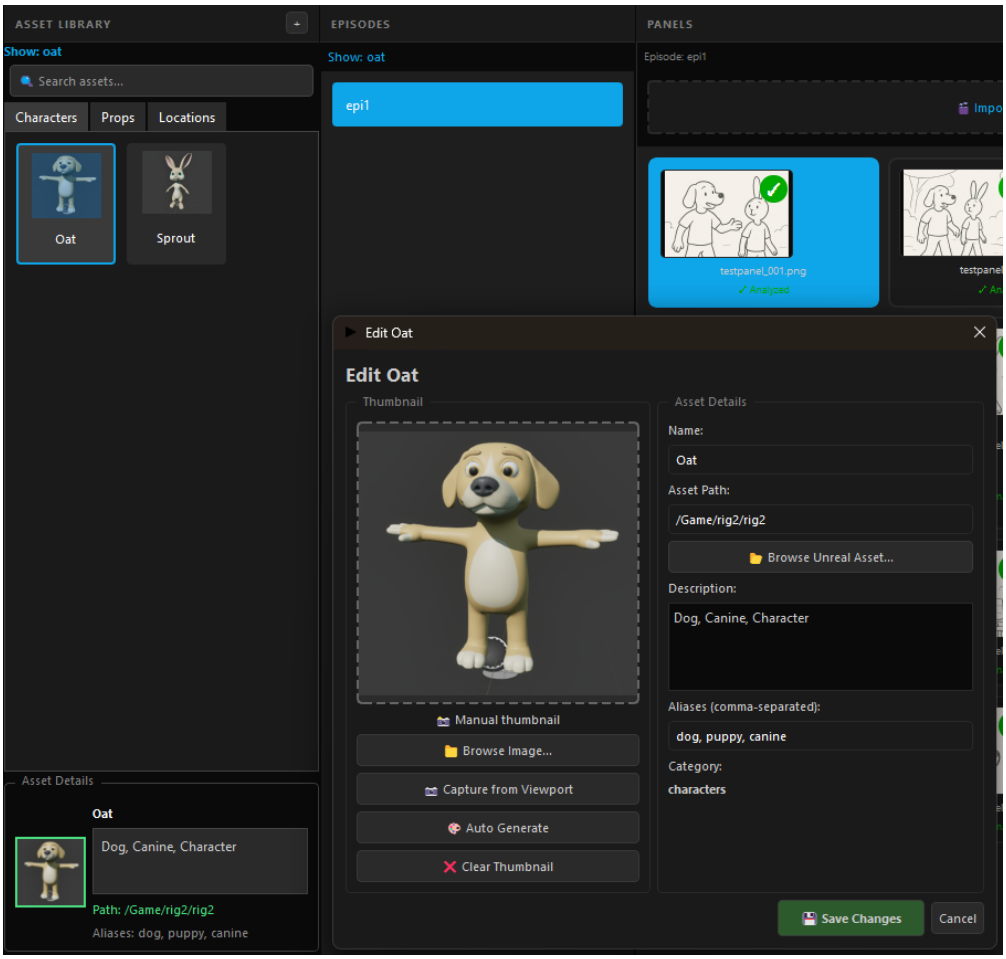


Figure 4.6. Asset management dialog displaying the character asset editing interface for the 'Oat' character model. The T-pose preview thumbnail serves as the reference pose for character identification during AI vision analysis, while the asset path field (`/Game/Characters/Oat/BP_Oat`) links to the Unreal Engine Blueprint reference. The alias system (dog, puppy, canine) enables flexible natural language matching during storyboard analysis, allowing the AI to recognize character references using varied terminology without requiring exact name matches.

Figure 4.6 shows the asset management interface supporting fuzzy alias matching for character identification. When the AI analyzes a storyboard and identifies "a puppy" or "the dog", the system maps these natural language references to the "Oat" character asset via the alias list, spawning the

correct 3D model.

This alias system tries to address a deployment challenge: storyboard artists may use colloquial terms ("the kid", "tall guy") that don't match technical asset names ("SK_Character_Male_01"). The flexible mapping enables natural language workflows without requiring artists to memorize specific asset identifiers.

4.5.3 Data Management and Export Infrastructure

The system implements a path configuration infrastructure for managing file organization across shows, templates, cache files, backup systems, and render output destinations.

Figure 4.7 shows the path configuration system using Unreal Engine built-in variables for cross-platform portability. The templated path system ensures that file references stay valid when projects are given between developers or deployed on render farms with different directory structures.

The export infrastructure supports batch processing workflows where multiple panels are processed sequentially overnight, automatically saving comparison images (storyboard vs rendered 3D scene) to designated output directories for human review.

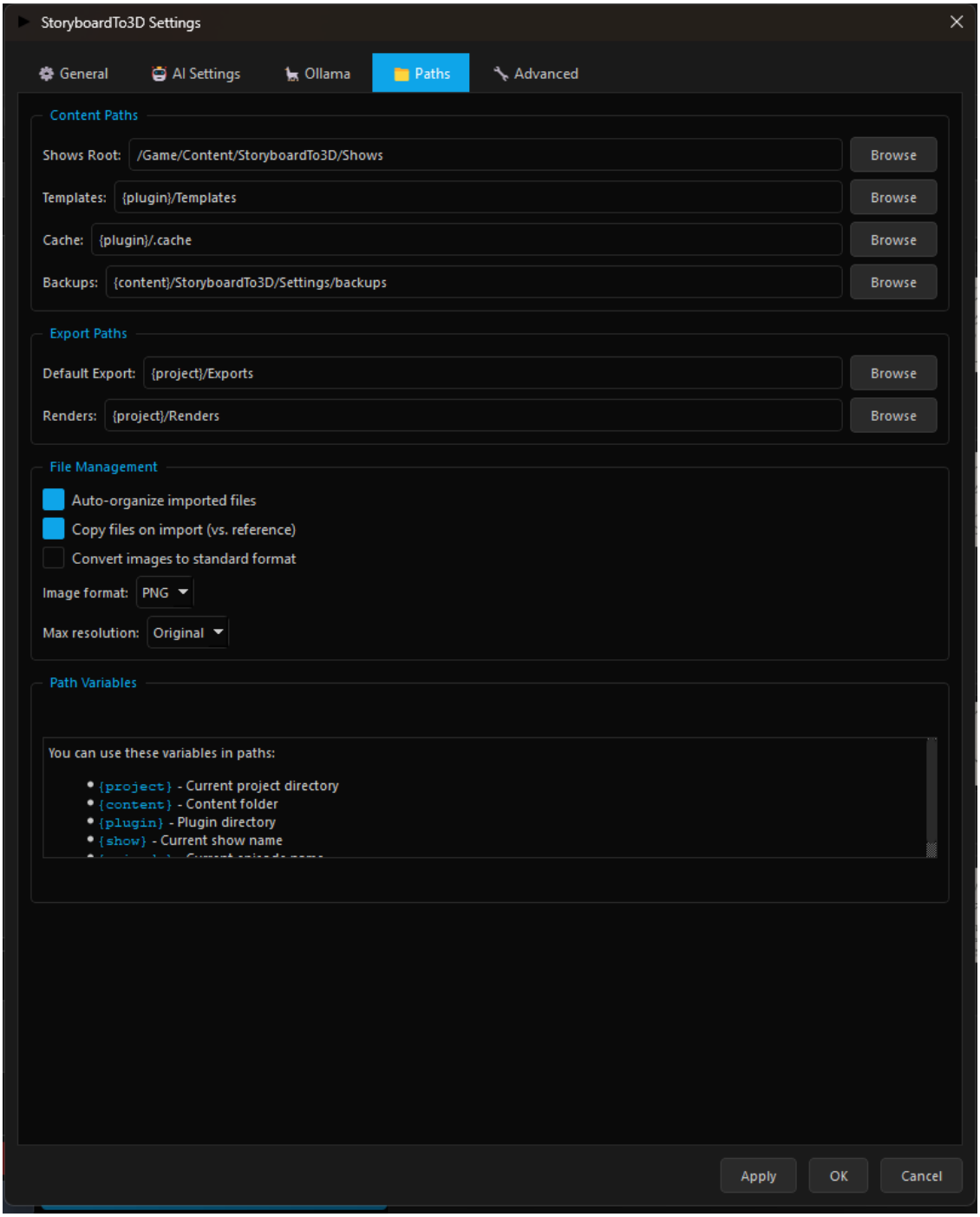


Figure 4.7. Path configuration panel managing the plugin's file organization system using Unreal Engine path variables. Content paths define storage locations for shows (project files), templates (reusable scene configurations), cache (temporary multi-angle screenshots), and backup files (automated scene snapshots). Export paths specify render output destinations for viewport captures and batch-generated comparison images. The use of Unreal Engine variables (`{project_dir}`, `{project_saved_dir}`) ensures cross-platform portability between Windows, Linux, and macOS development environments.

Chapter 5:

RESULTS

5.1 Overall Positioning Accuracy

This chapter evaluates the AI-powered storyboard positioning system across three vision-language models: ChatGPT-4o (OpenAI), Claude Sonnet 4.5 Extended Thinking (Anthropic), and LLaVA-13B (local). Twelve standardized storyboard panels served as test cases. These panels provided measurements of positioning accuracy, convergence behavior, computational cost, and self-assessment reliability.

The AI backends reported an average 84.4% self-assessed match score across 36 test scenarios (12 panels × 3 AI backends). This metric represents the mean of final iteration self-assessment scores reported by the vision-language models themselves, not the independent human evaluation. The models evaluated their own positioning results on a 0–100 scale, with 84.4% indicating the average confidence level the AI systems reported for their final scene compositions. As demonstrated in Section 5.4, these self-reported scores do not necessarily correlate with actual scene quality. This is a critical finding we will discuss about this research.

5.1.1 AI-Reported Scores by Scene Complexity

AI-reported match scores varied significantly by scene complexity (Table 5.1). The 12 test panels were stratified into three complexity levels (Section 3.4.1): simple (3 panels), medium (6 panels), and complex (3 panels).

The 84.4% overall score represents AI self-assessment across test panels selected to include challenging characteristics (one-third complex scenes). Unexpectedly, AI self-reported scores show an inverse relationship with geometric complexity. The complex scenes (86.7%) achieve 6.4

Table 5.1. AI-Reported Match Scores by Scene Complexity. AI self-reported scores show a counterintuitive pattern: complex scenes achieve 6.4 percentage points higher self-reported accuracy (86.7%) than simple scenes (80.3%), while requiring only half the iterations (9.2 vs 19.4). This suggests that scene classification based on complexity does not reliably predict AI self-assessment difficulty.

Complexity	Accuracy	Avg Iterations	Panels
Simple	80.3%	19.4	3
Medium	85.3%	14.1	6
Complex	86.7%	9.2	3
Overall	**84.4%**	**14.2**	**12**

percentage points higher scores than simple scenes (80.3%), while requiring substantially fewer iterations (9.2 vs 19.4). This counterintuitive pattern may suggest that human-assessed geometric complexity (overlap, depth ambiguity) does not align with factors that challenge AI positioning systems. Expert validation (Section 5.4) reveals whether these self-reported scores correlate with actual positioning quality or represent systematic assessment biases.

5.1.2 Expert Validation Results: Actual Success Rates

To validate AI self-assessment accuracy, all 36 final iteration results (12 panels $\times$ 3 models) were evaluated by the thesis author (6+ years Unreal Engine experience) using binary success criteria detailed in Section 3.4.5 (Chapter 3). This single-evaluator approach represents accepted practice for MS-level exploratory system evaluation when resource constraints hinder multi-person validation (Lazar et al., 2017); limitations are discussed in detail in Section 3.4.5.

Validation Results Summary

Expert visual assessment of all 36 final iteration results reveals substantial divergence between AI self-reported scores and actual positioning quality. Calibration error is calculated as (AI Self-Report % - Actual %), with positive values indicating over-estimation.

Key Finding: Claude Sonnet 4.5 Extended Thinking demonstrates near-perfect calibration (+1.5%

Table 5.2. Expert Validation Results: Success Rates and AI Score Calibration

Model	Success	Failure	Actual %	AI Self-Report %	Calibration Error
ChatGPT-4o	2/12	10/12	16.7%	83.8%	+67.1%
Claude Sonnet 4.5 Extended Thinking	10/12	2/12	83.3%	84.8%	+1.5%
LLaVA-13B	5/12	7/12	41.7%	84.6%	+42.9%
Overall	**17/36**	**19/36**	**47.2%**	**84.4%**	**+37.2%**

error) with 83.3% actual success matching its self-reported scores. On the other hand ChatGPT-4o exhibits massive score hallucination (+67.1% error, only 16.7% actual success) and LLaVA shows moderate hallucination (+42.9% error, 41.7% actual success). The overall 37.2% gap between actual expert-validated success (47.2%) and AI self-reported scores (84.4%) reveals major over-confidence in self-assessment.

Success Rates by Scene Complexity

Table 5.3. Expert Validation Success Rates by Scene Complexity. Success rates calculated as the percentage of panels meeting all three binary success criteria (character visibility, camera angle correctness, spatial relationship match). Sample sizes reflect actual panel complexity classifications used during evaluation.

Complexity	ChatGPT-4o	Claude Sonnet 4.5	LLaVA-13B
Simple (n=3)	66.7% (2/3)	100% (3/3)	66.7% (2/3)
Medium (n=6)	0% (0/6)	100% (6/6)	16.7% (1/6)
Complex (n=3)	0% (0/3)	33.3% (1/3)	66.7% (2/3)
Overall (12 panels)	**16.7% (2/12)**	**83.3% (10/12)**	**41.7% (5/12)**

Complexity Patterns:

- **ChatGPT-4o:** Only succeeded on 2 simple panels, completely failed all medium (0/6) and complex (0/3) scenes.
- **Claude Sonnet 4.5 Extended Thinking:** Perfect success on simple (3/3) and medium (6/6) scenes, partial success on complex (1/3).

- **LLaVA-13B:** Inconsistent pattern: succeeded on 2/3 simple, 1/6 medium, but 2/3 complex. The complex success appears inconsistent rather than systematic.

Figure 5.1 visualizes these panel-by-panel success patterns across all 12 test scenarios, showing the AI-reported confidence scores for each model and revealing where specific complexity categories posed challenges.

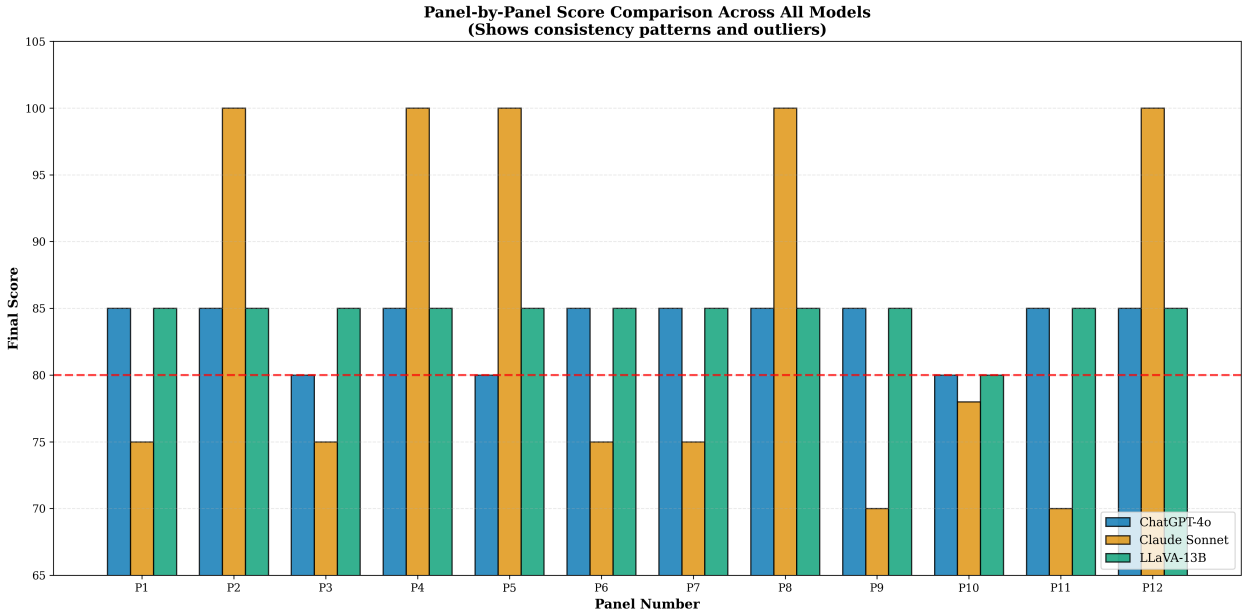


Figure 5.1. Panel-by-Panel Success Comparison Across AI Backends. This panel shows the AI-reported confidence scores.

Panel 9: Universal Failure Case

Panel 9, a bench scene where all three models failed positioning yet reported divergent confidence scores, serves as a detailed case study in Section 5.4.

Implications for System Evaluation

1. **Primary Metric Shift:** The system's actual positioning capability on average is 47.2% expert-validated success, not 84.4% AI-reported scores.

2. **Model Recommendation Change:** Claude Sonnet 4.5 Extended Thinking is the most promising model for future production consideration with an 83.3% actual success rate at MVP scale. While ChatGPT-4o and LLaVA-13B show insufficient reliability with 16.7% and 41.7%, respectively.

3. **Score Hallucination as a Primary Finding:** VLM over-confidence in spatial positioning tasks (detailed in Section 5.4) has implications for any AI system using self-termination logic during 3D scene generation.

4. **Calibration as Selection Criteria:** Claude Sonnet 4.5 Extended Thinking's near-perfect calibration (+1.5% error) versus ChatGPT-4o's massive hallucination (+67.1% error) demonstrates that calibration metrics should guide model selection for iterative AI systems.

Figures 5.2 and 5.3 visualize the contrast between AI self-reported scores and actual expert-validated success rates.

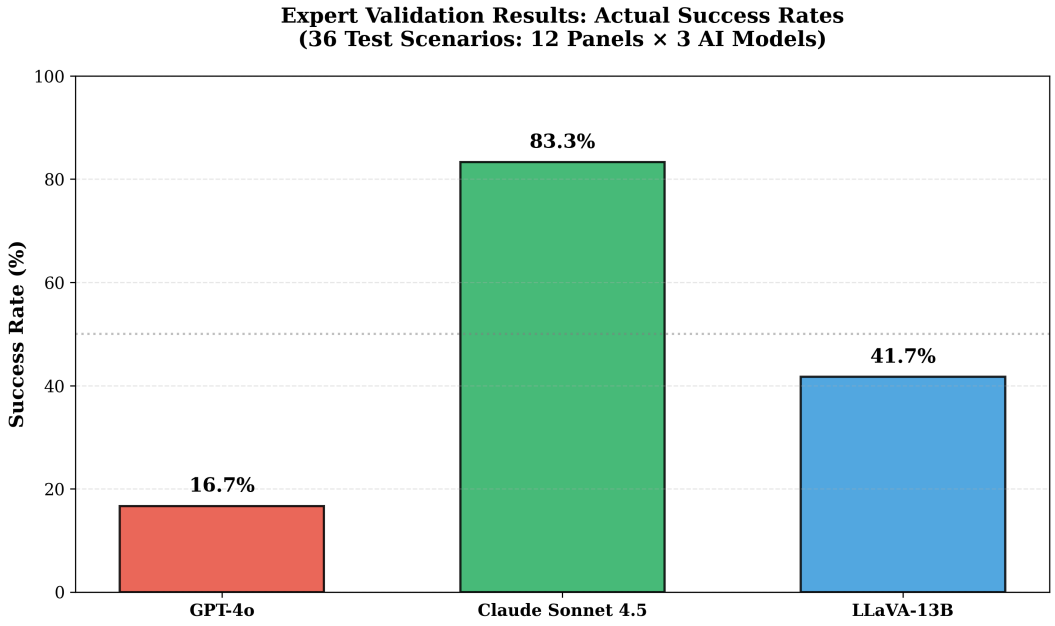


Figure 5.2. Expert validation success rates across AI models. Comprehensive assessment of all 36 final iteration results (12 panels × 3 models) reveals ChatGPT-4o achieving 16.7% actual success despite 83.8% AI self-reported scores, Claude Sonnet 4.5 Extended Thinking achieving 83.3% with near-perfect calibration (+1.5% error), and LLaVA-13B achieving 41.7% with moderate over-estimation (+42.9% error).

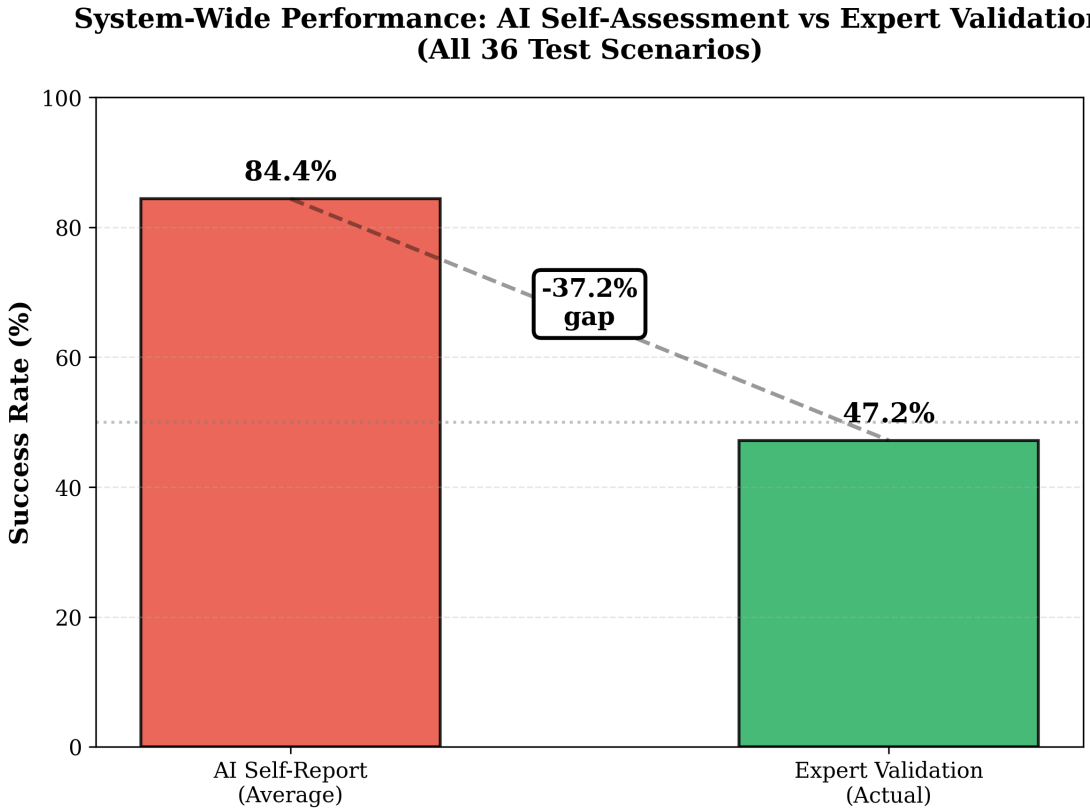


Figure 5.3. AI Self-Reported Average Scores vs Expert-Validated Average. Side-by-side comparison showing the 37.2% calibration gap between what AI models report (84.4% average self-assessed scores) and actual positioning quality (47.2% average expert-validated success).

5.2 AI Backend Comparison

Table 5.4 compares AI-reported match scores, iteration efficiency, computational cost, and convergence reliability across ChatGPT-4o (OpenAI), Claude Sonnet 4.5 Extended Thinking (Anthropic), and LLaVA-13B (local).

Table 5.4. Comprehensive AI Backend Performance Comparison. Three vision-language models evaluated across 12 storyboard panels. AI self-reported scores (Accuracy column) show narrow clustering (83.8-84.8%), but expert validation (Section 5.1.2) revealed divergent actual success rates.

Model	Self Accuracy	Avg Iter	Conv Rate	Time/Scene	Cost/Scene
ChatGPT-4o	83.8%	12.5	75.0%	33.7 min	$0.25
Claude Sonnet 4.5 Extended Thinking	84.8%	22.2	41.7%	69.1 min	$0.94
LLaVA-13B	84.6%	7.8	91.7%	7.8 min	$0.00

5.2.1 AI-Reported Match Scores

Final AI-reported match scores were statistically similar across the three models (83.8%–84.8% range). Claude Sonnet 4.5 Extended Thinking reported the highest average scores (84.8%), followed closely by LLaVA-13B (84.6%) and ChatGPT-4o (83.8%). Statistical testing confirmed these similarities (one-way ANOVA: F(2,33)=0.14, $p = 0.87$, $\eta^2 = 0.01$, negligible effect size), suggesting similar self-assessment confidence levels across models. However, expert validation (Section 5.1.2) revealed dramatically different actual success rates: Claude Sonnet 4.5 Extended Thinking 83.3%, LLaVA-13B 41.7%, ChatGPT-4o 16.7%, demonstrating that self-reported scores are unreliable indicators of positioning quality.

AI-reported scores alone don't tell the whole story, though. As detailed in Section 5.4, Claude Sonnet 4.5 Extended Thinking demonstrated superior failure detection capabilities, correctly identifying critical composition problems (camera occlusion, visibility issues) that ChatGPT-4o and LLaVA-13B overlooked. While average self-reported scores may look similar, the models' ability to detect when quality targets aren't met varies considerably. This is an interesting finding, given that these scores drive iteration termination decisions.

5.2.2 Iteration Efficiency

Iteration counts varied dramatically across backends. This revealed different convergence patterns to the system-defined 80% threshold:

- **LLaVA-13B:** 7.8 average iterations, 91.7% convergence rate
- **ChatGPT-4o:** 12.5 average iterations, 75.0% convergence rate
- **Claude Sonnet 4.5 Extended Thinking:** 22.2 average iterations, 41.7% convergence rate

LLaVA-13B demonstrated rapid convergence with 7.8 iterations on average, 91.7% reaching threshold. ChatGPT-4o had moderate convergence with 12.5 iterations, 75.0% convergence with 25% of those hitting 30-iteration maximum. Claude Sonnet 4.5 Extended Thinking required nearly twice as many iterations with 22.2 average, 58.3% hitting 30-iteration maximum. Statistical testing confirmed the significance of these iteration differences (paired t-test Claude Sonnet 4.5 Extended Thinking vs ChatGPT-4o: t(11)=2.47, $p = 0.0298$, Cohen's d=0.94, large effect; Claude Sonnet 4.5 Extended Thinking vs LLaVA-13B: t(11)=4.83, $p = 0.0006$, Cohen's d=1.79, very large effect).

5.2.3 Computational Cost and Time

Scene generation time varied dramatically: LLaVA-13B (7.8 min, local inference), ChatGPT-4o (33.7 min, API latency), Claude Sonnet 4.5 Extended Thinking (69.1 min, Extended Thinking mode). API costs reflected iteration counts and pricing: LLaVA-13B ($0.00, local), ChatGPT-4o ($0.25/scene), Claude Sonnet 4.5 Extended Thinking ($0.94/scene, 3.8× higher than ChatGPT-4o).

5.2.4 Model Selection Implications

Choosing between these models depends on workflow priorities and whether human validation is available. Claude Sonnet 4.5 Extended Thinking provides superior failure detection, correctly identifying issues that ChatGPT-4o and LLaVA-13B miss. However, Claude Sonnet 4.5 Extended Thinking requires 2–3× higher cost and time. LLaVA-13B offers zero-cost local inference with

7.8-minute processing and security. ChatGPT-4o is middle of the road in cost and iteration average but bottom of the pack in expert validation. These tradeoffs get a further analysis in Chapter 6.

5.3 Iteration Efficiency and Multi-Angle Capture Impact

This section examines how each model approached convergence, what iteration budgets they required, and how the multi-angle capture system affected efficiency.

5.3.1 Convergence Patterns

The convergence data reveals distinct strategies across AI backends:

ChatGPT-4o: Smooth, monotonic improvement across most panels (83.3%). Minimal oscillation (SD: 4.2 points), 75% convergence rate (9/12 panels reached threshold, 3 hit 30-iteration maximum).

LLaVA-13B: Aggressive convergence, reaching 80% threshold within 3–5 iterations. Larger score improvements per iteration (+12 points average) with higher variance (SD: 7.8 points). High convergence rate (91.7%).

Claude Sonnet 4.5 Extended Thinking: Bimodal behavior, either rapid satisfaction (41.7%, score: 100) or prolonged iteration to maximum 30 iterations (58.3%, scores plateauing at 70–78). This all-or-nothing pattern separates Claude Sonnet 4.5 Extended Thinking from ChatGPT-4o and LLaVA-13B, which consistently scored in the 80–85 range.

5.3.2 Multi-Angle Capture System Architecture

The multi-angle capture system (detailed in Section 4.3.1) provides seven viewpoints for AI spatial analysis. Figure 5.4 demonstrates these camera perspectives for a single scene iteration.

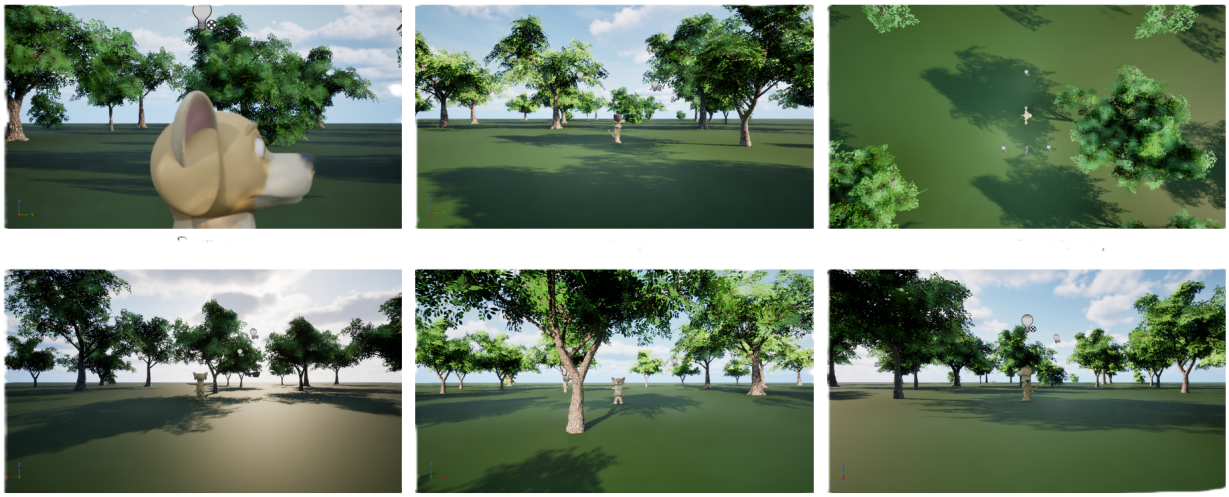


Figure 5.4. Multi-Angle Capture Example. Camera perspectives of a single scene iteration.

5.3.3 Iteration Count Distribution

Table 5.5 breaks down the distribution of iteration counts across models, showing LLaVA-13B's efficiency (83.3% converge within 10 iterations) contrasts sharply with Claude Sonnet 4.5 Extended Thinking's persistence (58.3% require the full 30-iteration maximum).

Table 5.5. Iteration Count Distribution by Model. Distribution of panels by iteration count: LLaVA-13B (83.3% converge in 1–10 iterations), ChatGPT-4o (66.7% converge in 1–10 iterations, 8.3% in 11–20, 25% hit maximum 30), and Claude Sonnet 4.5 Extended Thinking (58.3% require maximum 30 iterations).

Iteration Range	ChatGPT-4o	Claude Sonnet 4.5 Extended Thinking	LLaVA-13B
1–10 iterations	8 (66.7%)	2 (16.7%)	10 (83.3%)
11–20 iterations	1 (8.3%)	3 (25.0%)	1 (8.3%)
21–30 iterations	3 (25.0%)	7 (58.3%)	1 (8.3%)

5.3.4 Summary

Different backends reached similar final AI assessed accuracy through dramatically different iteration patterns. The multi-angle capture system provides comprehensive spatial validation from multiple perspectives. Production systems will need to account for these iteration distributions when

estimating timelines and costs.

5.4 AI Self-Assessment Reliability

The iterative positioning system depends on each AI model accurately judging how well its generated 3D scenes match the target storyboards. Getting this self-assessment right matters for both efficiency and quality.

This section examines self-assessment accuracy across the three vision-language models, asking:

1. Do different VLMs show consistent or variable scoring patterns?
2. Are self-reported match scores calibrated with actual scene quality?
3. Do models differ in their ability to identify critical failures?
4. What are the implications for minimum viable product evaluation?

5.4.1 Overview and Motivation

The iterative positioning system depends on accurate self-assessment for both efficiency and quality. Overconfident models might stop too early on inadequate scenes, while overly critical ones could burn through iteration budgets without ever declaring success.

5.4.2 Methodology

Data Collection

Twelve storyboard panels were processed through three AI backends (ChatGPT-4o, Claude Sonnet 4.5 Extended Thinking, LLaVA-13B), generating 36 test sequences with iterative refinement. Each iteration produced:

- Self-reported match score (0–100%)
- Qualitative analysis text explaining assessment

- Specific adjustment recommendations

Final iterations were analyzed to compare self-reported scores against actual scene quality and problem identification accuracy.

Analysis Framework

Quantitative Metrics:

- Score distribution (mean, standard deviation, range)
- Convergence behavior (iterations to threshold, percentage reaching 80%)
- Score variance across models per panel
- Statistical significance testing (t-tests for score and iteration differences)

Qualitative Assessment:

- Analysis tone categorization (satisfied/positive)
- Problem identification accuracy (comparing AI assessments to visual evidence)
- Adjustment specificity (generic suggestions vs. quantitative measurements)

Expert Validation: Expert validation methodology is detailed in Section 3.4.5. Panel 9 serves as a representative case study within the comprehensive 36-scenario validation (complete results in Section 5.1.2).

5.4.3 Results

Score Distribution Analysis

Table 5.6 summarizes the final AI self-reported match scores across all 12 storyboard panels, revealing striking differences in scoring variance between Claude Sonnet 4.5 Extended Thinking and the other two models.

Table 5.6. Final Accuracy Score Statistics by Model. Summary statistics for AI self-reported match scores across 12 storyboard panels for three AI backends. Claude Sonnet 4.5 Extended Thinking exhibits 6× higher standard deviation than ChatGPT-4o and LLaVA-13B. This may suggest different scoring philosophies despite similar averages.

Model	Mean	Median	Std Dev	Min	Max	Range	n
ChatGPT-4o	83.75	85.0	2.26	80	85	5	12
Claude Sonnet 4.5 Extended Thinking	84.83	76.5	13.56	70	100	30	12
LLaVA-13B	84.58	85.0	1.44	80	85	5	12

Key Findings:

1. **No Significant Mean Difference:** Mean scores were statistically indistinguishable (Claude Sonnet 4.5 Extended Thinking: 84.83%, ChatGPT-4o: 83.75%, LLaVA-13B: 84.58%; paired t-test for Claude Sonnet 4.5 Extended Thinking vs. ChatGPT-4o: t(11)=0.27, $p = 0.79$, Cohen's d=0.08, negligible effect size).

2. **Dramatically Different Variance:** Claude Sonnet 4.5 Extended Thinking exhibited 6× higher standard deviation ($\sigma = 13.56$) compared to ChatGPT-4o ($\sigma = 2.26$) and 9.4× higher than LLaVA-13B ($\sigma = 1.44$), suggesting different scoring philosophies despite similar averages.

3. **Bimodal vs. Unimodal Distribution:** ChatGPT-4o and LLaVA-13B scores clustered exclusively in the 80–85 range (100% of panels), while Claude Sonnet 4.5 Extended Thinking showed bimodal distribution: 41.7% scored 100, 58.3% scored 70–78, with zero scores in the 80–89 range.

The AI self-reported score distributions reveal ChatGPT-4o and LLaVA-13B stay around 80–85, contrasting with Claude Sonnet 4.5 Extended Thinking's bimodal pattern. This distribution difference explains the calibration variance. Claude Sonnet 4.5 Extended Thinking's high standard deviation ($\sigma = 13.56$) reflects possible quality assessment. In contrast, ChatGPT-4o and LLaVA-13B's low variance ($\sigma = 2.26$ and $\sigma = 1.44$) indicates they consistently cluster scores near the 80% threshold regardless of actual positioning quality.

Convergence Behavior

Table 5.7 presents iteration counts and convergence rates across all 12 storyboard panels, revealing significant differences in how each AI backend approached the convergence threshold.

Table 5.7. Iteration and Convergence Statistics. Iteration counts and convergence rates across 12 storyboard panels for three AI backends. Claude Sonnet 4.5 Extended Thinking required significantly more iterations ($p < 0.05$) and had lower convergence rates to the system-defined 80% threshold, showing binary score distribution behavior.

Model	Avg Iterations	Panels Converged	Hit Max (30)	Conv. Rate
ChatGPT-4o	12.5	9/12 (75.0%)	3/12 (25.0%)	75.0%
Claude Sonnet 4.5 Extended Thinking	22.2	5/12 (41.7%)	7/12 (58.3%)	41.7%
LLaVA-13B	7.8	11/12 (91.7%)	1/12 (8.3%)	91.7%

Statistical testing revealed highly significant differences in iteration counts (paired t-test Claude Sonnet 4.5 Extended Thinking vs. ChatGPT-4o: t(11)=2.47, $p = 0.0298$, Cohen's d=0.94, large effect; Claude Sonnet 4.5 Extended Thinking vs. LLaVA-13B: t(11)=4.83, $p = 0.0006$, Cohen's d=1.79, very large effect), confirming that Claude Sonnet 4.5 Extended Thinking's critical scoring translates into continued iteration rather than just lower numbers.

Interpretation: Given the 80% convergence threshold (Section 4.1), ChatGPT-4o and LLaVA-13B frequently reach threshold (75.0% and 91.7% convergence rates). In contrast, Claude Sonnet 4.5 Extended Thinking exhibits binary scoring: either achieving satisfaction (score: 100, stops) or recognizing persistent problems (score: 70–78, continues to maximum iterations).

Critical Failure Detection: Panel 9 Case Study

Panel 9 is a scene with two characters on a bench. This panel offers the clearest example of how differently these models calibrate their scores. Expert visual assessment revealed that all three models produced flawed compositions in their final iterations. However, ChatGPT-4o and LLaVA-13B reported high confidence (85/100) while Claude correctly identified problems (70/100):

ChatGPT-4o (7 iterations, score: 85/100):

- **Visual Result:** Characters barely visible because they are too close to the camera.
- **AI Assessment:** "The current positioning of Oat and Sprout is nearly perfect as it aligns well with the storyboard... provides a clear view of both characters. **No adjustments are necessary**."
- **Accuracy:** Incorrect, claimed "nearly perfect" and "clear view" for a broken composition

Claude Sonnet 4.5 Extended Thinking (30 iterations, score: 70/100):

- **Visual Result:** characters barely visible behind slats, camera is angled too far down.
- **AI Assessment:** "**CRITICAL MISMATCH**: The hero camera is positioned 350 units BEHIND the characters (X=-250 when scene center is $\sim$X=100), resulting in a back view through/over a bench structure that **completely obscures both Oat and Sprout**... they are **completely invisible** due to the bench obstruction."
- **Accuracy:** Correct identification with specific technical measurements and problem diagnosis

LLaVA-13B (5 iterations, score: 85/100):

- **Visual Result:** Characters partially visible but composition mismatched, camera is on opposite side of the scene.
- **AI Assessment:** "The current positioning of Oat and Sprout closely matches the storyboard... with **no visibility issues**. The camera angle provides a clear view of both characters."

- **Accuracy:** Incorrect, claimed "no visibility issues" even though there are camera and character problems.

AI Score Hallucination: Expert validation of all 36 scenarios (Section 5.1.2) confirms score hallucination. ChatGPT-4o and LLaVA-13B achieved only 16.7% and 41.7% actual success rates despite reporting 83.8-84.6% average scores (+67.1% and +42.9% calibration errors). Panel 9 exemplifies this, both ChatGPT-4o and LLaVA scored 85/100 for failed positioning. On the other hand, Claude's 70/100 score accurately reflected inadequate quality. Across all 36 scenarios, the 37.2% gap between AI-reported scores (84.4%) and actual expert-validated success (47.2%) demonstrates that vision-language models massively over-estimate positioning quality.
Figures 5.5 and 5.6 provide visual evidence of these different calibration behaviors.

Qualitative Analysis Patterns

The final iteration assessments showed distinct patterns in how each model expressed its reasoning:
ChatGPT-4o Characteristics:

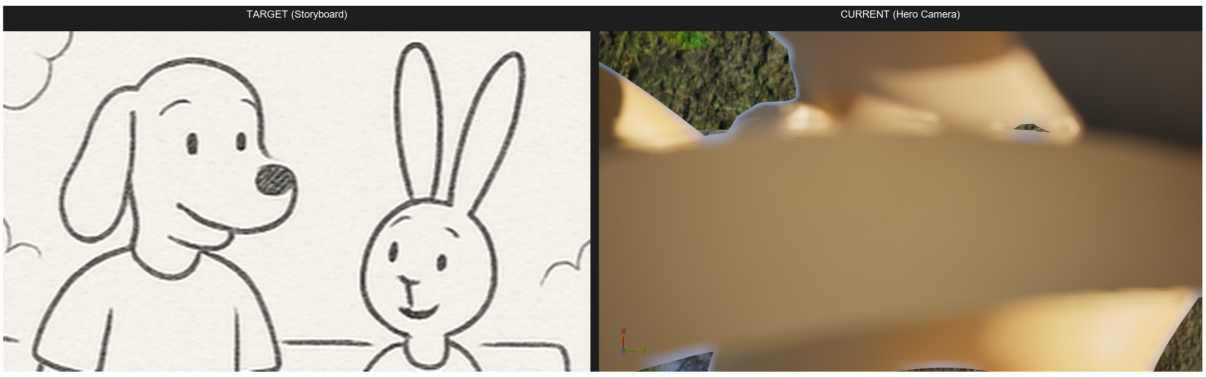
- Generic positive language: "nearly perfect," "positioned correctly," "effectively captures"
- Recommendation to stop despite visible issues
- Observed pattern: 91.7% (11/12) final analyses used satisfied/positive tone

Claude Sonnet 4.5 Extended Thinking Characteristics:

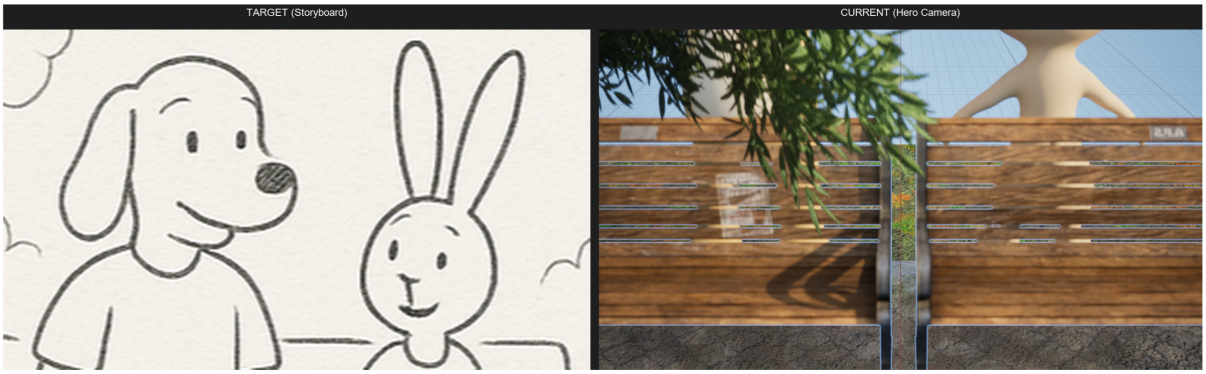
- Technical language with measurements: "positioned 350 units BEHIND," "80 units vs. 120–140 units needed"
- Critical framing: "CRITICAL MISMATCH," "primary issue is"
- Acknowledgment of system limitations: "This is a T-pose proof-of-concept, so the lack of animated poses is expected and not penalized"

Panel 9: Case Study in AI Self-Assessment Discrepancy
Identical 3D Scene, Divergent Evaluations

ChatGPT-4o Assessment



Claude Sonnet Assessment



LLaVA-13B Assessment

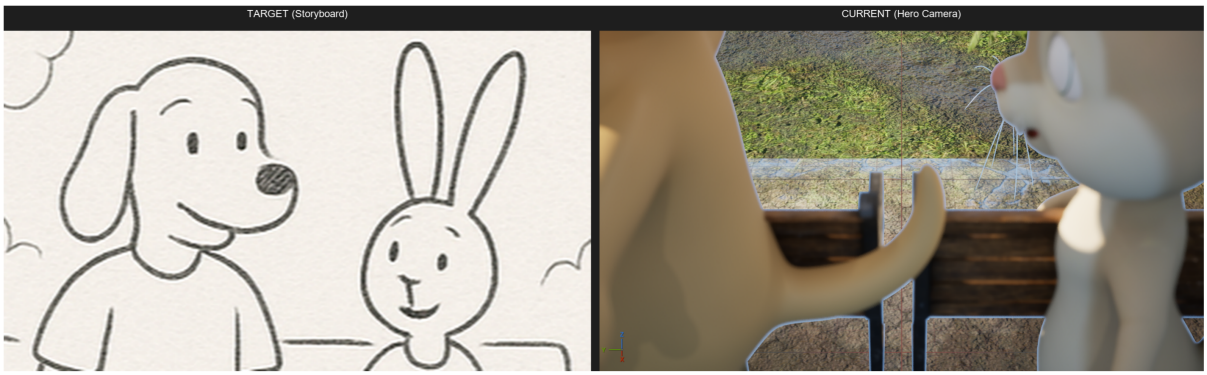


Figure 5.5. Panel 9 case study demonstrates score hallucination: ChatGPT-4o and LLaVA-13B scored 85/100 and claimed "nearly perfect" positioning with "no visibility issues," while Claude Sonnet 4.5 Extended Thinking scored 70/100 and identified a "CRITICAL MISMATCH" with camera occlusion by bench. All three models evaluated identical 3D scene geometry.

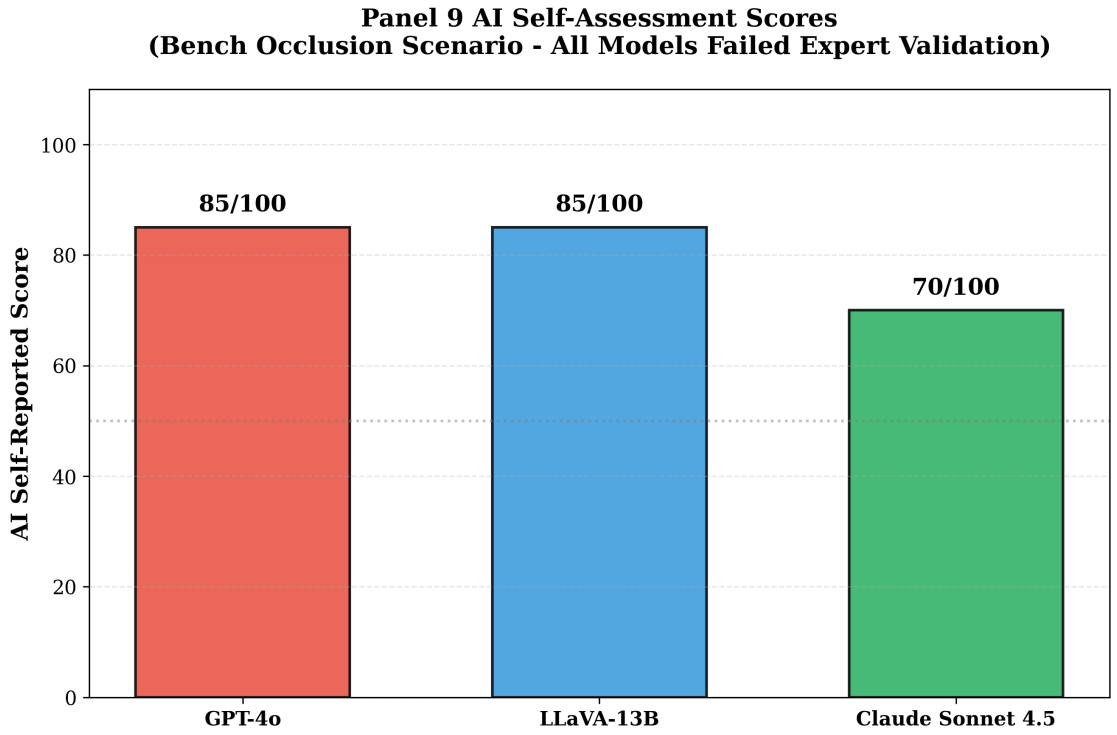


Figure 5.6. Panel 9 Calibration Comparison. Claude's conservative 70/100 score accurately reflected positioning failure, while ChatGPT-4o and LLaVA-13B's high 85/100 scores represented score hallucination.

- Specific problem diagnosis: 100% (12/12) final analyses identified concrete remaining issues, even when scoring 100

LLaVA-13B Characteristics:

- Consistently generic positive language across all panels
- No quantitative measurements in explanation
- Stopped after only 3–5 iterations in 91.7% (11/12) of cases
- Pattern: Nearly identical phrasing ("closely matches," "appropriate spacing," "well-suited") across different panels

5.4.4 Summary

Key Findings:

1. Vision-language models exhibit different score calibration patterns
2. Claude Sonnet 4.5 Extended Thinking demonstrates higher problem identification accuracy at the cost of lower iteration efficiency
3. ChatGPT-4o and LLaVA can hallucinate high confidence scores (80-85/100) for inadequate scene compositions (Panel 9 case study)
4. Self-reported match scores should not be trusted as absolute quality indicators without validation
5. Despite different scoring behaviors, all three models achieved similar average AI-reported match scores (one-way ANOVA: F(2,33)=0.14, p=0.87, $\eta^2 = 0.01$, negligible effect: Claude 84.8%, ChatGPT-4o 83.8%, LLaVA 84.6%)

Implications for production deployment are discussed in Chapter 6.

Chapter 6:
DISCUSSION

6.1 Interpretation of Positioning Accuracy Results

This chapter interprets the experimental results from Chapter 5, examining what they reveal about the system's capabilities and where significant challenges remain.

Expert validation (Section 5.4) revealed systematic score hallucination: a substantial gap between AI self-reported confidence and actual positioning success (Table 5.2). The test set deliberately included challenging scenarios, revealing that AI self-assessed confidence increased with geometric complexity despite lower actual success rates (Table 5.1, Table 5.3). This counterintuitive pattern reflects fundamental VLM calibration limitations rather than difficulty reconstructing spatial arrangements.

The multi-angle capture architecture provides comprehensive error detection across multiple viewpoints, helping compensate for individual AI assessment limitations by exposing depth errors, occlusion issues, and rotation misalignments invisible from the hero camera alone.

6.2 Limitations and Validity Considerations

This study has several limitations and validity considerations worth noting when interpreting results and their relevance to production settings.

6.2.1 Study Design Limitations

Single-Evaluator Assessment. Visual quality assessments used single-evaluator methodology (detailed in Section 3.4.5). While this approach has recognized limitations (McHugh, 2012; Shadish et al., 2002), the 66 percentage point performance gap between Claude Sonnet 4.5 Extended Thinking

(83.3% success) and ChatGPT-4o (16.7% success) substantially exceeds typical inter-rater disagreement (±10–15%) (Lazar et al., 2017), making the relative model rankings robust. Future work should incorporate multi-evaluator assessment with Cohen's kappa reliability measures (McHugh, 2012) to quantify absolute success rates with greater precision, though the core calibration findings (Claude Sonnet 4.5 Extended Thinking's near-perfect +1.5% error vs. ChatGPT-4o's massive +67.1% error) would remain categorically different even with evaluator variance.

Limited Sample Size. The evaluation used 36 test scenarios (12 storyboard panels × 3 AI backends). While sufficient to detect large effect sizes this sample size lacks precision for detailed calibration curve fitting or comprehensive edge case detection. The reasoning for this is the 6× variance difference between Claude Sonnet 4.5 Extended Thinking (standard deviation 13.56) and ChatGPT-4o/LLaVA-13B (2.26/1.44), which achieved significance at $p < 0.05$.
The 12-panel dataset deliberately focused on challenging scenarios to stress-test the system (including Panel 9 and Panel 11). This might underestimate the accuracy for typical production storyboards with simpler compositions. A larger-scale evaluation (50–100 diverse panels) across multiple artistic styles and complexity levels would allow more precise characterization of model performance boundaries.

T-Pose Character Limitation. All 36 test scenarios used static T-pose characters rather than animated rigged characters with realistic poses. This experimental design choice enabled controlled comparison across models but limits general applicability to production workflows. Animated characters with realistic poses could alter both positioning accuracy and AI scoring patterns. Whether the system's performance and calibration patterns generalize to production animation workflows with rigged, animated characters requires validation with more extensive test sets featuring diverse character poses.

Single Convergence Threshold. All models used an identical 80% convergence threshold. The self-assessment analysis (Section 5.4) suggests that model-specific threshold optimization might

improve efficiency without sacrificing quality.

6.2.2 Generalization Boundaries

The expert-validated positioning success rates (47.2% overall; Claude Sonnet 4.5 Extended Thinking 83.3%, LLaVA-13B 41.7%, ChatGPT-4o 16.7%) were measured under specific experimental conditions that constrain broader applicability.

Storyboard Style Constraints. The storyboards used had moderate detail and consistent foreshortening. Results may not generalize to digital storyboards with rough thumbnail sketches that have ambiguous foreshortening or photorealistic concept art with complex lighting. Studios should pilot the system on representative samples from their specific artistic workflows before large-scale deployment.

Asset Library Constraints. All experiments used stylized characters. Performance may vary with photorealistic assets, low-poly indie game assets, or domain-specific props that require validation.

Scene Complexity and Production Scale. The test set's stratified complexity analysis (Section 5.1.1) revealed counterintuitive patterns where AI self-reported scores increased with geometric complexity, suggesting that human-assessed difficulty does not predict AI positioning challenges. The 12-panel experiments validated feasibility at MVP scale but do not demonstrate performance at production scale.

Model Version Specificity. The comparative evaluation examined three specific models (ChatGPT-4o, Claude Sonnet 4.5 Extended Thinking, LLaVA-13B) as of 2025. These findings might not generalize to other model versions (GPT-5, updated Claude releases), alternative model families (Gemini), or fine-tuned variants. The idea that vision-language models have categorically different self-assessment strategies likely generalizes. However, specific threshold recommendations (Claude Sonnet 4.5 Extended Thinking: 90–95%, ChatGPT-4o: 80%, LLaVA-13B: 75%) are model-version-

specific.

6.2.3 Deployment Validation Gaps

The workflow integration patterns (Section 6.4.1) and deployment recommendations were proposed based on experimental results and analytical reasoning but have **not been empirically validated** in production studio settings.

6.2.4 Summary of Validity Boundaries

The experimental findings (47.2% overall success with significant model differences; Claude Sonnet 4.5 Extended Thinking's superior calibration; ChatGPT-4o/LLaVA-13B's score hallucination) are robustly supported within tested conditions. General applicability requires validation across:

- **Storyboard diversity:** Digital, photorealistic, or abstract styles
- **Asset diversity:** Single artistic style; photorealistic, low-poly, non-biped characters
- **Character poses:** Rigged animated characters
- **Production scale:** 100+ panel workflows

These boundaries don't undermine the core contributions; in fact, they demonstrate AI feasibility for storyboard-to-3D automation and revealing systematic calibration differences across models. Instead, define the scope where we can confidently generalize and identify critical directions for follow-on validation research.

6.3 AI Backend Analysis and Deployment Recommendations

The comparative evaluation of three vision-language models shows that choosing an AI backend for iterative positioning requires balancing problem detection accuracy, computational efficiency, and cost. No single model excels across all dimensions. Instead, each has its own strengths and

weaknesses. These results suggest that optimal model selection depends on specific workflow priorities and whether human validation will supplement automated assessment.

6.3.1 Claude Sonnet 4.5 Extended Thinking's Superior Failure Detection

Claude Sonnet 4.5 Extended Thinking excels at failure detection, identifying critical composition problems that other models missed. As demonstrated in the Panel 9 case study (Section 5.4), Claude Sonnet 4.5 Extended Thinking correctly identified critical occlusion issues while ChatGPT-4o and LLaVA-13B assigned satisfied scores to identical scene geometry.

Claude Sonnet 4.5 Extended Thinking's bimodal score distribution indicates binary assessment: the model either achieves satisfaction or recognizes persistent problems warranting continued iteration. This pattern produces more specific problem identification with quantitative technical measurements, making Claude Sonnet 4.5 Extended Thinking the only backend tested that consistently flagged severe positioning failures.

This advantage comes with significant tradeoffs in iteration count, processing time, and API cost (Table 5.4). Figure 6.1 illustrates these three-dimensional tradeoffs, showing Claude Sonnet 4.5 Extended Thinking's higher costs ($0.94 per scene) against its 83.3% success rate compared to ChatGPT-4o's lower cost ($0.25) but 16.7% success rate. Despite statistically similar AI self-reported scores across all models, Claude Sonnet 4.5 Extended Thinking's superior actual success rate and reliable problem detection might justify the premium costs in quality-focused workflows where catching composition failures early prevents expensive rework.

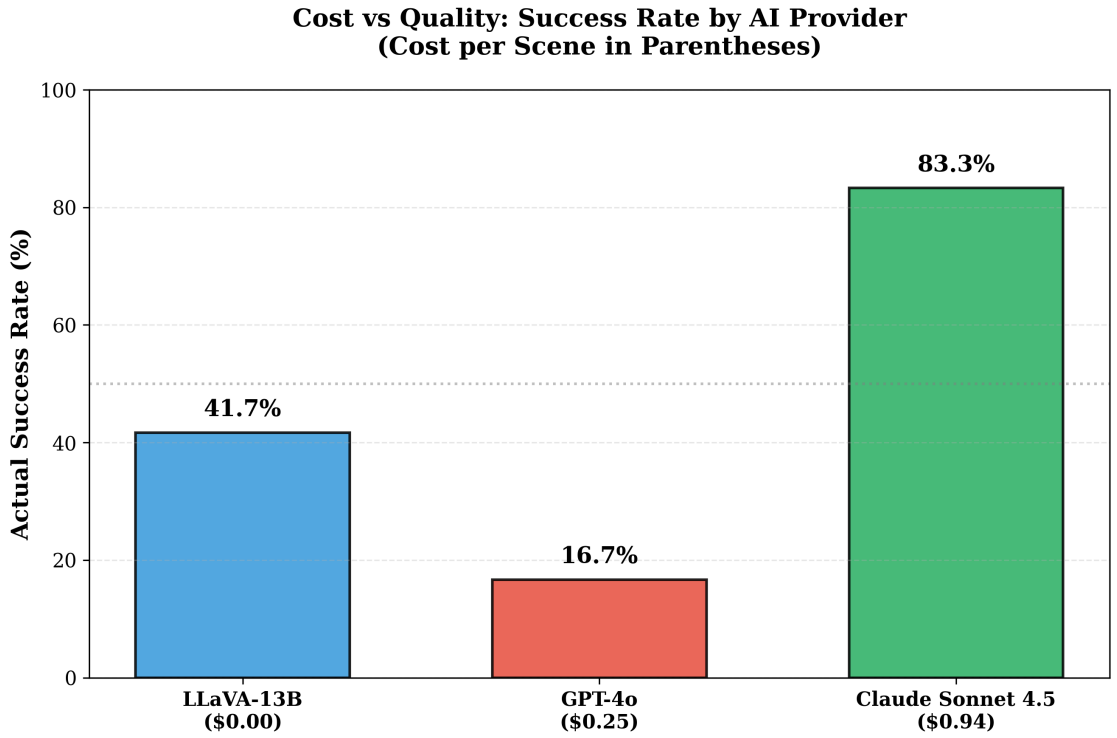


Figure 6.1. Cost vs Quality Tradeoff Across AI Backends. Three-dimensional tradeoff visualization plotting cost per scene ($0 for LLaVA-13B, $0.25 for ChatGPT-4o, $0.94 for Claude Sonnet 4.5 Extended Thinking), processing time (7.8–69.1 minutes), and expert-validated actual success rates (16.7% for ChatGPT-4o, 41.7% for LLaVA-13B, 83.3% for Claude Sonnet 4.5 Extended Thinking).

6.3.2 ChatGPT-4o and LLaVA Score Clustering and AI Score Hallucination

ChatGPT-4o and LLaVA-13B both show narrow score clustering with notably consistent distributions (Table 5.4). These models converge rapidly to the system-defined convergence threshold with high convergence rates.

As demonstrated in the Panel 9 case study (Section 5.4), both models assigned satisfied scores despite severe occlusion. Examining the assessment texts reveals generic positive language without specific technical measurements, suggesting limited sensitivity to actual quality differences.

Despite these calibration differences, both models reported AI self-assessment scores statistically indistinguishable from Claude Sonnet 4.5 Extended Thinking, though actual success rates differed dramatically (Table 5.2). ChatGPT-4o and LLaVA-13B provide substantial efficiency advantages in processing time and cost. For workflows where human artists plan to review and refine all scenes

anyway, these efficiency advantages might outweigh the lower actual success rates.

6.3.3 MVP Evaluation Insights

The cost-benefit-quality tradeoff analysis reveals model-specific characteristics worth considering for future development. While AI self-reported scores proved similar across models (ANOVA p=0.87), actual success rates, score calibration, iteration requirements, and costs differed substantially.

6.3.4 Score Calibration Implications for AI System Design

The systematic calibration differences detailed in Section 5.4 have critical implications for AI system design: self-reported match scores should not be trusted as absolute quality indicators without external validation. Future production systems require supplemental validation mechanisms (human review, objective metrics) to detect calibration failures where AI confidence exceeds actual quality.

6.4 Production Deployment Considerations

The empirical evaluation shows technical feasibility for AI-powered storyboard-to-3D conversion, demonstrating 47.2% overall expert-validated success with Claude Sonnet 4.5 Extended Thinking achieving 83.3% as the most promising backend for future production consideration. While the system achieves these success rates at MVP scale (12 panels), deployment at production scale remains unvalidated. This section proposes deployment strategies grounded in findings while showing requirements for real-world studio integration.

6.4.1 Workflow Integration Patterns

Future work could explore a couple integration patterns for production deployment. First, an AI-generated first pass with artist refinement, where non-expert storyboard artists generate 3D previsualization (7.8–69.1 min AI processing + 3–5 min artist adjustment) that would otherwise be inaccessible due to lack of Unreal Engine expertise. Second, a hybrid multi-stage pipeline

leveraging LLaVA-13B's speed (7.8 min, zero cost) for initial positioning with Claude Sonnet 4.5 Extended Thinking's failure detection (69.1 min, $0.94 cost, 83.3% success) for quality-critical validation. All patterns require validation in studio workflows through time studies with professional artists, workflow comparisons, and quality assessment.

6.4.2 Model Selection by Studio Context

Model selection depends on studio priorities and workflow constraints. Indie studios and small animation teams (1–5 developers) may favor LLaVA-13B's zero-cost local deployment (41.7% success, 7.8 iterations) despite lower success rates. Mid-size studios (10–50 employees) with dedicated technical artists may prefer ChatGPT-4o's cost-efficiency ($0.25/panel, 33.7 min) for routine positioning, freeing artists for complex work where human expertise remains essential. Quality-critical workflows (feature film production, cinematic game cutscenes) may justify Claude Sonnet 4.5 Extended Thinking's premium cost ($0.94/panel, 69.1 min) for 83.3% success with reliable failure detection demonstrated in the Panel 9 case study. These projections represent analytical predictions from MVP data requiring validation in production settings.

6.4.3 Quality Assurance Considerations

Production systems would benefit from model-specific review strategies based on empirical score distributions. Proposed review thresholds could treat low scores as uncertainty signals warranting human judgment. Additional strategies could include: statistical process control for drift detection (monitoring score distribution changes that might indicate model degradation or API updates), spot-check sampling for high-confidence cases, and human feedback loops where reviewer corrections create training data for calibration refinement.

6.4.4 Scalability Constraints

The system's computational architecture, validated at MVP scale (12 panels), might face substantial bottlenecks if scaled to production volumes (100+ panels). Sequential processing due to Unreal

Engine's single-instance limitation could create significant delays (100 panels × 33.7 min = 56.2 hours for ChatGPT-4o). Potential mitigation strategies include multi-machine deployment (distributing panels across multiple workstations for parallel processing), cloud rendering (AWS EC2 g4dn instances with NVIDIA GPUs for elastic scaling), or overnight batch processing (LLaVA-13B's 7.8-minute mean enables 12 panels in 93.6 minutes). Preliminary batch testing beyond the core 12-panel experimental validation indicate potential memory accumulation issues (scout camera actors, sequencer tracks, rendered image buffers might consume RAM without full cleanup). Implementing periodic Unreal Editor restarts every 15–20 panels, persisting progress to CSV files (Chapter 5, Data Collection Methodology), could mitigate this constraint. Cloud-based models (ChatGPT-4o, Claude Sonnet 4.5 Extended Thinking) also impose rate limits (OpenAI: 10,000 requests/minute, Anthropic: 1,000 requests/minute) and monthly cost budgets requiring API tier upgrades or hybrid local/cloud strategies for high-volume deployment. These scalability considerations show that transitioning from MVP evaluation to production deployment requires substantial infrastructure planning, cost management, and workflow planning beyond the scope of this initial research prototype.

Chapter 7:

CONCLUSION

7.1 Contributions and Key Findings

This thesis presented StoryboardTo3D, a system for converting 2D storyboard panels into 3D animated scenes within Unreal Engine through iterative visual feedback from vision-language models. The following sections amalgamate contributions and empirical findings organized by research impact area.

7.1.1 System Contribution: Minimum Viable Product for Storyboard-to-3D Positioning

Contribution. The StoryboardTo3D Unreal Engine 5.6 plugin represents the first minimum viable product system achieving automated storyboard-to-3D positioning using a 3D asset library in Unreal Engine. Expert validation revealed substantial variation in actual success rates across AI backends (Chapter 5), with Claude Sonnet 4.5 Extended Thinking demonstrating superior calibration while ChatGPT-4o and LLaVA-13B exhibited score hallucination. While prior sketch-to-3D systems focused on single-object reconstruction or text-to-3D generation, this system handles multi-character spatial reasoning. The system establishes technical feasibility at MVP scale.

7.1.2 Research Contribution: AI Score Hallucination

Contribution. Comprehensive expert validation (Section 5.4) revealed systematic AI score hallucination in spatial reasoning tasks, with dramatically different model calibration patterns despite similar self-reported confidence. This appears to be the first empirical study measuring VLM confidence calibration in iterative spatial positioning with expert validation.

7.1.3 Architectural Contribution: Multi-Angle Capture Design

Contribution. The seven-viewpoint sequential capture architecture (Section 4.3.1) provides comprehensive spatial context for AI analysis. Future work should compare single- vs multi-camera architectures using expert validation to determine whether this design improves positioning accuracy or primarily influences AI confidence reporting.

7.1.4 Comparative Analysis: Multi-Model Performance Tradeoffs

The multi-model comparison establishes that while AI self-reported scores remained statistically similar, expert validation revealed dramatically different actual success rates (Chapter 5). This demonstrates that model selection critically impacts system viability, with Claude Sonnet 4.5 Extended Thinking achieving superior calibration while ChatGPT-4o exhibited massive hallucination. The comparison reveals distinct cost-efficiency-quality tradeoffs: Claude Sonnet 4.5 Extended Thinking provides superior failure detection at premium cost, LLaVA-13B offers zero-cost local inference with moderate success, and ChatGPT-4o provides balanced middle-ground performance but critically low success rates (Chapter 6).

7.1.5 MVP Evaluation

The work provides rigorous AI evaluation methodology by explicitly measuring that automated positioning requires longer processing time than expert manual work (7.8–69.1 minutes vs 1.7 minutes per panel, Section 3.4.3). However, unattended execution (including overnight batch processing) eliminates human labor time, providing cost savings when artists are unavailable. This assessment reframes the system's value proposition from "accelerating hands-on workflows" to "enabling time-shifted automated positioning," which may democratize access for storyboard artists, indie developers, and educators who lack technical artist expertise.

7.1.6 Research Questions Answered

This research systematically addressed the four research questions (Section 1.3) through comprehensive empirical evaluation (Chapter 5). RQ1 examined convergence patterns and complexity stratification (Section 5.1.1), RQ2 established multi-model performance differences (Section 6.3), RQ3 revealed systematic calibration failures (Section 5.4), and RQ4 confirmed MVP status with deployment limitations (Section 6.2).

7.2 Practical Impact

This research has practical implications for animation studios, game developers, AI system designers, and researchers studying human-AI collaboration in creative domains.

7.2.1 Impact for Animation and Game Production

The system's primary value proposition centers on time-shifted automated positioning that may democratize access to 3D previsualization. While processing time (7.8–69.1 minutes) exceeds expert manual work (1.7 minutes, Section 3.4.3), unattended overnight execution eliminates human labor costs, providing time savings when artists are unavailable. This may enable storyboard artists, indie developers, and educators to achieve automated positioning despite lacking Unreal Engine expertise, addressing workflows previously inaccessible due to technical or financial barriers. Model-specific tradeoffs create distinct use cases: LLaVA-13B's zero-cost local inference suits indie studios and educational contexts, ChatGPT-4o's balanced cost-speed profile enables workflow reallocation for mid-size studios, and Claude Sonnet 4.5 Extended Thinking's superior failure detection justifies premium costs in quality-critical production contexts.

7.2.2 Impact for AI System Design and VLM Research

The discovery of systematic AI score hallucination with model-specific calibration patterns offers critical guidance for any AI system employing autonomous termination logic. Systems should

implement model-specific confidence thresholds (Claude Sonnet 4.5 Extended Thinking 90-95%, ChatGPT-4o/LLaVA-13B 75-80%) and supplemental validation mechanisms rather than trusting self-reported scores. These principles extend beyond storyboard positioning to medical diagnosis, autonomous systems, and content moderation—any domain where AI confidence determines workflow routing.

The 12-panel test set and multi-model comparison methodology provide a replicable benchmark for evaluating VLM spatial reasoning capabilities. The framework measures accuracy, convergence behavior, iteration efficiency, cost, and calibration patterns through comprehensive data archival (CSV files, JSON dumps, screenshots, configuration files). Addressing gaps in VLM evaluation literature which predominantly focuses on object recognition and captioning while neglecting iterative 3D spatial reasoning with self-assessment reliability.

7.2.3 Impact for Human-AI Collaboration and Creative Tool Design

The workflow integration strategies suggest design patterns for human-AI collaboration in creative domains. Confidence thresholds treating AI scores as meta-cognitive signals indicating when human expertise is required may extend to generative image editors, code completion systems, and design assistants. The acknowledgment that automated positioning requires 4–41× longer processing time than expert work, while providing human labor time savings through unattended execution, demonstrates how tools should communicate capability accurately. This positions the system as both a cost-saving tool (when run overnight) and an accessibility tool that may expand access for non-experts.

The system architecture demonstrates robustness patterns separating production tools from research prototypes: fault tolerance for AI service failures, data persistence with crash recovery, comprehensive observability through logging, and state management preventing resource leaks. These patterns offer a model for academic projects targeting real-world adoption beyond purely demonstrating conceptual feasibility.

7.2.4 Workforce and Education Implications

The finding that automated positioning requires longer processing time but can run unattended demonstrates that AI augments rather than replaces technical artist expertise. Economic impact centers on workforce transformation: technical artists may transition from routine positioning to higher-value tasks while overnight batch processing handles volume work, new artists may access 3D previsualization previously blocked by technical barriers, and small studios may gain capabilities previously requiring larger team sizes. If automated positioning becomes standard infrastructure, educational curricula might shift from teaching manual Unreal Engine operation toward AI tool selection, quality validation, workflow orchestration, and advanced technical work where human knowledge remains irreplaceable.

7.3 Future Work

This research lays groundwork for automated storyboard-to-3D positioning while revealing extensions and open questions worth investigating. Future work divides into three categories: near-term technical extensions, medium-term research directions, and long-term possibilities.

7.3.1 Near-Term Technical Extensions

These extensions address known limitations through relatively clear implementation paths:

Animated Character Support. All experiments used static T-pose characters; production workflows require rigged, posed characters. This would require: (1) pose extraction from storyboard drawings using keypoint detection (Cao et al., 2019), (2) animation retargeting to 3D skeletons via inverse kinematics, (3) AI prompt modification explaining dynamic poses versus static positioning, (4) revalidation with posed characters to confirm whether Claude Sonnet 4.5 Extended Thinking's 83.3% success rate (or the 47.2% overall rate across all three models) transfers to animated character workflows.

Research Question: Do realistic character poses improve AI positioning accuracy or degrade it?

Geometric Validation and Sanity Bounds. Preliminary testing revealed that unconstrained AI positioning occasionally produces catastrophic failures. Characters placed underground (Y < 0), objects scaled 10× normal size, or actors positioned far outside playable regions.
Deployment requires validation layers: (1) Y-position bounds (0–300 for standing, -50–50 for seated), (2) scale limits (0.5–2.0× base size), (3) collision detection for overlapping actors, (4) distance sanity checks for depth relationships. AI adjustments violating bounds are rejected with maximum 3 retry attempts before flagging for manual review.

Model-Specific Threshold Optimization. Self-assessment patterns (Section 5.4) suggest tuning convergence thresholds per model: Claude Sonnet 4.5 Extended Thinking 90–95%, ChatGPT-4o 80% (baseline), LLaVA-13B 75% with mandatory review.

Objective Quality Metrics Integration. Supplement AI self-assessment with objective metrics: SSIM (Z. Wang et al., 2004), LPIPS (Zhang et al., 2018), depth map comparison. Flag panels where objective metrics contradict AI scores (SSIM <0.7 but AI score *geq*85).

Large-Scale Evaluation (50–100 Panels). The 12-panel test set offers limited statistical power; 50–100 panel evaluation could enable: (1) precise accuracy estimates (95% CI: ±3–4% vs ±6–8%), (2) stratified sampling across artistic styles, (3) medium effect size detection (Cohen's d *geq* 0.5) at 80% power. Resources: 84 hours processing, $12.50 API costs, one week with overnight batching.

7.3.2 Medium-Term Research Directions

These directions would require more substantial investigation:

Ensemble Voting and Confidence Aggregation. Deploy multiple VLMs simultaneously, aggregating positioning recommendations and using inter-model disagreement as uncertainty signals. Approach: (1) parallel inference to ChatGPT-4o, Claude Sonnet 4.5 Extended Thinking, LLaVA-13B, (2) weighted averaging of proposed adjustments, (3) flagging for human review when models

diverge >15 points.

Research Questions: Does ensemble averaging improve accuracy over Claude Sonnet 4.5 Extended Thinking's 83.3% actual success?

Reinforcement Learning from Human Feedback (RLHF). Log adjustment pairs when human reviewers correct AI positioning (AI proposal $\rightarrow$ human correction) to create training data for calibration refinement. Collect 200–500 human correction examples per studio, train reward model mapping (storyboard, positioning, adjustment).

Domain-Specific Fine-Tuning. General-purpose VLMs could benefit from fine-tuning on domain-specific data (paired storyboard-3D examples, cinematography principles, artistic conventions). Requires 500–2000 paired examples from production projects or synthetic generation. LoRA (Hu et al., 2021) fine-tuning of ChatGPT-4o vision encoder could specialize spatial reasoning while preserving general capabilities. Expected: 5–10 percentage point accuracy improvement, particularly for complex scenes.

7.3.3 Long-Term Transformative Directions

Beyond near and medium-term extensions, several long-term research directions emerge from this work, each requiring multi-year investigation: (1) full animation pipeline integration extending beyond positioning to character performance, camera motion, temporal timing, physics simulation, and lighting design, transforming the system from positioning assistant to complete previsualization generator; (2) interactive co-creative workflows where artists and AI collaborate iteratively through sketch-based input, natural language feedback, and parameter control rather than fully automated operation; (3) generalization to other 2D-to-3D creative domains including architectural visualization, game level design, product design, and UI/UX prototyping by identifying common patterns in iterative spatial reasoning workflows; (4) fundamental confidence calibration research investigating why models exhibit categorically different self-assessment strategies, whether calibration can be improved through training interventions, and how calibration behavior affects user trust across

domains from code generation to medical diagnosis. These directions represent emerging research at the intersection of AI system design, human-computer interaction, and cognitive science, building on the principles, architectures, and evaluation methodologies established here with implications extending beyond storyboard positioning to all AI-assisted decision-making systems.

7.4 Closing Remarks

This thesis established the technical feasibility of AI-powered storyboard-to-3D positioning through comprehensive empirical evaluation. Claude Sonnet 4.5 Extended Thinking achieved 83.3% expert-validated success, demonstrating proof-of-concept feasibility for automated positioning workflows despite ChatGPT-4o and LLaVA-13B's failures (16.7% and 41.7% respectively). The StoryboardTo3D system offers a functional MVP tool for practitioners while contributing methodological standards for rigorous AI evaluation. By having transparent baseline comparisons, comprehensive expert validation, and acknowledgment of limitations rather than promotional claims.

The most significant research contribution extends beyond system functionality to fundamental understanding of vision-language model behavior. The finding of AI score hallucination reveals that VLM calibration varies dramatically across providers and cannot be trusted for autonomous quality assessment. This finding carries critical implications for any AI system employing self-termination logic, from medical diagnosis to autonomous vehicles to content moderation. The calibration insight that models have different self-assessment strategies requiring model-specific deployment adaptations exemplifies knowledge that comes from methodical evaluation.

This work contributes to a vision of AI as an enabling technology in creative industries. One that expands access to creative expression rather than replacing human expertise. By documenting that automated positioning requires longer processing time (4–41× slower) but enables unattended overnight execution that saves human labor time, the research establishes the value proposition as both cost-saving through time-shifted processing and potentially democratizing access for non-experts: storyboard artists, indie developers, educators, and solo creators who lack technical 3D skills. As vision-language models continue advancing, the principles, architectures, and evaluation

methodologies established here provide base for more AI-assisted content creation. This future should be approached with methodological integrity and human-centric design. Ensuring that AI augments human creativity rather than constraining it and enhances creative workflows rather than diminishing the value of human artistry.

BIBLIOGRAPHY

Amershi, S., Weld, D., Vorvoreanu, M., Fourney, A., Nushi, B., Collisson, P., Suh, J., Iqbal, S., Bennett, P. N., Inkpen, K., et al. (2019). Guidelines for human-AI interaction [18 design guidelines for AI augmentation based on 20+ years of HCI research]. *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems*, 1–13. https://doi.org/10.1145/3290605.3300233.

Anthropic. (2025). *Claude Sonnet 4.5: Extended thinking mode* (tech. rep.) (Vision-language model with extended reasoning capabilities. Accessed January 2025). Anthropic. https://www.anthropic.com/claude

Canemaker, J. (1999). *Paper dreams: The art and artists of disney storyboards* [Comprehensive history of Disney storyboarding and Webb Smith's innovations in the 1930s]. Hyperion Press.

Cao, Z., Hidalgo, G., Simon, T., Wei, S.-E., & Sheikh, Y. (2019). OpenPose: Realtime multi-person 2d pose estimation using part affinity fields. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *43*(1), 172–186. https://doi.org/10.1109/TPAMI.2019.2929257.

Cohen, J. (1988). *Statistical power analysis for the behavioral sciences* (2nd). Lawrence Erlbaum Associates.

Creswell, J. W., & Creswell, J. D. (2017). *Research design: Qualitative, quantitative, and mixed methods approaches* (5th). SAGE Publications.

Epic Games. (2025). Unreal Engine 5.6 [Real-time 3D creation tool. Version used in development].

Frayling, C. (1993). *Research in art and design* (Vol. 1). Royal College of Art Research Papers.

Gamma, E., Helm, R., Johnson, R., & Vlissides, J. (1994). *Design patterns: Elements of reusable object-oriented software* [The Gang of Four design patterns book]. Addison-Wesley Professional.

Geirhos, R., Rubisch, P., Michaelis, C., Bethge, M., Wichmann, F. A., & Brendel, W. (2019). ImageNet-trained CNNs are biased towards texture; increasing shape bias improves accuracy and robustness. *International Conference on Learning Representations (ICLR)*. https://doi.org/10.48550/arXiv.1811.12231.

Guo, C., Pleiss, G., Sun, Y., & Weinberger, K. Q. (2017). On calibration of modern neural networks. *International Conference on Machine Learning (ICML)*, 1321–1330. http://proceedings.mlr.press/v70/guo17a.html

Hartley, R., & Zisserman, A. (2004). *Multiple view geometry in computer vision* (2nd). Cambridge University Press.

Ho, J., Jain, A., & Abbeel, P. (2020). Denoising diffusion probabilistic models [Foundational work on diffusion models for iterative image generation]. *Advances in Neural Information Processing Systems (NeurIPS)*, *33*, 6840–6851. https://arxiv.org/abs/2006.11239

Hood, S. (2023). Draw-a-ui: AI-powered tool to convert hand-drawn wireframes to HTML [Demonstration of GPT-4 Vision interpreting hand-drawn sketches].

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

Igarashi, T., Matsuoka, S., & Tanaka, H. (1999). Teddy: A sketching interface for 3d freeform design [Seminal work on sketch-based 3D modeling; SIGGRAPH 1999 Impact Paper Award]. *Proceedings of the 26th Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH)*, 409–416. https://doi.org/10.1145/311535.311602.

Kadavath, S., Conerly, T., Askell, A., Henighan, T., Drain, D., Perez, E., Schiefer, N., Hatfield-Dodds, Z., DasSarma, N., Tran-Johnson, E., et al. (2022). Language models (mostly) know what they know [Anthropic research on self-assessment accuracy in large language models]. *arXiv preprint arXiv:2207.05221*. https://arxiv.org/abs/2207.05221

Kim, K., Joyner, D. A., Houser, J., Patel, P., & Li, Y. (2019). Analysis of previsualization tasks for animation, film and theater [Task analysis of digital previsualization workflows in film production]. *Extended Abstracts of the 2019 CHI Conference on Human Factors in Computing Systems*. https://doi.org/10.1145/3290607.3312953.

Lazar, J., Feng, J. H., & Hochheiser, H. (2017). *Research methods in human-computer interaction* (2nd) [Comprehensive HCI research methods including evaluation methodology and single vs. multi-evaluator assessment]. Morgan Kaufmann.

Li, J., Li, D., Savarese, S., & Hoi, S. (2023). BLIP-2: Bootstrapping language-image pre-training with frozen image encoders and large language models. *International Conference on Machine Learning (ICML)*. https://arxiv.org/abs/2301.12597

Li, Y., Choi, D., Chung, J., Kushman, N., Schrittwieser, J., Leblond, R., Eccles, T., Keeling, J., Gimeno, F., Dal Lago, A., et al. (2022). Competition-level code generation with AlphaCode [Iterative code generation using testing and refinement]. *Science*, *378*(6624), 1092–1097. https://doi.org/10.1126/science.abq1158.

Lin, C.-H., Gao, J., Tang, L., Takikawa, T., Zeng, X., Huang, X., Kreis, K., Fidler, S., Liu, M.-Y., & Lin, T.-Y. (2023). Magic3D: High-resolution text-to-3d content creation [Two-stage coarse-to-fine text-to-3D optimization, 2× faster than DreamFusion]. *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 300–309. https://arxiv.org/abs/2211.10440

Liu, H., Li, C., Li, Y., & Lee, Y. J. (2023). Improved Baselines with Visual Instruction Tuning. *arXiv preprint arXiv:2310.03744*. https://arxiv.org/abs/2310.03744

Liu, H., Li, C., Wu, Q., & Lee, Y. J. (2023). Visual Instruction Tuning. *Advances in Neural Information Processing Systems (NeurIPS)*. https://doi.org/10.5555/3666122.3668742.

Liu, Z.-Q., & Leung, K.-M. (2006). Script visualization (ScriptViz): A smart system that makes writing fun [Text-to-3D scene generation system for screenplay visualization]. *Soft Computing*, *10*, 34–40. https://doi.org/10.1007/s00500-005-0461-4.

Ma, M., & McKevitt, P. (2006). Virtual human animation in natural language visualisation [Natural language to 3D animation system using virtual humans]. *Artificial Intelligence Review*, *25*, 37–53. https://doi.org/10.1007/s10462-007-9042-5.

McHugh, M. L. (2012). Interrater reliability: The kappa statistic [Tutorial on Cohen's kappa for assessing inter-rater agreement and when single-evaluator approaches are acceptable]. *Biochemia Medica*, *22*(3), 276–282. https://doi.org/10.11613/BM.2012.031.

Minderer, M., Djolonga, J., Romijnders, R., Hubis, F., Zhai, X., Houlsby, N., Tran, D., & Lucic, M. (2021). Revisiting the calibration of modern neural networks. *Advances in Neural Information Processing Systems (NeurIPS)*, *34*, 15682–15694. https://proceedings.neurips.cc/paper/2021/hash/8420d359404024567b5aefda1231af24-Abstract.html

OpenAI. (2024). *GPT-4o: Multimodal flagship model* (tech. rep.) (Vision-language model with 128K context window). OpenAI. https://openai.com/index/hello-gpt-4o/

Pang, K., Li, K., Yang, Y., Zhang, H., Hospedales, T. M., Xiang, T., & Song, Y.-Z. (2019). Generalising fine-grained sketch-based image retrieval. *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. https://openaccess.thecvf.com/content_CVPR_2019/html/Pang_Generalising_Fine-Grained_Sketch-Based_Image_Retrieval_CVPR_2019_paper.html

Perlin, K. (2002). Improving noise [Improved Perlin noise algorithm for procedural content generation; widely used in games and film]. *Proceedings of the 29th Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH)*, 681–682. https://doi.org/10.1145/566570.566636.

Piccolo, S. R., & Frampton, M. B. (2016). Tools and techniques for computational reproducibility. *GigaScience*, *5*(1), 30. https://doi.org/10.1186/s13742-016-0135-4.

Poole, B., Jain, A., Barron, J. T., & Mildenhall, B. (2022). DreamFusion: Text-to-3d using 2d diffusion [Introduces Score Distillation Sampling (SDS) for text-to-3D generation using 2D diffusion priors]. *arXiv preprint arXiv:2209.14988*. https://arxiv.org/abs/2209.14988

Radford, A., Kim, J. W., Hallacy, C., Ramesh, A., Goh, G., Agarwal, S., Sastry, G., Askell, A., Mishkin, P., Clark, J., et al. (2021). Learning transferable visual models from natural language supervision. *International Conference on Machine Learning (ICML)*, 8748–8763. https://doi.org/10.5555/3546258.3546408.

Ranftl, R., Bochkovskiy, A., & Koltun, V. (2021). Vision transformers for dense prediction [Dense Prediction Transformer (DPT) for monocular depth estimation using vision transformers]. *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, 12179–12188. https://doi.org/10.1109/ICCV48922.2021.01196.

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., & Klimov, O. (2017). Proximal policy optimization algorithms [Reinforcement learning algorithm widely used in robotics for iterative policy refinement]. *arXiv preprint arXiv:1707.06347*. https://arxiv.org/abs/1707.06347

Shadish, W. R., Cook, T. D., & Campbell, D. T. (2002). *Experimental and quasi-experimental designs for generalized causal inference* [Authoritative text on validity threats and research design including single-evaluator considerations]. Houghton Mifflin.

Stodden, V., McNutt, M., Bailey, D. H., Deelman, E., Gil, Y., Hanson, B., Heroux, M. A., Ioannidis, J. P., & Taufer, M. (2016). Enhancing reproducibility for computational methods. *Science*, *354*(6317), 1240–1241. https://doi.org/10.1126/science.aah6168.

Wang, C., Miguel Buenaposada, J., Zhu, R., & Lucey, S. (2018). Learning depth from monocular videos using direct methods. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2022–2030. https://doi.org/10.1109/CVPR.2018.00216.

Wang, S., Leroy, V., Cabon, Y., Chidlovskii, B., & Revaud, J. (2024). DUSt3R: Geometric 3d vision made easy. *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 20697–20709. https://arxiv.org/abs/2312.14132

Wang, Z., Bovik, A. C., Sheikh, H. R., & Simoncelli, E. P. (2004). Image quality assessment: From error visibility to structural similarity. *IEEE Transactions on Image Processing*, *13*(4), 600–612. https://doi.org/10.1109/TIP.2003.819861.

Yang, L., Kang, B., Huang, Z., Zhao, Z., Xu, X., Feng, J., & Zhao, H. (2024). Depth anything v2 [Foundation model for monocular depth estimation with improved accuracy and efficiency through synthetic data training. arXiv preprint]. *Advances in Neural Information Processing Systems (NeurIPS)*. https://arxiv.org/abs/2406.09414

Yelamarthi, S. K., Reddy, S. K., Mishra, A., & Mittal, A. (2018). A zero-shot framework for sketch based image retrieval. *Proceedings of the European Conference on Computer Vision (ECCV)*, 300–317. https://doi.org/10.1007/978-3-030-01225-0_19.

Zell, E., Levy, B., & Kapadia, M. (2021). A survey on deep learning for skeleton-based human animation [Comprehensive survey of deep learning methods for motion synthesis, character control, and motion editing]. *Computer Graphics Forum*, *40*(6), 122–157. https://doi.org/10.1111/cgf.14426.

Zhang, R., Isola, P., Efros, A. A., Shechtman, E., & Wang, O. (2018). The unreasonable effectiveness of deep features as a perceptual metric. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 586–595. https://arxiv.org/abs/1801.03924

Zimmerman, J., Forlizzi, J., & Evenson, S. (2007). Research through design as a method for interaction design research in HCI. *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI)*, 493–502. https://doi.org/10.1145/1240624.1240704.

Appendix A:

CODE LISTINGS

This appendix provides complete implementations of the key data collection and metrics tracking algorithms discussed in Chapter 5. These code listings are provided for reproducibility and to enable future researchers to replicate the experimental methodology.

Code Simplification Note: The listings in this appendix are simplified for academic clarity while preserving all functional logic. Debug logging, emoji-enhanced console output, and extensive defensive error handling have been removed to focus on core algorithms. Complete production implementations with full error handling are available in the accompanying software repository.

A.1 AI Provider Auto-Detection

Listing 5 implements runtime provider detection for metric routing, as discussed in Section 3.5.

Listing 5. AI Provider Auto-Detection for Metric Routing. This introspection-based algorithm achieves 100% correct routing in validation testing (verified by manually inspecting logged provider names against CSV file assignments), eliminating human error in 240+ experimental trials.

```
def detect_current_model(self, ai_client) -> str:
    """
    Auto-detect which AI model is currently active for data routing

    Args:
        ai_client: The AI client object with provider/model attributes

    Returns:
        Model key: 'gpt4o', 'llava', 'sonnet', or 'groundtruth'
    """
    if not ai_client:
        return 'groundtruth'  # No AI client = manual positioning

    # Inspect provider type (set during client initialization)
    provider = getattr(ai_client, 'provider', '').lower()
    model_name = getattr(ai_client, 'model', '').lower()
```

```
    # OpenAI / ChatGPT-4o
    if 'openai' in provider or 'gpt-4o' in model_name or 'gpt4o' in model_name:
        return 'gpt4o'

    # Anthropic Claude / Sonnet (handles Claude 3.5, 4, 4.5)
    if 'anthropic' in provider or 'claude' in provider or 'sonnet' in
        model_name:
        return 'sonnet'

    # LLaVA (local Ollama model)
    if 'llava' in provider or 'llava' in model_name:
        return 'llava'

    # Default to ground truth (manual positioning)
    return 'groundtruth'

def get_scene_number_from_panel(self, panel_number: int) -> str:
    """
    Convert panel number to standardized scene ID for cross-model comparison

    Args:
        panel_number: Panel number (1-12 from standardized dataset)

    Returns:
        Scene ID like 'Storyboard_01' for consistent CSV key matching
    """
    if 1 <= panel_number <= 12:
        return f"Storyboard_{panel_number:02d}"
    else:
        # Extra panels (beyond standard 12) use timestamp for uniqueness
        from datetime import datetime
        return f"Storyboard_Extra_{datetime.now().strftime('%H%M%S')}"
```

A.2 Iteration-Level Metrics Capture

Listing 6 captures iteration-level metrics for convergence analysis, as discussed in Section 3.5.

Listing 6. Iteration-Level Metrics Capture (Core Algorithm). This function aggregates data from multiple sources (AI response, cost tracking, timing, hyperparameters) into a unified iteration record, generating 6,720 data points (7 features × 20 iterations × 48 scenes) for statistical analysis. Production implementation includes additional camera position extraction and debug logging infrastructure (omitted for clarity).

```
def _record_iteration_metrics(self):
    """
```

```
    Record comprehensive metrics for current iteration to enable convergence
        analysis

    Captures:
    - Match score (AI-reported similarity to target storyboard)
    - Adjustments applied (positioning deltas or absolute coordinates)
    - Camera modifications (boolean flag)
    - API cost (USD)
    - Positioning mode (absolute vs. relative)
    - AI temperature parameter
    - Iteration duration (seconds)
    """
    if not self.metrics_tracker:
        return  # Metrics disabled

    from datetime import datetime

    # Calculate iteration duration
    iteration_time = 0.0
    if hasattr(self.metrics_tracker, 'current_iteration_start'):
        duration = datetime.now() - self.metrics_tracker.current_iteration_start
        iteration_time = duration.total_seconds()

    # Extract match score from AI response
    match_score = self.last_match_score if self.last_match_score is not None
        else 0.0

    # Extract adjustments from last AI analysis
    adjustments_applied = {}
    if self.last_ai_analysis and 'adjustments' in self.last_ai_analysis:
        for adj in self.last_ai_analysis['adjustments']:
            actor_name = adj.get('actor', 'Unknown')
            if adj.get('type') == 'move' and 'position' in adj:
                pos = adj['position']
                adjustments_applied[actor_name] = {
                    'X': pos.get('x', 0),
                    'Y': pos.get('y', 0),
                    'Z': pos.get('z', 0)
                }
            elif adj.get('type') == 'rotate' and 'rotation' in adj:
                rot = adj['rotation']
                adjustments_applied[actor_name] = {
                    'Pitch': rot.get('pitch', 0),
                    'Yaw': rot.get('yaw', 0),
                    'Roll': rot.get('roll', 0)
                }

    # Determine if camera was adjusted this iteration
    camera_adjusted = False
```

```
    if self.last_ai_analysis and 'adjustments' in self.last_ai_analysis:
        camera_adjusted = any('camera' in adj.get('actor', '').lower()
                              for adj in self.last_ai_analysis['adjustments'])

    # Get API cost from AI client
    api_cost = 0.0
    if hasattr(self, 'ai_client') and self.ai_client:
        api_cost = getattr(self.ai_client, 'last_cost', 0.0)

    # Record to metrics tracker
    self.metrics_tracker.record_iteration(
        match_score=match_score,
        adjustments_applied=adjustments_applied,
        camera_adjusted=camera_adjusted,
        cost=api_cost,
        positioning_mode='absolute' if self.use_absolute_positioning else
            'relative',
        temperature=getattr(self.ai_client, 'temperature', 0.7) if hasattr(self,
            'ai_client') else 0.7,
        iteration_time=iteration_time
    )

    unreal.log(f"Metrics recorded: Score={match_score:.1f}, "
               f"Cost=${api_cost:.4f}, Time={iteration_time:.1f}s")
```

A.3 Multi-Model CSV Export Pipeline

Listing 7 exports metrics to per-model CSV files, as discussed in Section 3.5.

Listing 7. Multi-Model CSV Export Pipeline. This method writes per-model comparison CSV files, updating specific scene rows while preserving other data. The method handles both updating existing scenes and adding new scene entries dynamically.

```
def update_model_csv(self, model_key: str, scene_id: str, metrics: Dict[str,
    Any]):
    """
    Update a specific model's CSV with metrics for a scene

    Args:
        model_key: 'gpt4o', 'llava', 'sonnet', or 'groundtruth'
        scene_id: Scene identifier (e.g., 'Storyboard_01')
        metrics: Dictionary with all metrics
    """
    if model_key not in self.MODELS:
        print(f"Unknown model key: {model_key}")
        return
```

```
    model_name = self.MODELS[model_key]
    csv_file = self.output_dir / f"{model_name}_comparison.csv"

    if not csv_file.exists():
        print(f"CSV file not found: {csv_file}")
        return

    # Read existing CSV
    rows = []
    with open(csv_file, 'r', newline='') as f:
        reader = csv.DictReader(f)
        rows = list(reader)

    # Find and update the matching scene row
    scene_found = False
    for row in rows:
        if row['scene_id'] == scene_id:
            # Update with new metrics
            row['initial_accuracy'] = metrics.get('initial_accuracy', '')
            row['final_accuracy'] = metrics.get('final_accuracy', '')
            row['improvement'] = metrics.get('improvement', '')
            row['iterations'] = metrics.get('total_iterations', '')
            row['converged'] = 'Yes' if metrics.get('converged') else 'No'
            row['convergence_iteration'] = metrics.get('convergence_iteration',
                '')
            row['total_time_sec'] = f"{metrics.get('total_time_seconds',
                0):.1f}"
            row['total_cost'] = f"${metrics.get('total_cost', 0):.4f}"
            row['avg_cost_per_iteration'] =
                f"${metrics.get('avg_cost_per_iteration', 0):.4f}"
            row['monotonic_improvement'] = 'Yes' if
                metrics.get('monotonic_improvement') else 'No'
            row['oscillating'] = 'Yes' if metrics.get('oscillating') else 'No'
            row['timestamp'] = datetime.now().strftime('%Y-%m-%d %H:%M:%S')
            scene_found = True
            break

    # If scene not found, add as new row
    if not scene_found:
        new_row = {
            'scene_id': scene_id,
            'initial_accuracy': metrics.get('initial_accuracy', ''),
            'final_accuracy': metrics.get('final_accuracy', ''),
            'improvement': metrics.get('improvement', ''),
            'iterations': metrics.get('total_iterations', ''),
            'converged': 'Yes' if metrics.get('converged') else 'No',
            'convergence_iteration': metrics.get('convergence_iteration', ''),
            'total_time_sec': f"{metrics.get('total_time_seconds', 0):.1f}",
```

```
            'total_cost': f"${metrics.get('total_cost', 0):.4f}",
            'avg_cost_per_iteration': f"${metrics.get('avg_cost_per_iteration',
                0):.4f}",
            'monotonic_improvement': 'Yes' if
                metrics.get('monotonic_improvement') else 'No',
            'oscillating': 'Yes' if metrics.get('oscillating') else 'No',
            'timestamp': datetime.now().strftime('%Y-%m-%d %H:%M:%S')
        }
        rows.append(new_row)

    # Write back to CSV
    with open(csv_file, 'w', newline='') as f:
        fieldnames = [
            'scene_id', 'initial_accuracy', 'final_accuracy', 'improvement',
            'iterations', 'converged', 'convergence_iteration',
            'total_time_sec', 'total_cost', 'avg_cost_per_iteration',
            'monotonic_improvement', 'oscillating', 'timestamp'
        ]
        writer = csv.DictWriter(f, fieldnames=fieldnames)
        writer.writeheader()
        writer.writerows(rows)

    print(f"Updated: {model_name}_comparison.csv")
    print(f"   Scene: {scene_id}")
    print(f"   Final Accuracy: {metrics.get('final_accuracy', 'N/A')}%")
    print(f"   Iterations: {metrics.get('total_iterations', 'N/A')}")
```

A.4 Iterative Positioning Main Loop

Listing 8 implements the main refinement cycle with bug fixes for scout camera accumulation and stale sequence evaluation, as discussed in Section 4.2.1 (Chapter 4).

Listing 8. Iterative Positioning Main Loop. This method starts a single refinement cycle and implements bug fixes for scout camera accumulation. The cleanup routines address two critical bugs documented during development: scout camera accumulation causing viewport locking, and stale sequence evaluation producing incorrect captures.

```
def _start_next_iteration(self):
    """Start the next capture iteration with comprehensive state management"""

    # Guard: Check if workflow was cancelled
    if not self.capture_workflow_active:
        unreal.log("Warning: Capture workflow cancelled - skipping iteration")
        return
```

```
        unreal.log("\n" + "="*70)
        unreal.log(f"Starting Iteration
            {self.current_iteration}/{self.max_iterations}")
        unreal.log("="*70)

        # Verify sequence is still open (may be closed between iterations)
        if self.active_panel and self.active_panel.get('sequence_path'):
            current_seq = unreal.LevelSequenceEditorBlueprintLibrary.
                get_current_level_sequence()

            if not current_seq:
                unreal.log_warning("Sequence was closed - reopening...")
                sequence_path = self.active_panel.get('sequence_path')
                sequence_asset = unreal.load_asset(sequence_path)
                if sequence_asset:
                    unreal.LevelSequenceEditorBlueprintLibrary.open_level_sequence
                        (sequence_asset)
                    unreal.log(f"Reopened sequence: {sequence_asset.get_name()}")

        # Track iteration start time for metrics
        if self.metrics_tracker:
            from datetime import datetime
            self.metrics_tracker.current_iteration_start = datetime.now()

        # ============================================================
        # CRITICAL FIX #1: Clean up scout cameras from previous iteration
        # ============================================================
        unreal.log("Cleaning up scout cameras from previous iteration...")
        try:
            # Eject from any piloted camera first
            level_editor_subsystem =
                unreal.get_editor_subsystem(unreal.LevelEditorSubsystem)
            level_editor_subsystem.eject_pilot_level_actor()

            # Delete all scout cameras (prevents viewport locking)
            subsystem = unreal.get_editor_subsystem(unreal.EditorActorSubsystem)
            all_actors = subsystem.get_all_level_actors()
            scouts_deleted = 0
            for actor in all_actors:
                if "AI_Scout" in actor.get_actor_label():
                    subsystem.destroy_actor(actor)
                    scouts_deleted += 1

            if scouts_deleted > 0:
                unreal.log(f"   Deleted {scouts_deleted} scout camera(s)")
        except Exception as cleanup_err:
            unreal.log_warning(f"   Scout cleanup warning: {cleanup_err}")

        # ============================================================
```

```
    # CRITICAL FIX #2: Force sequence evaluation before captures
    # ============================================================
    unreal.log("Forcing sequence evaluation before captures...")
    try:
        import time
        active_seq = unreal.LevelSequenceEditorBlueprintLibrary.
            get_current_level_sequence()
        if active_seq:
            # Bind camera cuts to viewport (makes sequence active)
            unreal.LevelSequenceEditorBlueprintLibrary.
                set_lock_camera_cut_to_viewport(True)

            # Set frame to 0 and force evaluation
            unreal.LevelSequenceEditorBlueprintLibrary.set_current_time(0.0)
            unreal.LevelSequenceEditorBlueprintLibrary.
                refresh_current_level_sequence()
            time.sleep(0.2)  # Allow viewport to update

            # Unbind so scout camera can pilot freely
            unreal.LevelSequenceEditorBlueprintLibrary.
                set_lock_camera_cut_to_viewport(False)
            unreal.log("   Sequence evaluated at frame 0, viewport updated")
    except Exception as e:
        unreal.log_warning(f"   Could not force sequence evaluation: {e}")

    # Re-enable workflow flag for this iteration's callbacks
    self.capture_workflow_active = True

    # Step 1: Pilot to Scout Camera
    unreal.log("Step 1: Piloting to Scout Camera...")
    self.test_pilot_to_scout()

    # Step 2: Initiate multi-angle capture sequence
    unreal.log("Step 2: Queuing Front View capture...")
    front_success = self.test_capture_front()
    if not front_success:
        unreal.log_error("Front capture failed to queue!")
        return

    # Schedule remaining captures with 15s delays
    from PySide6.QtCore import QTimer
    QTimer.singleShot(15000, self._capture_right_delayed)
```

A.5 AI Adjustment to Sequence Keyframes

Listing 9 translates natural language positioning feedback into Unreal Engine keyframe operations, as discussed in Section 4.4.1 (Chapter 4).

Listing 9. AI Adjustment to Sequence Keyframes. This pipeline validates sequence bindings, logs planned adjustments for debugging, delegates to `SceneAdjuster` for keyframe manipulation, and forces viewport refresh to prevent oscillation. The pre-flight checks prevent silent failures when the AI suggests moving non-existent actors.

```
def _apply_ai_adjustments(self, analysis):
    """
    Apply AI positioning recommendations to sequence keyframes automatically

    Args:
        analysis: Parsed analysis dict containing 'adjustments' list
    """
    from core.scene_adjuster import SceneAdjuster

    # Get the sequence path from active panel
    sequence_path = None
    if self.active_panel and 'sequence_path' in self.active_panel:
        sequence_path = self.active_panel['sequence_path']

    # Load the sequence asset
    sequence_asset = unreal.load_asset(sequence_path)
    if not sequence_asset:
        unreal.log_error(f"Could not load sequence: {sequence_path}")
        return

    # Log positioning mode
    mode_str = "ABSOLUTE" if self.use_absolute_positioning else "RELATIVE"
    unreal.log(f"Positioning Mode: {mode_str}")

    # Create scene adjuster with positioning mode
    adjuster = SceneAdjuster(
        sequence_asset=sequence_asset,
        use_absolute_positioning=self.use_absolute_positioning
    )

    # Get adjustments from AI analysis
    adjustments = analysis.get('adjustments', [])

    # ============================================================
    # PRE-FLIGHT CHECK: Verify sequence bindings exist
    # ============================================================
    bindings = sequence_asset.get_bindings()
```

```
    binding_names = [str(b.get_display_name()) for b in bindings]
    unreal.log(f"   Found {len(binding_names)} bindings in sequence:
        {binding_names}")

    # Check if all actors mentioned in adjustments have bindings
    missing_bindings = []
    for adj in adjustments:
        actor_name = adj.get('actor', 'Unknown')
        # Check if actor name appears in any binding (case-insensitive partial
            match)
        if not any(actor_name.lower() in b.lower() for b in binding_names):
            missing_bindings.append(actor_name)

    if missing_bindings:
        unreal.log_error(f"   FATAL: {len(missing_bindings)} actors NOT in
            sequence!")
        for actor in missing_bindings:
            unreal.log_error(f"      - '{actor}' cannot be adjusted (not
                spawned)")
        return

    # ============================================================
    # LOG ADJUSTMENT COMMANDS (pre-execution validation)
    # ============================================================
    unreal.log(f"\nADJUSTMENT COMMANDS TO BE APPLIED")
    unreal.log(f"   Mode: {'ABSOLUTE' if self.use_absolute_positioning else
        'RELATIVE'}")
    unreal.log(f"   Total adjustments: {len(adjustments)}\n")

    expected_positions = {}
    for i, adj in enumerate(adjustments, 1):
        actor = adj.get('actor', 'UNKNOWN')
        adj_type = adj.get('type', 'unknown')

        if adj_type == 'move' and adj.get('position'):
            pos = adj['position']
            expected_positions[actor] = pos
            unreal.log(f"   [{i}] {actor}: MOVE to X={pos.get('x', 0):.1f}, "
                       f"Y={pos.get('y', 0):.1f}, Z={pos.get('z', 0):.1f}")
            if adj.get('reason'):
                unreal.log(f"       Reason: {adj['reason'][:80]}")

    # Apply adjustments via SceneAdjuster
    results = adjuster.apply_all_adjustments(analysis)

    unreal.log(f"\nADJUSTMENT RESULTS")
    unreal.log(f"   Total: {results.get('total', 0)}")
    unreal.log(f"   Success: {results.get('success', 0)}")
    unreal.log(f"   Failed: {results.get('failed', 0)}")
```

```
    # ============================================================
    # CRITICAL: Force viewport update after applying keyframes
    # ============================================================
    unreal.log("   Forcing viewport refresh after adjustments...")
    try:
        import time
        # Force sequencer to evaluate at frame 0
        unreal.LevelSequenceEditorBlueprintLibrary.set_current_time(0.0)
        unreal.LevelSequenceEditorBlueprintLibrary.play()
        time.sleep(0.1)  # Brief play to force evaluation
        unreal.LevelSequenceEditorBlueprintLibrary.pause()
        unreal.LevelSequenceEditorBlueprintLibrary.set_current_time(0.0)

        # Give viewport time to refresh
        time.sleep(0.2)
        unreal.log("   Viewport refreshed - positions now visible")
    except Exception as e:
        unreal.log_warning(f"   Could not force refresh: {e}")
```

Appendix B:

USE OF ARTIFICIAL INTELLIGENCE IN RESEARCH PROCESS

Use of Artificial Intelligence in Research Process

This appendix documents the use of artificial intelligence (AI) tools during the development of this thesis, in compliance with Drexel University's academic integrity policies.

AI Tools Employed

- **Grammarly**: Used for grammar correction and editorial suggestions to improve academic writing clarity and tone.
- **ChatGPT-4o**: Used for limited rewording assistance to improve clarity of selected passages. No original research ideas, data interpretation, methodological design, or analytical conclusions were generated by this tool.
- **Claude Sonnet 4.5 Extended Thinking**: Employed for debugging assistance and code comment generation during plugin development. All core algorithms, system architecture, and implementation logic were independently designed and authored by the researcher.

Scope and Limitations of AI Use

AI tools were used strictly in a supportive capacity for editorial refinement and technical debugging. All tools operated under direct author supervision. Specifically:

- AI tools did not formulate research questions or hypotheses
- AI tools did not design the experimental methodology

- AI tools did not interpret research results or draw conclusions
- AI tools did not generate the theoretical framework or literature analysis
- All AI-assisted content was thoroughly reviewed, edited, and verified by the author

Research Subjects vs. Assistive Tools

It should be noted that ChatGPT-4o, Claude Sonnet 4.5 Extended Thinking, and LLaVA-13B were the *subjects* of this research (evaluated for storyboard positioning capabilities) and are distinct from the assistive AI tools listed above that supported thesis writing and development.