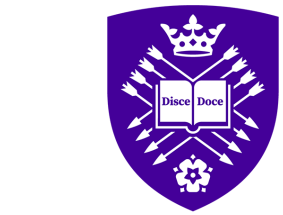


University of Sheffield

Beyond Simple Prompts: Controllable and Interactive 3D Human Motion Generation in Unity

Jack Smith

Supervisor: Zhixiang Chen

A report submitted in fulfilment of the requirements for the degree of BSc in Computer Science

in the

School of Computer Science

May 13, 2026

Declaration

All sentences or passages quoted in this report from other people's work have been specifically acknowledged by clear cross-referencing to author, work and page(s). Any illustrations that are not the work of the author of this report have been used with the explicit permission of the originator and are specifically acknowledged. I understand that failure to do this amounts to plagiarism and will be considered grounds for failure in this project and the degree examination as a whole.

Name: Jack Smith

Signature: JSmith

Date: May 13, 2026

Acknowledgements

I would like to thank my supervisor, Zhixiang Chen, for his guidance and feedback throughout this project. I am also grateful to the participants who took part in the user study and made the evaluation possible. I would also like to thank the Unity developers who granted me access to the Unity AI beta build used in the comparative study; without that support, the evaluation against Unity AI could not have been conducted in the form reported here. Finally, I should acknowledge my coffee machine, which may need to be replaced after this dissertation.

Abstract

Text-to-motion generation has matured to the point where research models can reasonably contribute to real animation work; yet, getting from a research system to a usable clip inside a real scene is still awkward. The current commercial options are either cloud services tied to an external account or standalone web tools that force repeated export and re-import just to check a motion against the target rig. At submission, there is still no Unity-native tool that is locally hostable, non-commercial, or easy to extend with multiple models.

This dissertation presents MotionGen, a Unity editor plugin with a local Python backend connected through gRPC. It integrates T2M-GPT, MoMask, and MDM; supports single and batched generation, text-prompted segment regeneration, two-clip inbetweening, and multi-segment composition; and writes the results back as humanoid `AnimationClip` assets inside the Unity project. A locally hosted Gemma 4 planner can split a complex prompt into a JSON plan with per-segment model choices, durations, transitions, and scene-anchor references. An exploratory variant ranker scores clips using foot skating, jerk, root drift, and ground penetration.

Evaluation comprised four studies: a within-subjects user study against Unity AI's Muse Animate, an internal model-fit study across seven prompt families with a blind reviewer pass, a local-planner latency benchmark, and a calibration test of the variant ranker. MotionGen led Unity AI on all four headline workflow measures: SUS rose from 54.72 to 87.75, raw NASA-TLX more than halved, average task time fell, and ease ratings improved. Mechanism data showed less setup and recovery friction, the model-fit study supported family-aware routing, the planner benchmark showed that prompt caching halves steady-state latency, and the ranker replay showed that the current heuristic over-rewards stillness.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Background

From game development to robotics, animating human motion is a common requirement. Doing it well still usually means keyframing by hand or incurring the costs of motion capture, which puts time, specialist skill, and equipment in the way. Recent advances in text-to-motion models have changed that. Just a prompt can now produce a plausible animation. The problem is that a plausible clip is not the same thing as a usable one.

Unity is a sensible place to tackle that gap. It is already used in games, XR, simulation, education, and other settings where humanoid motion matters, and it is the same environment where the final animation has to be previewed, adjusted, and applied. That makes it much more useful than a standalone demo that stops at generation.

Tools such as Unity Technologies (2026) and DeepMotion Inc. (2024) can already generate motion, but the awkward part starts after the first result is produced: comparing variants, previewing them on the right rig, fixing what went wrong, and checking scene fit. This dissertation focuses on bridging that gap. The goal is a Unity-based workflow where motion can be easily generated, inspected, refined, and used in the same editor session.

The core authoring loop developed from this idea is summarised later in Figure 3.2: prompt, generate variants, compare them, preview the chosen motion in the scene, and apply it without leaving the editor.

1.2 Project Aim and Objectives

The aim of this project is to create a Unity plugin that consolidates a useful and capable workflow for generating, editing, and evaluating human motion. This is an editor-first system, with a Unity frontend and a local python backend connected through gRPC. This system has the baseline ability to generate one or more motion variants from a text prompt using selectable models in the backend, which are then converted to humanoid AnimationClips that can be used directly in Unity.

The workflow is intended to allow users to view and compare generations, as well as to

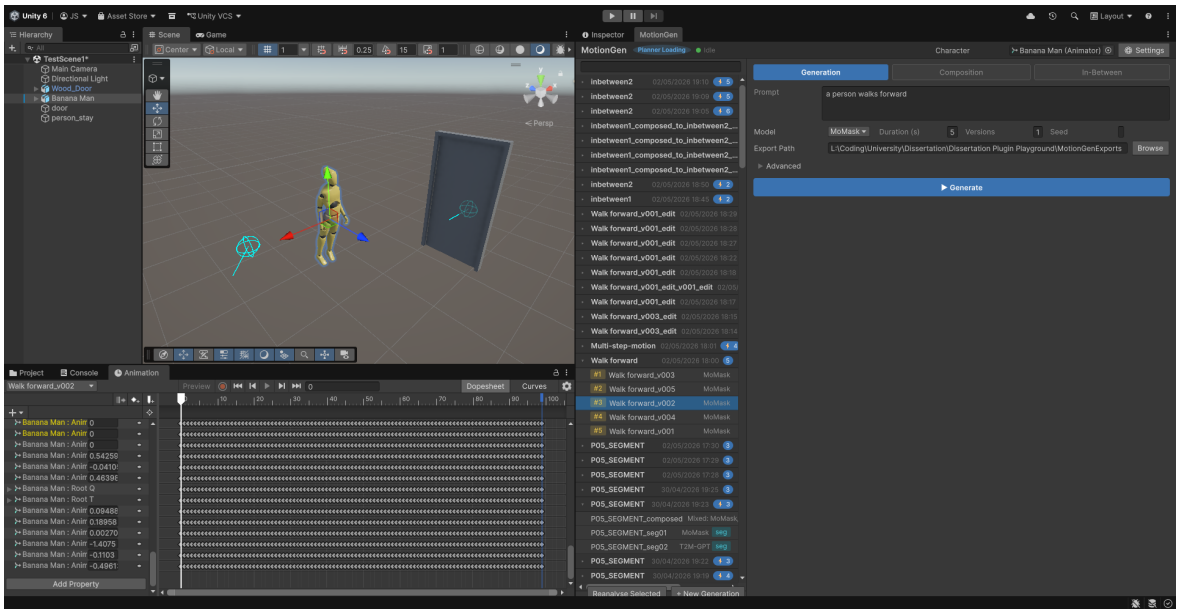


Figure 1.1: Overview of the MotionGen Unity plugin workflow.

edit them using a variety of tools that give them more control over the output. In addition to this, experimental features are explored, such as decomposing more complex prompts into multiple motion segments and ranking generated variations using multiple metrics in an attempt to surface stronger results sooner.

The objectives of the project are, therefore:

- To integrate text-to-motion generation into a practical Unity workflow;
- To support the generation of multiple motion variants and conversion into usable humanoid AnimationClips;
- To provide tools for comparison, editing, refinement, and spatial adjustment of generated motion;
- To explore whether complex prompts can be handled more effectively through decomposition into segments;
- To explore whether generated variants can be ranked in a useful way to improve the workflow;
- To reduce setup and deployment friction by supporting automated local backend setup and on-demand installation of required motion-generation and language models;
- To evaluate whether this workflow improves usability, control, and task completion compared with existing motion generation tools;

1.3 Scope and Constraints

This project consists of a user-facing Unity plugin that connects to a local Python backend through a gRPC connection, enabling prompt-based generation that is converted into a Unity-compatible format. As an interactive authoring workflow, the plugin includes motion comparison, editing, and refinement features such as regeneration, post-processing, spatial constraints, and exploratory support for prompt decomposition and generation ranking. As this tool focuses on bridging the user experience gap between motion generation models and practical use, a user study is included to adequately evaluate it.

This dissertation does not create or invent any generative models; instead, it enables pre-existing models to be used in real workflows. It also does not attempt to solve general-purpose re-targeting for all rig types, as humanoid rigs are the main focus of this project and are the most common target in current motion-generation applications. Although the backend could be deployed in the cloud, in this project it will remain local in order to support a self-contained and user-constructed workflow and reduce dependence on external services. Full-scene-aware generation is also beyond the scope of this dissertation, as it would require substantially more time and technical complexity.

As development is being carried out on a Windows machine, this project is Windows-first, with support for other platforms left a future consideration. The contract between the frontend and backend will support a humanoid BVH format, generated from a selection of dynamically downloadable models. Although the project aims to be usable on consumer-grade hardware, local model compatibility, download overhead, and inference performance are still practical constraints. The experimental features, such as prompt decomposition and generation ranking, are also treated as exploratory parts of the workflow rather than fully solved problems.

1.4 Contributions

This dissertation makes five main contributions to the problem of integrating text-to-motion generation into a practical animation workflow.

- It presents an editor-first workflow for generating, comparing, refining, and evaluating human motion directly within Unity, bridging the gap between motion generation models and the environment in which the generated motion is previewed and used.
- It implements a local end to-end system that connects a Unity frontend to a Python backend through gRPC, enabling prompt-based generation with selectable backend models and the conversion of generated outputs into usable humanoid animation assets.
- It introduces a prompt decomposition mechanism for handling complex prompts by breaking them down using a language model, generating each segment with the best model for that motion, and merging the segments using an inbetweening model.

- It explores an experimental ranking mechanism for generated motion variants, using motion-quality metrics to analyse and prioritise results in an attempt to surface stronger candidate motions earlier in the workflow.
- It provides a comparative evaluation of the proposed workflow through a dedicated user-study framework, allowing for the evaluation of usability, control, and task performance relative to existing motion-generation tools.

1.5 Overview of the Dissertation

The remainder of this dissertation is organised as follows.

Chapter 2 (Background and Literature Review) surveys the four bands of work that the project draws on: the generative text-to-motion models that the backend integrates (T2M-GPT, MoMask, MDM, and the rejected DART), the controllability and composition techniques that make those models usable beyond single-action prompts, the datasets and evaluation conventions that frame motion quality, and the local-inference infrastructure and commercial tools that define the practical landscape. It closes by identifying the workflow gap that motivates the project: no current tool combines local hosting, Unity editor integration, multi-model routing, LLM-driven composition with scene anchors, and post-generation refinement.

Chapter 3 (Requirements and System Design) translates that gap into design drivers and a set of functional and non-functional requirements covering generation, comparison, editing, scene fitting, decomposition, and ranking. It then sets out the system architecture and key design decisions that follow from those requirements.

Chapter 4 (Implementation) describes the resulting artefact in the order in which problems were solved rather than the codebase's file structure. It covers the gRPC service boundary and the iterations that led to BVH as the interchange format, the editor-window frontend, motion reconstruction onto Unity's humanoid avatar, the root-stability fix that replaced an earlier path-editing pipeline, the editing and spatial tools, the composition planner driven by a locally hosted Gemma 4 model, the integrated motion library and variant ranker, and the cross-cutting safeguards that keep the editor responsive.

Chapter 5 (Evaluation) sets out the four evidence streams used to assess the artefact: a within-subjects comparative user study against Unity AI's Muse Animate, an internal model-fit study across the three integrated backends, a local-planner latency benchmark, and a calibration test of the variant-ranking heuristic.

Chapter 6 (Results and Discussion) reports the outcomes of those four streams and interprets them as a single argument about the workflow contribution rather than as isolated measurements, including the cases in which the supporting evidence contradicts a design decision.

Chapter 7 (Conclusions) summarises the contributions in light of the evaluation, reflects on the limits of the cohort and the heuristic ranker, and outlines the future work that the manifest-driven, model-agnostic design was deliberately engineered to support.

Chapter 2

Background and Literature Review

2.1 Framing the Gap

Text-to-motion generation has matured to the point where its use for practical animation tasks is plausible; yet, research models, generation tooling, and animator workflows have evolved on largely separate paths. This review covers four bands relevant to the system described in Chapters 3 and 4: (a) the generative models that the project integrates, (b) the controllability and composition techniques that make those models usable beyond single-action one-shot generations, (c) the datasets and evaluation methodology that frame motion quality, and (d) the local infrastructure and commercial tools that define the practical context. The chapter closes by stating the gap that motivates the dissertation: no current commercial or open tool combines local hosting, editor-native integration, multi-model routing, LLM-driven composition with scene anchors, and post-generation refinement.

2.2 Text-to-Motion Generation

The system routes text prompts to one of three backend models: T2M-GPT, MoMask, and MDM, and it was designed to accommodate a fourth (DART). This section establishes what each model contributes and why three rather than one.

2.2.1 Discrete tokenisation: T2M-GPT

T2M-GPT Zhang et al. (2023) pairs a VQ-VAE with an autoregressive transformer conditioned on CLIP text features. In practice, the VQ-VAE compresses short windows of continuous motion into a discrete vocabulary of "motion tokens", and the transformer predicts those tokens one after another from left to right. CLIP supplies the text embedding that tells the model what the prompt means in semantic terms. On HumanML3D Guo et al. (2022), it reported an FID of 0.141, and later masked-transformer work Guo et al. (2024) pushed the same benchmark down to 0.045. Here, FID is the standard feature-space distance between generated and reference motion, so lower values indicate a closer match to the target

distribution. The project still keeps T2M-GPT because it produces a noticeably different feel for short locomotion prompts, which makes the variant pool more useful during comparison and composition. Its limitation here is straightforward: left-to-right decoding cannot hold the future fixed while regenerating the middle of a clip. That is why the segment-regeneration tools in Chapter 4 are limited to MoMask and MDM.

2.2.2 Masked / residual VQ: MoMask

MoMask Guo et al. (2024) stacks multiple codebooks, such that each one models the residual error left by the previous one, then runs a masked bidirectional transformer over the resulting tokens. Put simply, it refines the motion representation in layers instead of relying on a single codebook. The masked part also matters: some tokens are hidden and predicted from both their past and future context, much like filling in missing words in a sentence when the surrounding words are known. That is exactly the behaviour this project needs for local segment regeneration. At the time of design, MoMask was the strongest HumanML3D Guo et al. (2022) baseline I could reasonably integrate. One point is worth stating plainly because it is often muddled in casual summaries: MoMask is text-conditioned. The transformer takes a CLIP-encoded prompt, and that is why it works for prompted local edits in Chapter 4.

2.2.3 Diffusion-based generation: MDM

The Human Motion Diffusion Model Tevet et al. (2023) brings diffusion to motion sequences and uses classifier-free guidance for text conditioning. In diffusion models, generation begins from noise and repeatedly denoises it until a structured sample appears. Classifier-free guidance is the standard trick of pushing that denoising process harder toward the text prompt, usually improving adherence at some cost to diversity. MDM sits behind much of the later work cited in this chapter, including FlowMDM, PriorMDM, OmniControl, GMD, and CondMDI. For this project, MDM matters for two reasons. It tends to produce smoother global motion than the token models on many prompts, and its denoising process also supports interior inpainting, meaning a missing section can be filled in while respecting the surrounding frames. The downside is speed, which shows up later in the latency results in Chapter 5.

2.2.4 Streaming / online generation: DART

DART (DartControl) Zhao et al. (2025) generates motion as a stream of short primitives and can run at over 300 frames per second. That made it the first model in this review that looked plausible for online generation and live usage at runtime. I considered it as a fourth backend, but the public implementation depends on CUDA, and the development machine did not have it, meaning that even if integrated successfully, online generation would not be possible. Rather than continue with a half-working integration, I left DART as a reserved enum value and documented it as future work in Chapter 4.

2.2.5 LLM as motion: MotionGPT and MotionChain

MotionGPT Jiang et al. (2023) tokenises motion and trains a T5-style language model jointly over text and motion tokens. MotionChain Jiang et al. (2024) pushes that idea toward a conversational controller. They matter here because they are the closest published systems to using language as part of motion planning. The boundary in this dissertation is still different. The LLM writes a JSON plan made of text sub-prompts, durations, and routing hints; it never sees motion tokens, and the motion generators never see the LLM. For that reason, MotionGPT and MotionChain are adjacent works, not direct alternatives.

2.3 Beyond Simple Prompts: Composition, Inbetweening and Spatial Control

The recent literature has started to tackle the limits of single-action prompting: longer sequences, smoother transitions, and spatial constraints. The controls built for MotionGen sit in the same area, but they are solved with tooling instead of training a new generator.

2.3.1 Inbetweening and transition synthesis

Harvey et al.'s Robust Motion Inbetweening Harvey et al. (2020) introduced the standard learnt inbetweening setup: a transition generator that bridges sparse keyframes, meaning a small number of fixed poses at known times. It remains the clearest reference for why naive linear or spline interpolation looks wrong on articulated humanoid motion. CondiMDI Cohan et al. (2024) extends the same problem into text-conditioned diffusion on HumanML3D. It is the closest published analogue to both the project's `Inbetween` RPC and the transitional-prompt step inside the composition pipeline described in Section 4.7.

2.3.2 Long and multi-action composition

TEACH Athanasiou et al. (2022) introduced the idea of mapping a sequence of textual actions to motion before diffusion became dominant. PriorMDM Shafir et al. (2024) then demonstrated DoubleTake, a sequential composition pipeline built on a short-clip diffusion prior. FlowMDM Barquero et al. (2024) extended long-sequence generation and proposed Peak Jerk and Area Under the Jerk as compositional metrics; the project's variant-ranking heuristic (Chapter 4) comes from the same family of motion-smoothness measures. These papers tackle the same core problem as MotionGen, but the approach is different. MotionGen keeps per-segment generators and adds a planner plus a scene-anchor layer on top.

2.3.3 Spatial and scene-aware control

GMD Karunratanakul et al. (2023) introduced a two-stage trajectory-then-motion diffusion process under spatial guidance, and OmniControl Xie et al. (2024) generalised that idea to arbitrary-joint conditioning at arbitrary times. This dissertation takes a lighter route.

Instead of conditioning the generator directly, it fits the generated motion to scene anchors after generation. That gives up some control fidelity, but it keeps the tooling manageable and works with any of the three backends.

2.3.4 LLM-driven motion planning

AvatarGPT Zhou et al. (2024) is the nearest published comparison to the project's language-model decomposer because it explicitly uses an LLM for task decomposition and motion planning. MotionChain Jiang et al. (2024) sits nearby for the same reason. The planner in this dissertation is narrower. It outputs a structured JSON plan with segment prompts, durations, model assignments, transitional prompts, and anchor references, but it does not generate motion itself. That separation is what makes a general-purpose local LLM usable here.

2.4 Datasets and Representations

All three integrated models are trained on HumanML3D Guo et al. (2022), which has become the default English-language benchmark for text-to-motion. HumanML3D pairs short natural-language descriptions with human motion clips, providing these models a shared training and evaluation space. KIT-ML Plappert et al. (2016) is the secondary dataset usually cited alongside it. HumanML3D sits on top of AMASS Mahmood et al. (2019) and the SMPL body model Loper et al. (2015), while BABEL Punnakkal et al. (2021) provides the labelled multi-action subset used by TEACH and FlowMDM. AMASS is the large, unified motion-capture corpus underlying many modern motion papers, and SMPL is the standard parametric human body representation used to express those motions consistently.

For this project, the practical point is the representation mismatch. The models output fixed 22-joint SMPL-style sequences, but Unity expects a Mecanim humanoid. A Mecanim humanoid is Unity's standard retargetable human rig, driven through an avatar definition rather than raw joint coordinates. Bridging those two worlds becomes one of the hardest implementation problems in Chapter 4. The project standardises on BVH Meredith and Maddock (2001) as the interchange format between the Python backends and the Unity client. BVH is a long-standing skeletal animation format that stores a joint hierarchy and per-frame motion channels in plain text. Early JSON experiments were easy to inspect but not useful for long; BVH already carries the skeleton and frame data that the Unity side needs.

2.5 Evaluation Methodology

Quantitative evaluation in modern T2M papers mostly follows the same HumanML3D-based pipeline: motion-FID, R-Precision, MM-Dist, and Diversity. Motion-FID asks how close the overall distribution of generated motions is to real motions in a learned feature space. R-Precision asks whether the generated motion matches the correct text description when a retrieval system ranks candidate captions. MM-Dist measures how far matched text-motion

pairs are from one another in that same feature space, and Diversity measures how much variation a model produces across samples. MDM, T2M-GPT, MoMask, FlowMDM, and OmniControl all report variants of that suite, so Chapter 5 uses it for like-for-like comparison.

The literature also tracks more specific artefacts such as foot skating and jerk. Foot skating is the visual error where a planted foot appears to slide across the floor. Jerk is the rate of change of acceleration, so large jerk often shows up as abrupt or mechanically harsh motion. FlowMDM Barquero et al. (2024) in particular made Peak Jerk and Area Under the Jerk a standard way to talk about compositional smoothness. The project's variant-ranking heuristic draws from the same family of measures, but it is hand-built rather than learned. That distinction matters later.

The user study measures workflow instead of clip fidelity, so it uses the System Usability Scale (SUS) Brooke (1996) and NASA-TLX Hart and Staveland (1988). SUS is a short, ten-item usability questionnaire; NASA-TLX is a workload measure covering dimensions such as mental demand, effort, and frustration. The weakness of the standard model metrics is simple: they reward motions that look statistically right, not motions that actually fit a scene or help an animator complete a task. That gap comes back in Chapter 6, where the ranker and user behaviour part company.

2.6 Local LLM Inference

The composition planner runs locally to satisfy the requirement for no dependence on an external API (Chapter 3, NFR06). In practice, that meant llama.cpp Georgi Gerganov and contributors (2024b), GGUF Georgi Gerganov and contributors (2024a) weights, and a 4-bit quantised model that would still hold the planner schema together. Quantisation means storing model weights at lower numerical precision, such as 4 bits instead of 16 or 32, which reduces memory use at some cost in accuracy. llama.cpp is the lightweight local inference runtime used here, and GGUF is its common single-file model format. The project pins the llama.cpp build `b8665` with CUDA 12.4 for reproducibility and ships the planner at Q4_K_M, one of GGUF's 4-bit variants.

After trying smaller options, Gemma 4 E2B Gemma Team and Google DeepMind (2026) was the first open model that reliably emitted the required JSON structure while still fitting the local deployment target. Chapter 4 later discusses prompt-prefix reuse and KV-cache behaviour. The KV-cache is the stored attention state from the fixed prompt prefix, which can be reused in later requests instead of being recomputed from scratch. PagedAttention Kwon et al. (2023) is the reference point for that kind of optimisation even though this project does not use vLLM.

2.7 Industry Tools and the Workflow Gap

The practical landscape in which this system operates clarifies what this dissertation contributes that existing tools do not.

2.7.1 Unity Muse Animate (deprecated)

Unity Muse Animate Unity Technologies (2024) was, until recently, the main Unity-native text-to-motion tool. Unity Technologies retired the Muse service on 1 October 2025 Unity Technologies (2025), and its successor, Unity AI Unity Technologies (2026), now ships text-to-motion again, but as a cloud service tied to an account and credit budget. That changes the shape of the gap, but it does not close it. At submission, there is still no Unity-native tool that is both locally hostable and multi-model. The other lesson from Muse still stands as well: motion is treated as a one-shot output, with limited room for comparison, regeneration, or explicit model choice.

2.7.2 SayMotion (DeepMotion)

The closest live commercial competitor is DeepMotion's SayMotion DeepMotion Inc. (2024), a hosted, web-based standalone tool. It is worth noting the historical naming collision: the same product was already branded "MotionGPT," unrelated to the academic MotionGPT Jiang et al. (2023). Recent SayMotion releases have also added browser-side generative features, including inpainting and prompted motion refinement, which narrows the raw feature gap. SayMotion's positioning is nevertheless the inverse of this project: it offers strong models behind a polished hosted UI but keeps generation and refinement detached from the target Unity scene, selected humanoid, and persistent project assets, requiring repeated export-import churn to verify motion against a rig and scene, exactly the round-trip that motivates the editor-first integration argued for in Chapter 3.

2.7.3 Cascadeur

Cascadeur Nekki Limited (2025) (v2025.3) is an AI-assisted *keyframe* animation tool with AutoPosing, AutoPhysics, and AI inbetweening features. It is a useful contrast point. The AI is there to support a keyframe workflow, not to replace it with prompting. That framing supports this project's decision to keep keyframe authoring out of the core flow and lean on prompted regeneration for refinement instead.

2.7.4 The gap this dissertation fills

No tool, to my knowledge, combines (a) local hosting, (b) Unity editor integration, (c) multi-model routing, (d) LLM-driven composition with scene anchors, (e) post-generation refinement via text-prompted segment regeneration and inbetweening, and (f) persistent project-asset integration. Each of (a)–(f) maps to a contribution stated in Chapter 1 and implemented in Chapter 4.

2.8 Inverse Kinematics and Motion Polishing

The project's path-editing and contact-locking tools sit within the long inverse-kinematics (IK) and post-hoc motion-correction tradition, but only as supporting utilities instead of the main generative paradigm. Classical IK solvers such as FABRIK, Cyclic Coordinate Descent, and Jacobian-based methods compute joint rotations that satisfy end-effector constraints Aristidou and Lasenby (2011); Wang and Chen (1991); Buss (2009), while Gleicher's Motion Path Editing Gleicher (2001) and later foot-locking and trajectory-smoothing techniques Lee and Shin (1999); Holden et al. (2017) show how an existing motion can be corrected after the fact. That lineage is relevant here because the project's scene-fit root-path tool and contact-locking pass are both post-generation refinements applied to an already generated clip, not alternatives to the generative models themselves.

2.9 Synthesis and Identified Gaps

Five gaps emerge from the foregoing review and motivate the work in the remaining chapters. First, every integrated text-to-motion model is fundamentally single-prompt and short-horizon, which the project addresses with the composition pipeline (FR14-FR19, Chapter 4). Second, although Unity AI has re-implemented an editor-native text-to-motion tool, it is cloud-only, based on credits, and single-model; thus, there is still no Unity-native tool that is both locally hostable and model-extensible, the niche the project's plugin occupies. Third, the only credible commercial alternatives depend on hosted APIs for both generation and any LLM planning; the project's local Gemma 4 planner with manifest-driven model installation answers this. Fourth, browser-based tools such as SayMotion now offer generative refinement features including inpainting, prompted refinement, and clip merging, but these remain detached from the target Unity scene, selected humanoid, and persistent project assets; the project's text-prompted segment regeneration and inbetweening bring that refinement loop into the editor-native authoring context. Fifth, the standard evaluation suite rewards distributional fidelity rather than authoring usability, motivating the workflow-focused user study in Chapter 5 and the explicit acknowledgement in Chapter 6 that the variant-ranking heuristic is hand-designed and partially contradicted by user behaviour. The next chapter translates these gaps into concrete functional and non-functional requirements.

Chapter 3

Requirements and System Design

The requirements in this chapter come from the gaps outlined in Chapter 2: the models are useful, but using them in a real animation workflow is still difficult.

3.1 Design Drivers

The primary driver of this project is to make text-to-motion generation usable within a real authoring workflow, not just to prove that it is possible through simple generation. This is supported by the goal of facilitating iterative authoring, allowing users to not only generate multiple variants of the same prompt but also to compare and refine them afterward. I also want to reduce setup friction to almost none by dynamically downloading the text-to-motion and language models after the installation of the plugin, whilst still giving them full control over the setup of their workflow, with multiple ways to configure the generation and decomposition of prompts.

These drivers correspond to core requirements. Additionally, we have two exploratory goals: firstly, to be able to handle prompts that are more complex than single actions, which is a major limitation of these models; secondly, to automatically rank generated variants of the same prompt to surface better candidates sooner. The following two sections translate these drivers into specific functional and non-functional requirements.

3.2 Functional Requirements

The functional requirements below are derived directly from the design drivers identified in Section 3.1. They are grouped by workflow area. Requirements marked *exploratory* are investigative design goals: implemented and evaluated within this dissertation, but not claimed as fully reliable or generalisable solutions.

Table 3.1: Functional requirements grouped by workflow area.

ID	Status	Requirement
Generation		
FR01	Core	The user can submit a natural language prompt and receive one or more generated humanoid `AnimationClip`s within the Unity editor.
FR02	Core	The user can select which motion generation model (T2M-GPT, MoMask, or MDM) to use for a given generation request.
FR03	Core	Multiple motion variants can be generated in parallel from a single prompt submission.
FR04	Core	Generated clips can be automatically applied to the selected scene Animator for immediate in-editor preview.
Comparison and Library		
FR05	Core	The user can browse all past generations in a persistent library within the Unity editor.
FR06	Core	The user can preview and compare multiple clips from the library.
FR07	Core	The user can access the `AnimationClip`s in the native Unity file explorer.
FR08	Core	The generation history is preserved across editor restarts.
Editing and Refinement		
FR09	Core	The user can regenerate a selected window of time within a clip using a new prompt, with the surrounding motion held fixed.
FR10	Core	Post-processing operations can be applied non-destructively.
FR11	Core	The user can adjust the root trajectory of a generated motion to fit a target position or path in scene space.
FR12	Core	Edit history is tracked so the user can compare an edited clip with its unmodified source.
FR13	Core	The user can provide two clips and generate a linking animation between them, either with a prompt or not.
Composition and Decomposition *(exploratory)*		
FR14	Exploratory	The user can submit a complex multi-action prompt for automatic decomposition into ordered sub-prompts by a language model.
FR15	Exploratory	Alternatively, the user can define composition segments manually without involving a language model.

Continued on next page

Table 3.1 continued...

ID	Status	Requirement
FR16	Exploratory	Each segment should have options such as selecting the model, duration, prompt, transitional prompt, spacial anchors and type of motion.
FR17	Exploratory	Adjacent segments are merged using an inbetweening model conditioned on a generated transitional text prompt.
FR18	Exploratory	Segments should be defined and generated in relation to spacial anchors, which are placed and orientated by the user.
FR19	Exploratory	The user can inspect, edit, and re-generate individual segments within a composition.
Variant Ranking *(exploratory)*		
FR20	Exploratory	Generated variants are automatically scored against motion-quality metrics and surfaced in rank order.
FR21	Exploratory	Per-metric scores are visible to the user in the clip inspector.
Evaluation Support		
FR22	Core	The system must be operable within a structured user study comparing it against an existing motion generation tool.
FR23	Core	Session data and researcher observations must be exportable for offline analysis.

Each requirement is revisited in Chapter 4, where its realisation in the implemented system is described.

3.3 Non-Functional Requirements

Non-functional requirements cover the properties that the system should exhibit as a whole rather than any given specific capability. For a tool focusing on workflow integration rather than model invention, these properties are particularly important. For example, a plugin that generates quality motion yet blocks the Unity editor thread, requires manual configuration, or cannot run without a cloud subscription cannot be useful as a tool, regardless of how good it's output is. The following requirements are grouped by concern, each with a rationale based on the design context of this project.

Table 3.2: Non-functional requirements grouped by concern area.

ID	Requirement and Rationale	Concern
Performance		
NFR01	*Non-blocking editor operation.* All generation and communication with the backend must be asynchronous from the Unity main thread. Unity's editor loop runs on the main thread; any synchronous call on this thread freezes the entire editor interface until the call returns. As inference can take tens of seconds, synchronous communication is not viable.	Performance
NFR02	*Acceptable inference latency.* The time from prompt submission to a usable `AnimationClip` in the library must be within a practical range. Generation time is hardware-dependent and cannot be specified as a fixed upper bound. For local decomposition planning, minute-scale latency is acceptable only for occasional editor-side requests as opposed to conversational use. The timings for each model measured on the development hardware are reported in Chapter 5.	Performance
Deployment		
NFR03	*Local deployability.* The final system must run on a single developer machine without any cloud dependency. A local deployment supports a self-contained evaluation environment and removes reliance on third-party service availability, network connectivity, or usage cost.	Deployment
NFR04	*Automatable backend setup.* Launching the Python inference backend must not require the user to open a terminal or manually activate a virtual environment. The backend runs inside a Python virtual environment (`venv`) with precise package versions; exposing this activation step to the user would exclude the non-specialist target audience.	Deployment

Continued on next page

Table 3.2 continued...

ID	Requirement and Rationale	Concern
NFR05	*On-demand model management.* Both motion model checkpoints and local language model weights must be downloadable and verified from within the editor without manual file placement. Motion model checkpoints (1–10 GB per model) are managed via `backend_manifest.json`; the Gemma GGUF weight and llama.cpp runtime are managed via a separate `local_llm_manifest.json`. Both use the same manifest-driven download and verification mechanism.	Deployment
NFR06	*No mandatory external API dependency.* The composition feature must provide a fully local inference path for language model decomposition. Requiring a third-party API key introduces cost, data-privacy considerations, and a dependency on external service availability. A locally hosted alternative must therefore be available at no additional configuration cost to the user.	Deployment
Interoperability		
NFR07	*Multi-model extensibility.* The backend must be capable of routing generation requests to different model implementations without requiring changes to the Unity frontend. No single text-to-motion model performs uniformly well across all prompt types; supporting multiple models allows the most appropriate one to be selected per request.	Interoperability
NFR08	*BVH as the frontend–backend interchange format.* The contract between the Unity plugin and the Python backend must use standard BVH (BioVision Hierarchy) as the skeletal animation exchange format. BVH is the native output format of the integrated motion models, removing the need for an intermediate conversion layer and allowing any BVH-producing model to be integrated without frontend modification.	Interoperability
Scope		

Continued on next page

Table 3.2 continued...

ID	Requirement and Rationale	Concern
NFR09	*Humanoid rig focus.* The workflow targets Unity's Humanoid Avatar system as the standard retargeting target. The text-to-motion models integrated in the backend are trained on human body motion datasets (HumanML3D, KIT-ML) that assume a human skeletal structure; Unity's Humanoid Avatar provides the standardised retargeting layer for this rig type. General retargeting for arbitrary rig topologies is outside the scope of this dissertation.	Scope
NFR10	*Windows-first deployment.* The system is developed and validated on Windows. Backend startup automation relies on PowerShell scripts and Windows-compatible process management. Cross-platform validation is left as a future consideration.	Scope

NFR01 and NFR02 interact directly with the system's communication architecture: both are addressed through asynchronous gRPC dispatch from the Unity plugin, discussed further in Section 3.4. NFR03 through NFR06 are addressed together through the manifest-driven deployment infrastructure described in Section 3.6.

3.4 System Architecture Overview

Figure 3.1 summarises the four core components: the Unity editor plugin, the local Python inference backend, the model management layer, and the standalone user-study application. The Unity plugin and backend communicate through the shared `motion.proto` gRPC contract, while optional decomposition runs either through a local `llama.cpp` service or a user-supplied endpoint.

The plugin is the user-facing surface, handling request submission, preview, backend lifecycle, and persistent clip history inside the editor. The backend validates requests, routes them to T2M-GPT, MoMask, or MDM, and manages model installation through manifests and on-demand downloads. The user-study application is separate evaluation software rather than part of the runtime motion workflow.

3.5 Workflow Design

The workflow is designed around whether generated motion can be previewed, compared, edited, and applied in scene, not just whether a model can emit a plausible sequence.

Figure 3.2 summarises the normal loop: the user submits a prompt, the local backend returns `BVH`, the plugin converts it into a humanoid `AnimationClip`, stores it in the library,

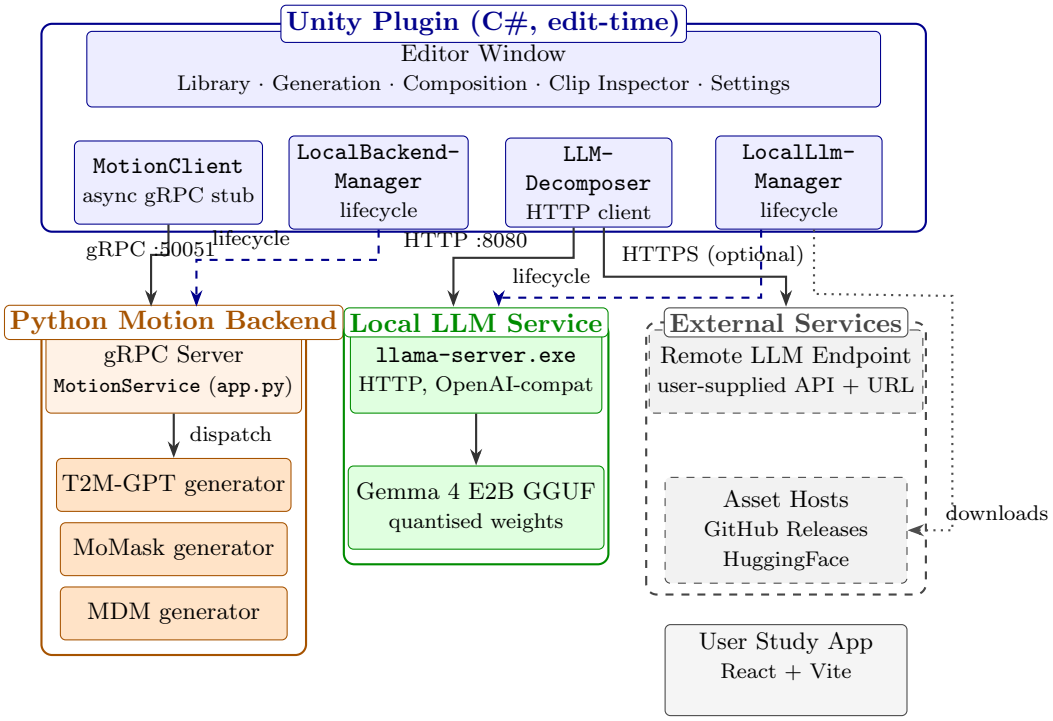


Figure 3.1: System architecture. Solid arrows show runtime data flow over loopback (gRPC, HTTP) or the optional HTTPS path to a remote language model. Dashed arrows show process lifecycle and manifest queries driven from inside the editor. Dotted arrows show one-off asset downloads, content-checked against SHA-256 hashes recorded in `backend_manifest.json` and `local_llm_manifest.json`. The user study application is structurally independent and shares no runtime channel with the plugin or backend.

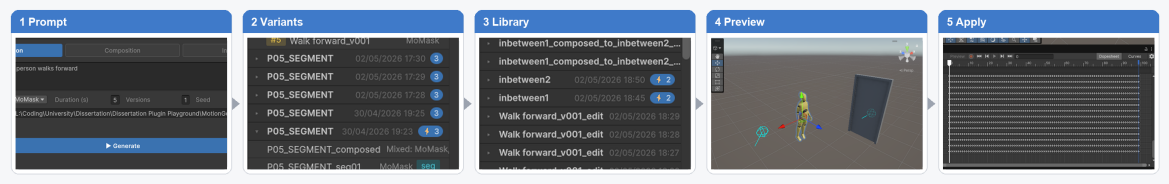


Figure 3.2: Core MotionGen authoring loop. The user enters a prompt and generation settings, compares the ranked variants produced for the current request, reuses clips from the persistent library, previews the selected motion on the target humanoid inside Unity, and then applies the chosen clip as a usable animation asset.

previews it on the selected character, and applies the chosen result as a reusable asset. Because first-pass generation is rarely enough, the same loop also supports segment regeneration, inbetweening, post-processing, and root-trajectory adjustment without destroying originals.

For complex motions, the composition pipeline splits a prompt into segments either manually or through an LLM, assigns models, and stitches segments with inbetweening while optionally resolving scene anchors. That path remains editable throughout, so manual composition remains the fallback when decomposition is poor or unavailable.

3.6 Key Design Decisions and Rationale

The system architecture choices address the requirements from Section 3.2. Where trade-offs existed, the design favoured workflow integration, low setup friction, and user control.

The core infrastructural decisions were local hosting, `BVH` interchange, and editor-first integration. A local Python backend keeps the system self-contained, easier to evaluate, and usable without recurring hosting cost or network dependence. Using `BVH`, the native output format of the selected models, avoids a custom transport layer and keeps later model swaps straightforward. Embedding the tool inside Unity removes the export-import loop that makes external tools awkward to judge against the target rig and scene.

Scope was constrained deliberately. The system targets humanoid animation because current text-to-motion models are trained mainly on humanoid datasets and Unity's Humanoid Avatar system provides robust retargeting within that scope. The backend remains model-selectable rather than single-model because T2M-GPT, MoMask, and MDM perform differently across prompt families, and composition sometimes benefits from choosing different models per segment.

Composition also had to degrade gracefully. LLM decomposition is optional so simple prompts avoid extra latency, and manual composition remains available when the planner is poor or unavailable. The same principle applies to LLM access: the system supports a user-provided API endpoint, a locally hosted Gemma 4 path, or pure manual composition, so the workflow does not depend on one provider or one hardware profile.

Finally, the system treats inbetweening and ranking as pragmatic heuristics rather than solved intelligence. Transition text is passed into the inbetweening model because a geometric

blend alone ignores the intended action between segments. Variant ranking stays session-relative and heuristic-based because no labelled dataset existed for absolute quality prediction; the goal was only to surface cleaner clips sooner, not to claim a validated preference model.

3.7 Chapter Summary

This chapter translated the workflow gap identified in Chapter 2 into a concrete design brief for MotionGen. The functional requirements defined the authoring loop around generation, comparison, editing, composition, scene anchors, and ranking, while the non-functional requirements constrained responsiveness, local deployment, extensibility, and reproducibility. From those requirements, the chapter derived a Unity editor frontend backed by a local Python service and an optional LLM runtime, with a workflow centred on persistent clip history, in-scene preview, and editable multi-segment composition. The main design trade-offs were also made explicit: local hosting over cloud dependence, `BVH` over a custom interchange, humanoid scope over general retargeting, and heuristic ranking as an exploratory aid rather than a validated quality signal. Chapter 4 then shows how those design decisions were realised in the final artefact.

Chapter 4

Implementation

4.1 Implementation Overview

This chapter shows how the authoring loop in Figure 3.2 was realised as a usable Unity tool. MotionGen is a workflow built around existing text-to-motion models, with a Unity editor frontend, a local Python backend, manifest-driven model management, an optional planner, and authoring tools that sit between first generation and final use in the scene.

Unity was chosen because the generation step belongs in the same environment where the clip is previewed, positioned, and applied. A platform-agnostic export would reach more tools, but it would lose the direct integration with Unity's humanoid animation, scene view, and asset pipeline, which makes the workflow practical.

Figure 4.1 shows the organising principle for the rest of the chapter. The orange block covers the backend boundary, validation, routing, and BVH export. The blue block covers the editor frontend that submits requests and receives results. The teal block covers BVH import, humanoid reconstruction, and the root-stability work needed to make that import usable. The dashed green path shows the composition variant, where one prompt expands into several requests. The purple block covers what happens after generation: library storage, ranking, and reuse inside Unity's existing animation tooling.

The chapter follows that request path through the system. Section 4.2 covers the backend interface and runtime. Section 4.3 covers the editor surface that submits requests and presents returned clips. Sections 4.4 and 4.5 cover the BVH-to-humanoid bridge and the stability problems within it. Section 4.6 covers the post-generation tools applied to reconstructed clips. Sections 4.7 and 4.8 cover the multi-request composition path. Section 4.9 covers what happens once clips have been generated. Section 4.10 concludes with the defensive behaviour that cuts across every stage above.

4.2 Backend Interface

The orange block in Figure 4.1 is the backend side of a single request, from the protocol boundary through validation, model dispatch, and BVH export. The subsections below

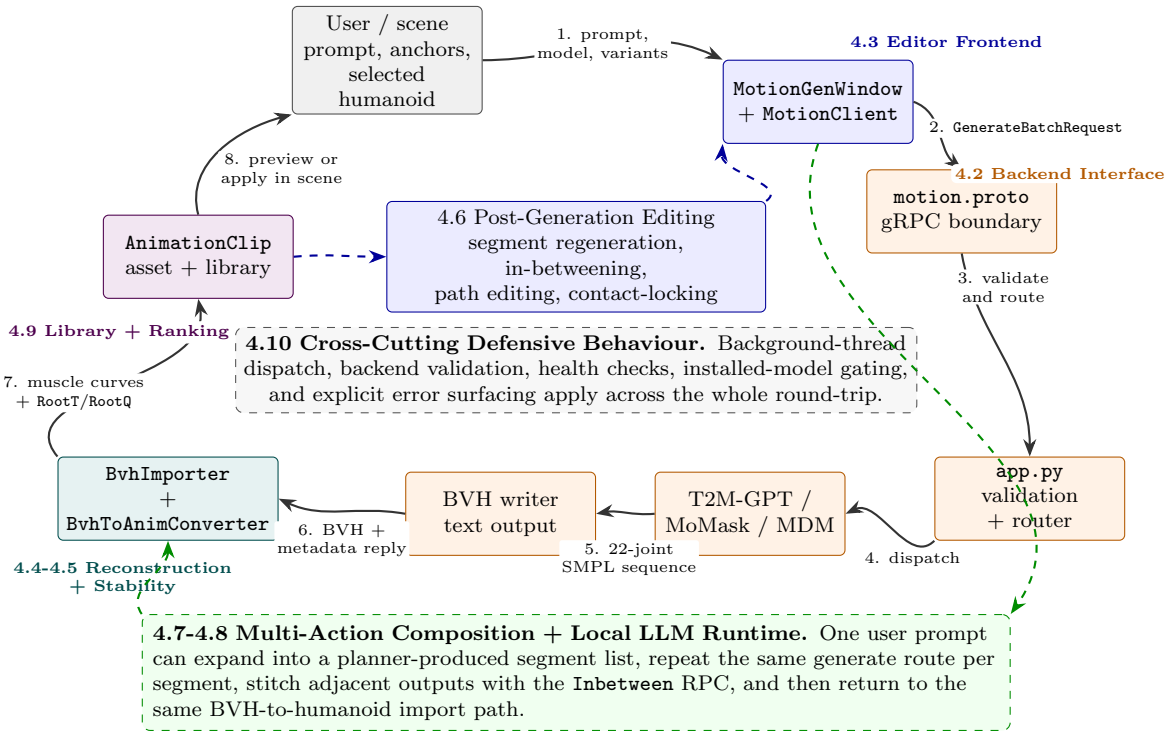


Figure 4.1: Pipeline view of a single generation request, arranged as a circular flow from prompt entry in the editor to a usable `AnimationClip` asset on the scene humanoid. The dashed separator between the contract and backend boxes marks the gRPC boundary defined in `motion.proto`. The dashed composition path shows the multi-request variant used in Sections 4.7 and 4.8, where one prompt expands into multiple segment generations before returning to the same BVH-to-humanoid import path.

cover four parts of that path: choosing BVH as the interchange format, defining the gRPC contract, wrapping the backend in an editor-managed runtime and installer, and adding the diagnostic tools that make the path debuggable. Together they realise FR01, FR03, FR09, FR13, FR14, NFR01, NFR03, NFR04, NFR05, NFR07, and NFR08.

4.2.1 Choosing BVH as the Interchange Format

To realise NFR08, the interchange format had to be chosen together with the retargeting strategy. The models decode to 22-joint Cartesian SMPL sequences, whereas Unity's humanoid system expects muscle curves on an avatar. Those representations are far enough apart that the transport format also determines where the expensive reconstruction work happens.

Option A was JSON-encoded numerical arrays. I prototyped that first because it was easy to inspect from C#. It was abandoned for three reasons. The arrays were large enough to make encoding and parsing noisy; the Unity side still needed a second conversion step before the motion became useful; and the format did nothing to simplify retargeting.

Option B was BVH, the Biovision Hierarchy skeleton-and-motion text format discussed

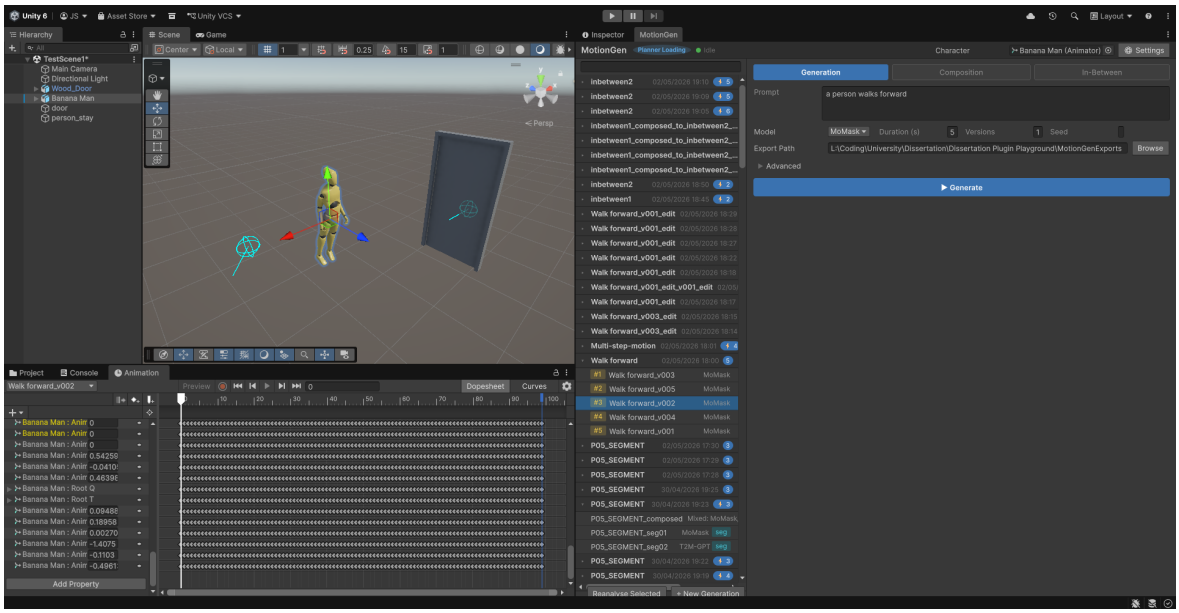


Figure 4.2: MotionGen in use inside the target Unity scene. The docked editor window, selected humanoid, and resulting clip remain visible together, so scene fit can be judged without leaving the editor.

in Chapter 2, produced from the reconstructed 22-joint output using a script from MoMask's utilities. BVH was selected because it already carries a skeleton hierarchy and per-frame motion data in a form that the Unity importer can consume directly. The `MotionFormat` enum still keeps both `BVH` and `JSON`, but only BVH works end to end. BVH is therefore the format that satisfies NFR08 and keeps NFR07 manageable, because any backend that can emit BVH can reuse the same frontend import path.

4.2.2 gRPC Service Contract

The shared contract, meaning the protobuf-defined message schema that fixes the frontend–backend boundary, lives in `motion.proto`. It is exposed over gRPC, Google's binary remote procedure call protocol, and mirrors the editor workflow instead of pretending everything is one generic prompt-to-motion call. There are dedicated RPCs (remote procedure calls) for generation, batch generation, editing, in-betweening, and composition. That keeps the Unity side stable while model-specific code stays behind generator interfaces. The RPC set also maps cleanly onto the requirements table from Chapter 3: `Generate`/`GenerateBatch` realises FR01 and FR03, `Edit`/`EditBatch` realises FR09, `Inbetween` realises FR13, and `Compose` realises FR14.

The contract also lists the motion models that the backend is able to serve. `MotionModel` currently exposes `T2M_GPT`, `MOMASK`, and `MDM`, with a fourth value reserved for `DART`. DART was considered seriously during planning because it is a frontier model that supports online generation, and online generation would have unlocked use cases that the other three simply cannot fulfil: runtime NPC animation, reactive character motion, and other interactive

scenarios where motion has to be produced as the scene unfolds rather than baked ahead of time. However, two practical obstacles prevented it from being shipped. The first is that DART's runtime depends on `conda` for environment management, which the other three models do not require, so adding it would have meant including a parallel install path alongside the otherwise unified Python service. On its own, this isn't the most unmanageable problem, but on top of that, DART's GPU acceleration depends on CUDA, and the development hardware available for this project only included an AMD GPU with no CUDA support; without GPU acceleration, the online-generation argument that motivated DART in the first place collapses. The integration was therefore scrapped, and the enum value is left in place so that the work can be picked up later on hardware that supports CUDA.

For editing and in-betweening, the backend does not accept raw Unity assets. Instead, it uses `MotionJointSequence` payloads, meaning sampled per-joint motion sequences extracted from an existing clip, together with explicit time windows. That representation is a better fit for the model-facing operations implemented on the Python side.

The server entry point in `app.py` centralises validation and dispatch. Requests are first checked for valid prompt text, frame rate, duration, seed values, and batch size, and are then routed through a generator table to the selected backend implementation. The batch endpoints also resolve per-item seeds, generate deterministic filenames, and attach metadata describing the batch index, resolved seed, and request context. This gives the Unity side a stable contract while keeping model-specific behaviour encapsulated behind generator interfaces.

4.2.3 Backend Runtime and Manifest-Driven Installer

The backend is something that the editor manages, not a separate research setup that the user has to manage. Unity starts it as a localhost process, checks it with `Ping`, and stores the configuration locally so the runtime returns to the same state on the next launch.

Installation follows the same idea. `backend_manifest.json` lists the artefacts each model needs, and `model_manager.py` checks, downloads, and verifies them on demand. That keeps the shipped plugin small while still letting the user install only the models they actually want. The same installation state is then used to gate features at runtime, so missing models fail clearly instead of disappearing into backend errors.

The runtime and installer, therefore, deliver NFR03, NFR04, and NFR05 through one mechanism: the backend stays local, the user never has to activate a virtual environment manually, and model assets can be installed on demand from inside the editor.

4.2.4 Diagnostic BVH Viewer and Per-Joint Logging

Most of the time spent implementing the BVH bridge could not have been productive without diagnostic tooling. A standalone Python viewer (Figure 4.3) shows the raw model output before it ever reaches Unity, which made it much easier to differentiate generation bugs from conversion bugs. On the Unity side, a per-joint logging tool compares expected and observed

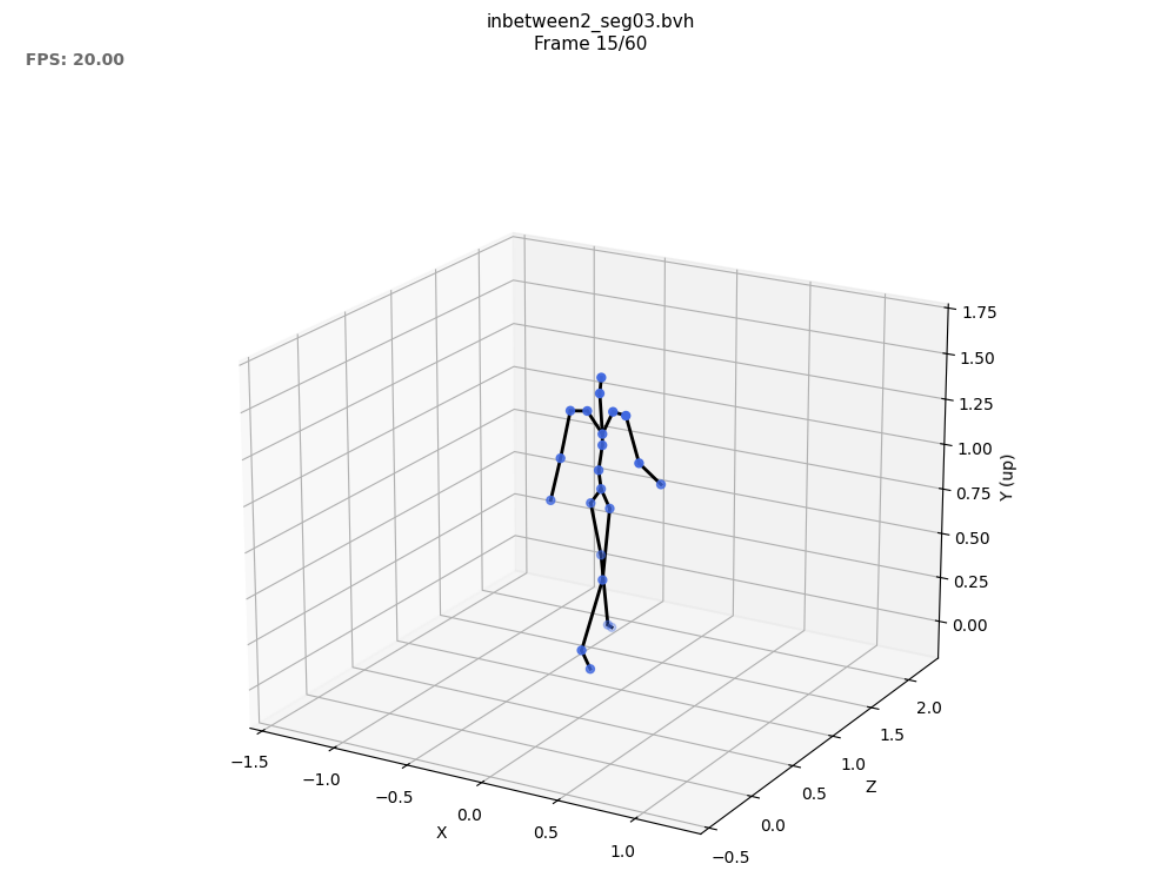


Figure 4.3: The diagnostic tool developed to assess raw BVH output from the motion models.

poses across imported clips so failures can be traced to specific bones. Without those two tools, most of Section 4.4 would have been guesswork.

Taken together, the backend structure supports three goals at once: it isolates model dependencies from Unity, exposes the generative features through a shared contract, and reduces setup friction enough for the workflow to be used repeatedly during testing and evaluation.

4.3 Editor Frontend

The blue block in Figure 4.1 covers what happens before a request crosses gRPC and immediately after a reply returns to the editor. The subsections below cover the UI framework migration, the decision to output ordinary Unity assets rather than Timeline-only segments, the preview scrubber that made fresh clips inspectable, the final window layout, and the persistent history that keeps requests comparable across editor sessions. Together, they realise FR01, FR05, FR06, FR08, and NFR01.

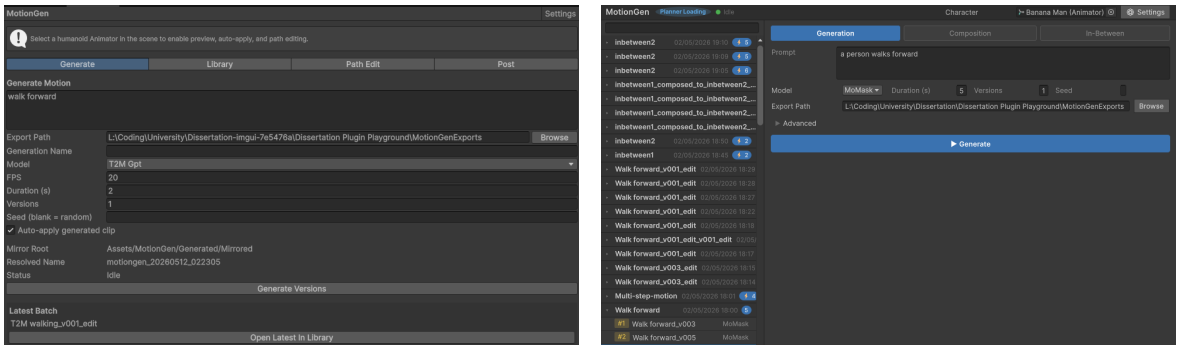


Figure 4.4: The MotionGen editor window before and after the migration from IMGUI to UIToolkit.

4.3.1 UI Framework: IMGUI to UIToolkit

I built the first version in Unity's older `IMGUI` because it was quick. IMGUI is immediate-mode: the interface is redrawn procedurally with each update, which is fine for small tools but becomes impractical as the window grows in scope. Moving to `UIToolkit`, Unity's retained-mode editor UI system, made the layout composable, ensured the status badges and panels were consistent, and gave the library and composition views room to grow into proper pages instead of long IMGUI loops.

UIToolkit also unlocked the structure that the final editor window needed. The persistent left-hand library, the larger right-hand work area, reusable `VisualElement` panels, and shared style sheets all became much easier to manage once the interface was built as a retained tree instead of through procedural draw calls. That migration is what made FR05 and FR06 practical rather than merely possible.

4.3.2 Asset-Based Output vs Timeline Integration

To realise FR01, two output paths were considered.

Option A was to integrate generated motion directly into Unity's Animation or Timeline window, with each prompt becoming a segment that could be reordered and trimmed. That idea was abandoned for two reasons. First, Unity animation workflows are asset-driven: animators expect `AnimationClip` assets in the Project window to be dropped onto Unity animator controllers and edited with existing tooling. Second, the Timeline path would not have resolved the more challenging problem because the generated motion still had to be reconstructed into an ordinary humanoid clip before the rest of Unity could utilise it.

Option B was to write a real `.anim` asset for every generation, store it under `Assets/MotionGen/Generated` (configurable in settings), and let Unity treat it as ordinary content. A `.anim` asset is Unity's serialised representation of an `AnimationClip`, so this keeps the output compatible with the Project window, Animator controllers, and Timeline itself. This is the path Section 4.4 actually produces. The scrubber in the next subsection and the Project-window integration later in Section 4.9 are both consequences of that choice.

Figure 4.5: The scrubber implemented in the preview view of a motion.

4.3.3 Editor-Native Preview Scrubber

A subtler issue surfaced once `.anim` generation worked. Newly generated clips could not be scrubbed inside Unity's animation panel until the scene had been entered and exited at least once. This turned out to be due to an internal binding step that runs only on play-mode entry.

Rather than wait for a fix or force users through a play-mode workaround, the editor window grew its own preview controls: a scrubber 4.5, play and pause buttons, and the ability to preview an animation on a humanoid in the scene without applying the clip permanently. The result is an editor-native preview that works immediately on a freshly generated clip and previews motion non-destructively on the actual scene character so that deselecting the clip restores the original. What started as a workaround became one of the more frequently used parts of the tool, and it directly supports FR06 by making comparison possible before a clip is committed.

4.3.4 Editor Window Layout

Figure 4.6 shows the editor window when MotionGen first opens, and Figure 4.7 shows the panel that appears once a clip is selected from the library. Both figures are referenced throughout the rest of the chapter as features are introduced, so they serve as the chapter's reference points.

Figures 4.6 and 4.7 show the single-surface design point: generation controls, library browsing, clip inspection, preview, and edit tools remain together in one panel, which is what allows FR01, FR05, and FR06 to operate as one loop.

4.3.5 Persistent Generation History

Persistent history is what turns the window into more than a transient request form. `MotionGenGenerationHistory` records sessions, prompts, resolved seeds, mirrored BVH files, imported clip paths, quality scores, timestamps, and edit lineage, so the same request can be revisited and compared later. That storage layer is what delivers FR05 and FR08.

`MotionClient` sits beside it as the frontend half of NFR01. It sends gRPC requests on background threads, waits for replies away from the Unity main thread, and only returns to the main thread when Unity assets need to be created or updated. That is what keeps long backend calls from freezing the editor while still allowing the resulting clips to enter the same persistent history.

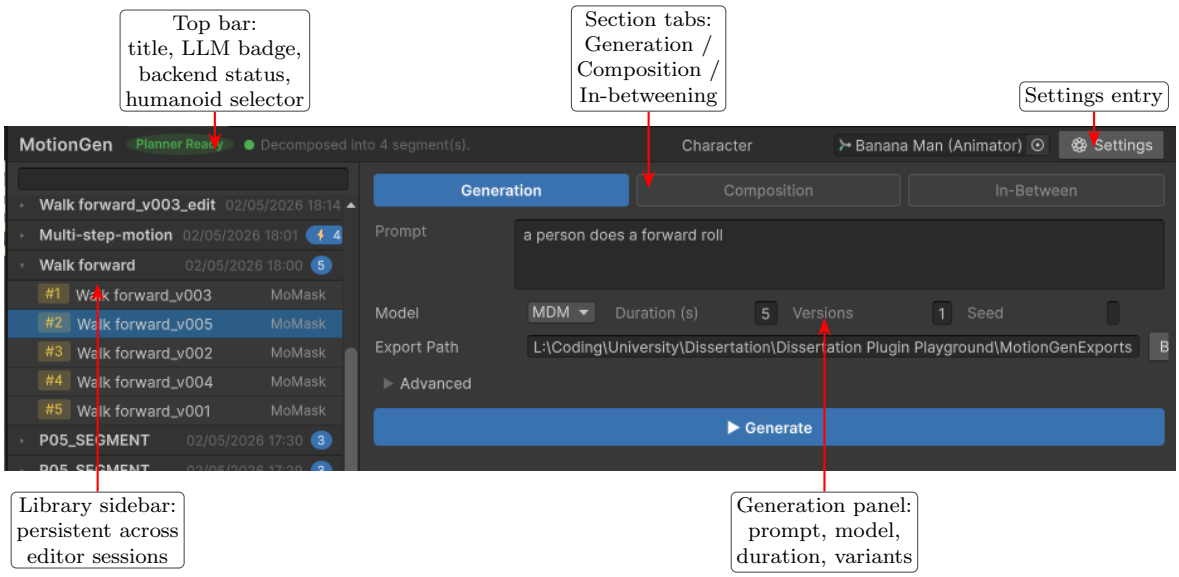


Figure 4.6: The `MotionGenWindow`, with the top bar, section tabs, persistent library sidebar, and primary content area shown in its simple-generation state. The backend-status badge is live and updates with operations (e.g. *generating segment 2 of 4* during composition).

4.4 Motion Reconstruction

The teal reconstruction stage in Figure 4.1 is where FR01 becomes a usable humanoid clip under NFR09. Getting that stage right was the single hardest part of the build. The models produce 22-joint SMPL-style sequences, but Unity's humanoid animation system expects normalised humanoid muscle curves applied through an avatar. I first tried going directly from the raw arrays to those curves, but that path kept losing the rest-pose offset and made the resulting clip difficult to reason about. Routing everything through BVH turned out to be the workable option, and that is the bridge `BvhImporter` and `BvhToAnimConverter` implement.

On the plugin side, `BvhImporter` parses the BVH hierarchy and uses Unity's `AvatarBuilder`, the API for constructing a humanoid avatar from a skeleton hierarchy, to build a temporary avatar for retargeting. `BvhToAnimConverter` then samples each frame with a `HumanPoseHandler`, Unity's API for reading and writing humanoid poses, and writes the result back into a new `AnimationClip`. The output contains muscle curves for the normalised humanoid joint rotations, plus explicit `RootT` and `RootQ` curves for the root translation and orientation that cannot be represented cleanly through muscle channels alone. The same path now captures the calibration pose automatically, so the imported curves are expressed against the correct Unity rest pose instead of carrying a permanent offset.

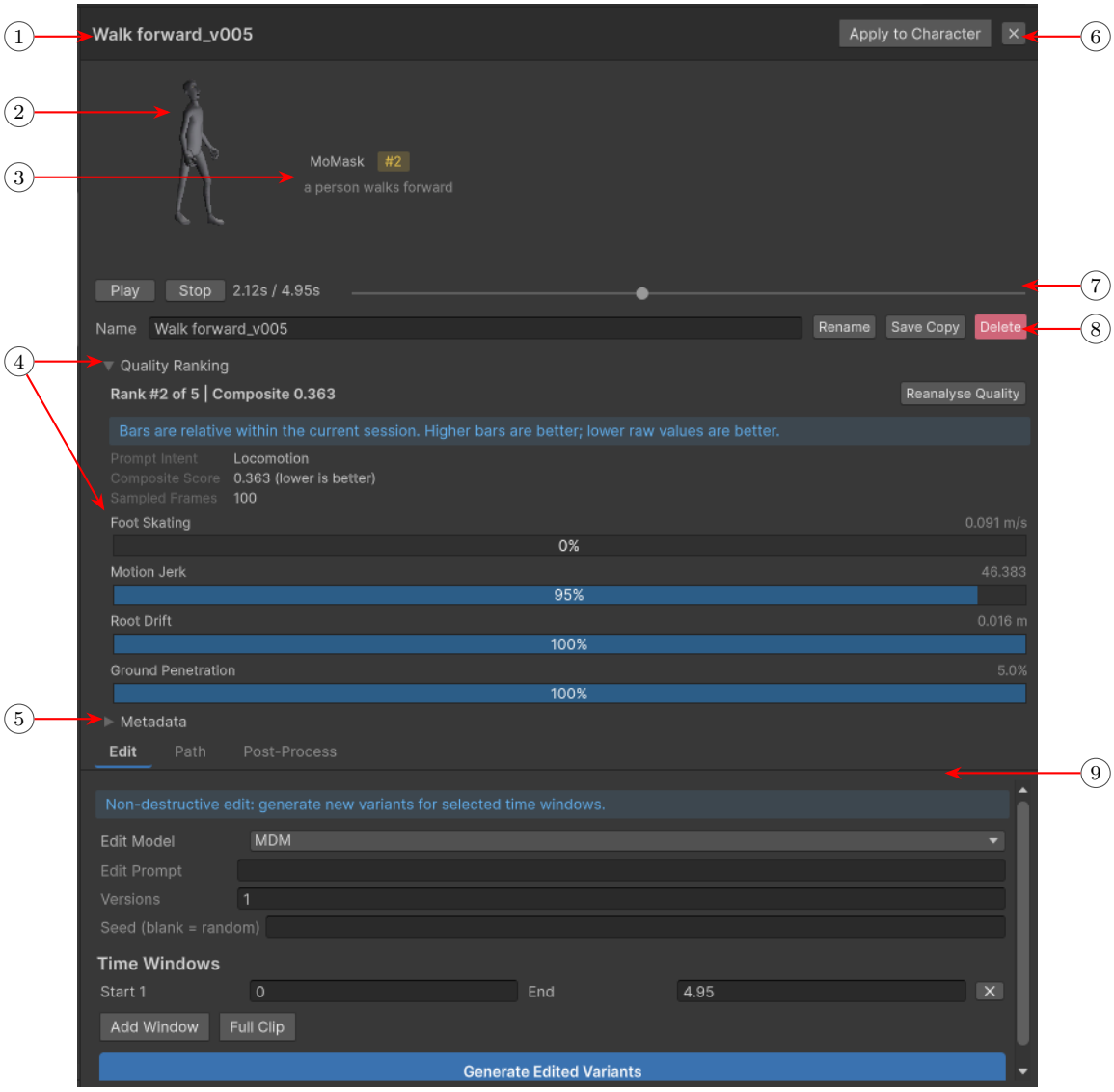


Figure 4.7: Clip detail panel. **1** title; **2** character preview; **3** model and prompt; **4** per-metric variant-ranking scores; **5** expandable metadata block; **6** apply-to-character (creates an `AnimatorController` and assigns the clip) and collapse button; **7** scrubber with play/pause for non-destructive scene preview; **8** rename, save-as-copy, delete; **9** three edit sub-panels: text-prompted segment regeneration, path editing, and post-processing.

4.5 Root Motion Stability

The next problem in Figure 4.1 appeared immediately after reconstruction. The motion was semantically correct, but the root trajectory was not stable in scene space. Characters would fly all over the place! To keep FR01 usable and to make FR11 a deliberate scene-fit edit rather than an emergency repair, that drift had to be isolated first.

Option A was path editing. The user can visualise the root path in the scene, place keyframes, drag them to the desired positions, and bake the corrected trajectory back into the clip's root translation curves. This was originally a survival tool: even when the model output was borderline unusable on its own, a few path keyframes could turn it into a clip that could still be previewed.

Option B was contact locking, implemented as a post-processing pass. The user selects which limbs are expected to be in contact with the ground for the motion: hands, feet, or both, since motions such as handstands and crawling change which contacts need to stay fixed. The system then detects contact windows frame by frame and constrains the selected joints against ground-relative drift during those windows. This helped but only partially. It stabilised the planted frames and then snapped back between them, which was still visibly wrong.

Option C moved the fix back into reconstruction, where the problem actually belonged. Automatic T-pose calibration, correct handling of the BVH world frame, and explicit `RootT`/`RootQ` reconstruction removed the drift at the source. Path editing then became the implementation of FR11 rather than a patch for broken import, and contact locking became a non-destructive clean-up pass supporting FR10 and FR12. Its contact-window detector was later reused for the anchor-aware swing-frame corrections in Section 4.7. Here, swing frames refer to the frames between detected contacts when the limb is moving rather than being planted.

4.6 Post-Generation Editing

The editing loop in Figure 4.1 begins once a reconstructed clip exists, but before that clip is accepted as final. The subsections below cover three pieces of that loop: local segment regeneration, user-facing in-betweening, and the spatial clean-up tools that sit beside the reconstruction bridge. Together, they realise FR09, FR10, FR11, FR12, and FR13.

4.6.1 Text-Prompted Segment Regeneration

Text-guided segment regeneration is the main implementation of FR09. The user selects an existing clip, an edit prompt, one or more time windows, a duration, and a variant count; the system resamples the source clip into the backend-facing joint representation, sends the request to the backend, and imports the returned motion through the same BVH path as an ordinary generation. Only MoMask and MDM are offered here because both support interior-window inpainting, the masked or denoising-based regeneration of a missing temporal window conditioned on the surrounding motion, as discussed in Section 2.2.

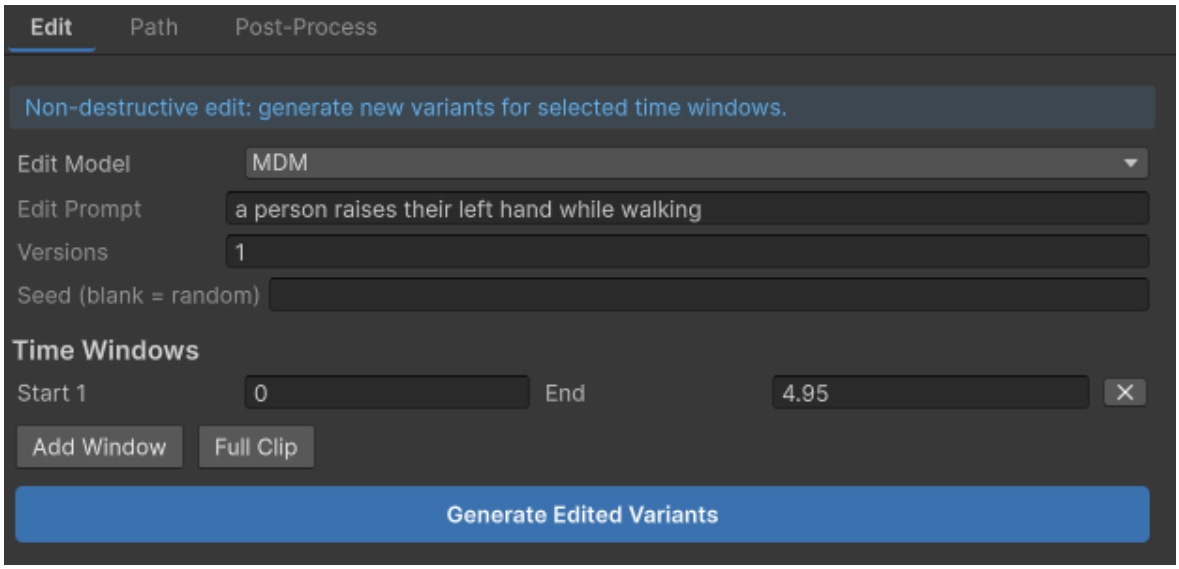


Figure 4.8: The segment regeneration tool, available at the bottom of the clip detail panel.

The edited output is stored as a new child generation rather than overwriting the source. That keeps the original clip, the edit prompt, and the generated alternatives side by side, which allows FR09 to sit alongside FR12 instead of destroying the comparison context.

4.6.2 In-Betweening

In-betweening is the implementation of FR13, and it is exposed directly rather than being hidden inside composition. The user drops a start clip and an end clip into the panel, supplies a prompt, duration, variants, and an optional seed, and the system treats the bridge as the only flexible segment while preserving the two surrounding poses and placements. Both inputs can come from the library or existing `.anim` assets, so transitions can be built around generated or hand-authored motion.

This matters because it separates the transition problem from the rest of the composition pipeline. A user can repair a gap between two clips without first building a full multi-segment plan, and the resulting bridge still comes back through the same asset pipeline as any other generated motion. The same mechanism is also how FR17 is realised when two neighbouring generated segments are merged under an explicit transition prompt inside composition.

4.6.3 Path Editing and Contact-Locking

Path editing and contact-locking are clean-up tools, not part of the reconstruction bridge itself. Path editing is the user-facing implementation of FR11: it lets the user reshape the root trajectory directly in scene space with explicit keyframes. Contact-locking is the post-processing pass that revisits support windows and constrains planted limbs against ground-relative drift. Figure 4.10 shows both controls.

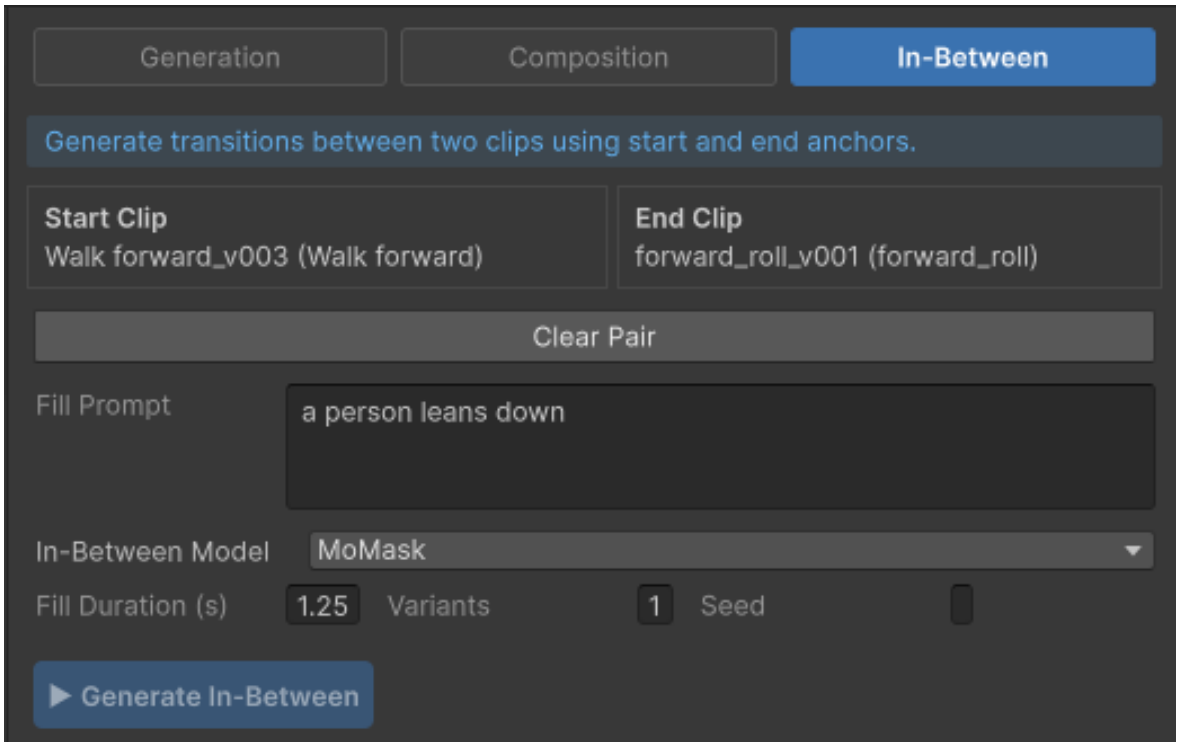


Figure 4.9: The inbetweening view, showing the areas to drag clips in, and choose a prompt, model, duration, and variants.

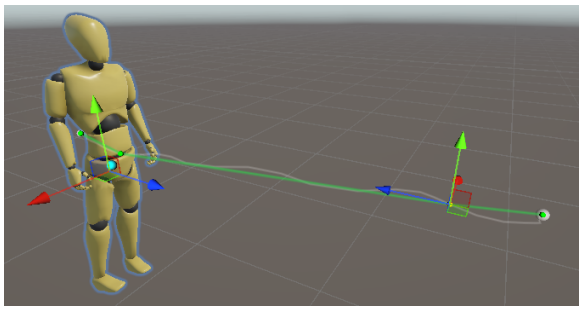
What matters in the implementation is that both tools remain non-destructive. The reference clip, reviewed contact windows, processed outputs, and timestamps are stored alongside the result rather than replacing it. That storage model is what makes FR10 and FR12 real rather than merely stated.

4.7 Multi-Action Composition

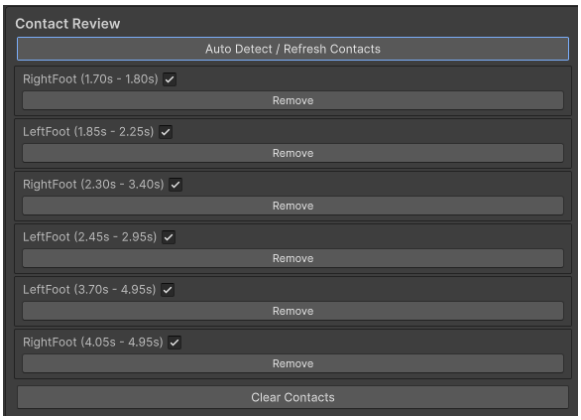
The dashed green path in Figure 4.1 handles long prompts by turning them into editable segments inside the editor, then previewing the assembled result against scene targets (Figures 4.11 and 4.12). This path exists because the integrated generators break down on long multi-action prompts.

4.7.1 Planner Prompt and Output Schema

At the planning stage, a language model turns the user's prompt into an explicit JSON plan with per-segment prompts, durations, model assignments, transitions, and anchors. Appendix B.2 reproduces the full prompt template; the key design point here is simply that the plan remains inspectable and editable rather than becoming a black box.



(a) Keyframe-based path editor.



(b) Contact-locking post-processing pass.

Figure 4.10: Editing tools for spatial correction. (a) The keyframe-based path editor enables direct editing of the root motion. (b) The contact-locking pass detects support windows and constrains selected limbs against ground-relative drift.

4.7.2 Segment Taxonomy

Segments are categorised as locomotion, interaction, or idle. Locomotion accepts up to two anchors because translation is part of its definition, whereas interaction and idle currently use a single anchor. The composition view renders the resulting plan as a colour-coded duration bar.

The taxonomy is not cosmetic. It is the bridge between the planner and the composer because the segment type decides which fields are relevant, how anchors are interpreted, and which placement logic is applied later.

4.7.3 Scene Anchors

The anchor system is the scene-facing half of the composition workflow and the direct implementation of FR18. Adding a `Motion Anchor` component to a game object exposes a named position and facing direction that the planner can reference and the composer can resolve. This is why anchors are limited to the composition view rather than simple generation. Figure 4.12 shows the anchored result that follows the door and person targets from the plan in Figure 4.11.

The important design choice is that anchors are Unity-native objects instead of coordinates hidden inside the planner prompt. That keeps the scene relationship inspectable, editable, and reusable across multiple plans.

4.7.4 Anchor Placement Taxonomy

`MotionGenSpatialComposer` handles placement and assembly once segments have been generated. To realise FR18, several constrained cases had to be handled explicitly rather than collapsed into one opaque solver. Table 4.1 summarises those cases.

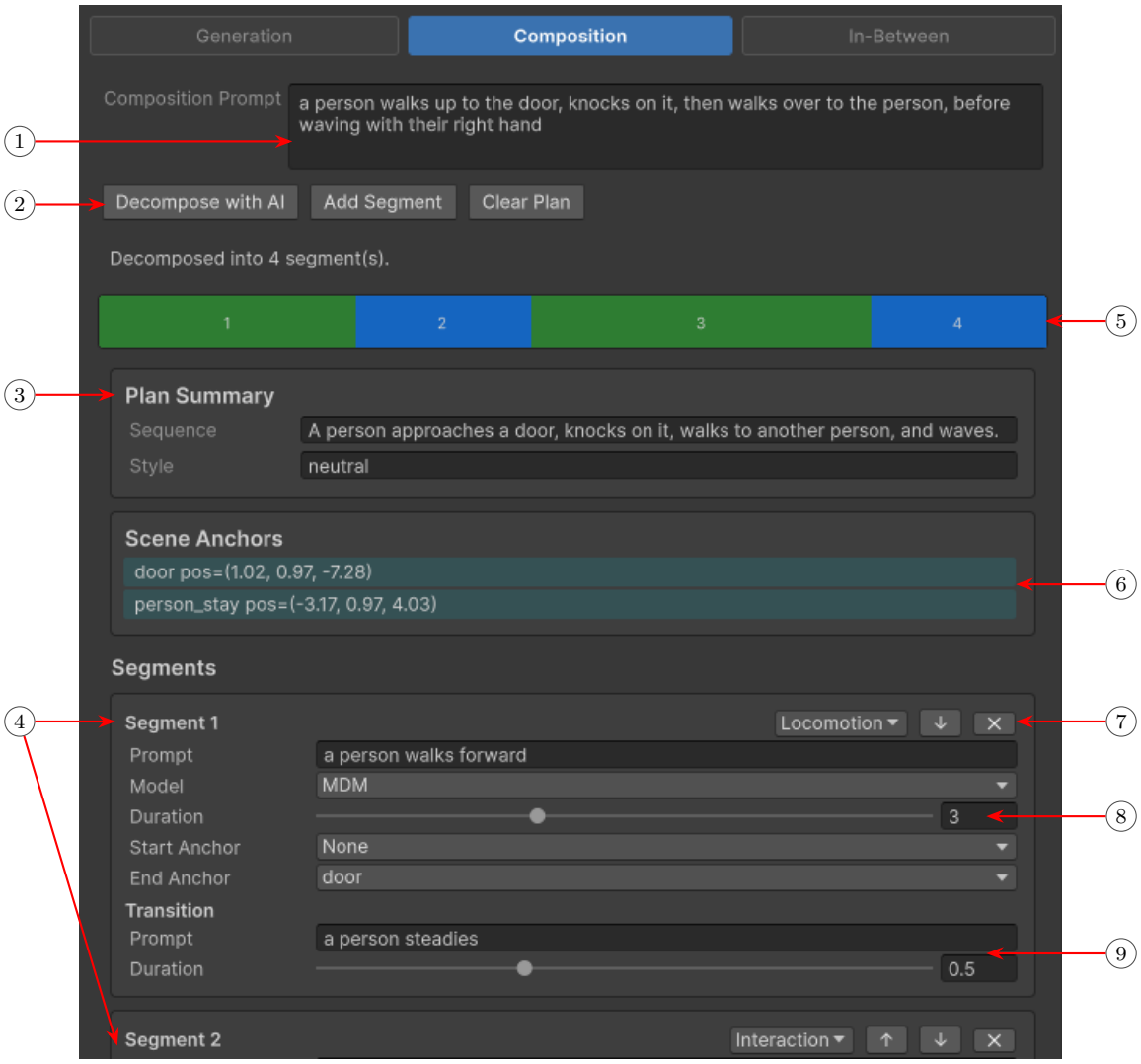


Figure 4.11: Composition view. **1** composition prompt; **2** decomposition controls; **3** plan summary; **4** segments listed in chronological order; **5** segments timeline; **6** scene anchors placed in the scene; **7** segment controls; **8** segment settings; **9** transition settings.

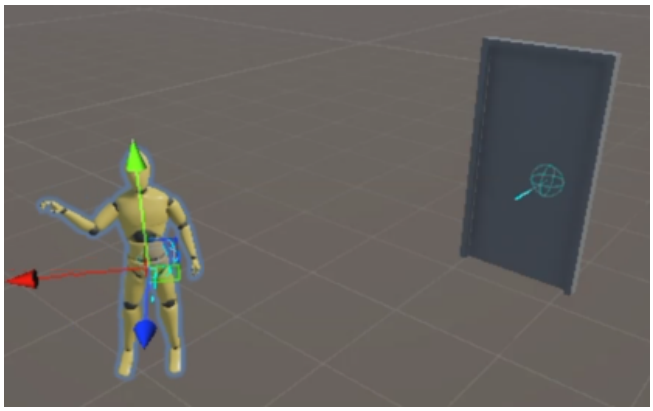


Figure 4.12: Anchored composition result. The door and person targets from Figure 4.11 are resolved into scene anchors, and the generated character follows the resulting motion in scene.

Constraint case	Placement strategy	Displacement handling
No anchors / start anchor only	Place at the origin or supplied start anchor and stitch forward	No additional correction beyond ordinary composition
End anchor only	Stitch backward so the preceding chain terminates at the requested location	No forward constraint after the anchored end
Single segment with start and end anchors	Place at the start anchor and orient to the start-end vector	Small miss: distribute across the segment; moderate miss: confine correction to swing frames from the contact-window detector; large miss: left open
Multiple segments spanning two anchors	Compute the total vector across the constrained span	Apply the same small/moderate/large policy across the affected segments

Table 4.1: Tiered anchor-placement cases implemented by `MotionGenSpatialComposer`.

The rule is to keep displacement small: small misses are distributed across the clip, moderate ones are confined to swing frames identified by the contact-window detector from Section 4.5, and large misses remain open.

4.7.5 Planner-Created Waypoints

Some plans need waypoints that are not already in the scene, such as walking around an obstacle. The planner can create those anchors, after which they go through the same visible composition path as user-placed ones.

4.7.6 Manual Composition Controls

LLM decomposition is optional: users can add, delete, reorder, and edit segments manually, including prompts, models, durations, and anchors. The model and anchor dropdowns are populated from the current project state so the plan remains runnable.

4.7.7 Per-Segment Model Routing

Model assignment inside the planner depends on what is installed and uses simple prompt-family biases: routine locomotion and continuity-sensitive prompts prefer `MDM`, then `MoMask`, then `T2M-GPT`; high-amplitude prompts swap the latter two. The point is not a perfect automatic choice, but visible and editable per-segment routing instead of one hidden global default Athanasiou et al. (2022); Shafir et al. (2024); Barquero et al. (2024).

4.7.8 Auto-Decompose Shortcut

For users who want the composition pipeline without manual setup, the simple-generation panel exposes an *auto-decompose* shortcut that runs decomposition, generation, and assembly in one step. The resulting plan remains browsable afterwards, so the shortcut does not become a black box.

4.8 Local LLM Runtime

The planner path in Figure 4.1 depends on an optional runtime that can produce the JSON segment plan locally. The subsections below cover the chosen model and runtime, the installer that makes them available inside the editor, and the performance work that made the local path usable. Together, they realise NFR06 and support FR14-FR17 when the planner path is enabled.

4.8.1 Planner Model and Runtime

The local planner uses Gemma 4 E2B in GGUF form through `llama.cpp`. Section 2.6 gives the broader justification for that choice and defines the GGUF packaging format and Q4_K_M quantisation used here. The practical point in Chapter 4 is that `llama.cpp` gives the editor something it can launch and manage directly on Windows, which is how NFR06 is met without pushing users toward a mandatory external API.

Performance is still tied to the user's hardware, and on the development machine, the lack of CUDA support kept the local path firmly in the "editor tool" category rather than anything interactive. That constraint shapes the next subsection and the performance work later in this section.

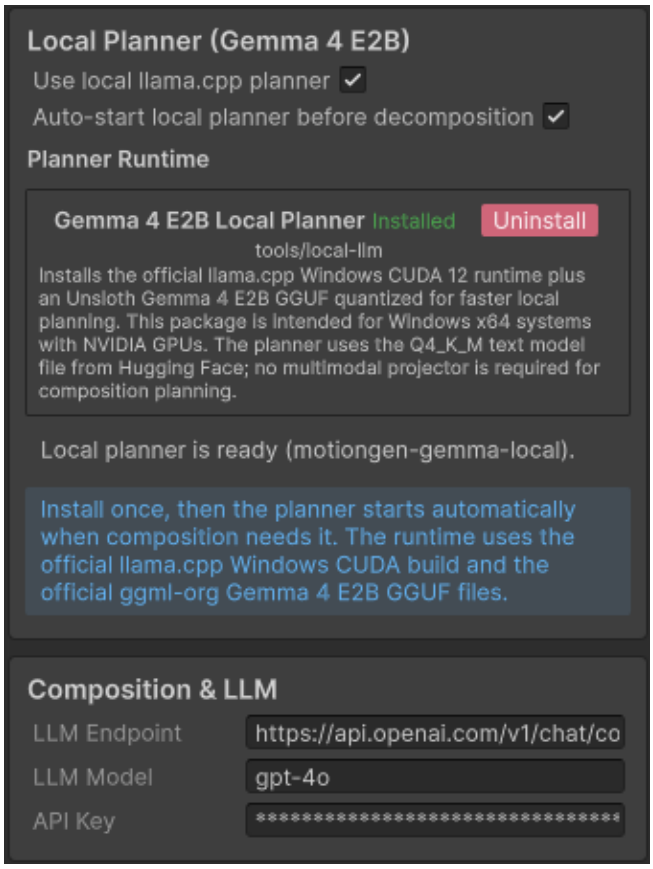


Figure 4.13: The LLM planner management settings, with the install/uninstall button, the enable toggle, and the preload setting for the local model, as well as the configuration for the external API endpoint.

4.8.2 Manifest-Driven Install

The local runtime is installed through the same manifest-driven pattern as the motion models. `local_llm_manifest.json` declares the runtime binary and the GGUF weights, and `MotionGenLocalLlmManager` downloads them on demand and applies the required endpoint and model settings inside the editor. The weights are stored compressed in GitHub releases so that the plugin does not ship them by default.

That symmetry matters. The planner path uses the same install, verify, and enable flow as the motion backends, so adding a local LLM does not invent a second deployment story. From the user's point of view, it behaves like any other optional model install.

4.8.3 Performance Work and Limits

Two performance issues mattered in practice: startup cost and per-request latency. Four configurations were considered. Option A did nothing. Option B preloaded the model into memory between requests. Option C reused the KV-cache, defined in Section 2.6 as the

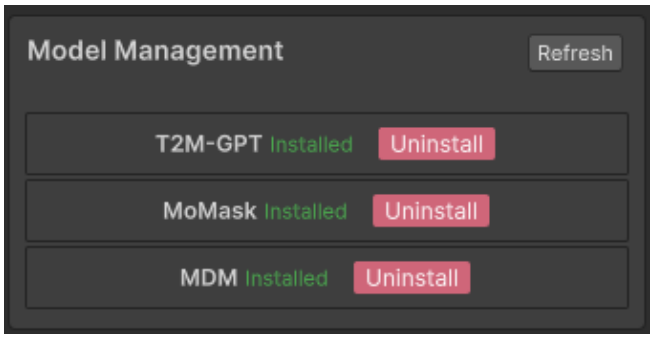


Figure 4.14: The model manager for installing/uninstalling any of the three models.

stored attention state for the fixed prompt prefix. Option D combined both.

Chapter 5 reports the measured outcome. Preload mainly removes startup delay, while KV-cache reuse is what materially improves steady-state latency. That is why the combined configuration became the practical editor-side default: it does not make the planner conversational, but it makes the local path viable enough to support FR14 without breaking NFR06.

4.9 Library and Variant Ranking

A practical takeaway from this project is that batch generation only becomes useful when the outputs can be inspected quickly. The purple endpoint in Figure 4.1 is where generated clips stop being transient replies and become reusable assets. The subsections below cover the persistent library structure, the way generated clips enter Unity's existing project tooling, and the heuristic ranker that was built to speed up comparison. Together, they realise FR05, FR06, FR07, FR08, FR20, and FR21.

4.9.1 Library Structure

The library preserves the comparison state around a request. It is backed by `MotionGenGenerationHistory`, which records sessions, seeds, mirrored BVHs, imported clips, quality scores, and edit lineage across editor sessions. Each generation is grouped as a folder-style entry containing its variants; composed outputs expose their individual segments, and edited or post-processed clips are saved as new entries so the original remains inspectable.

That grouping is what makes FR05 and FR08 tangible. The user is not just reopening a folder full of clips; they are reopening the request context, the alternatives produced for it, and the edits that followed. Badges then surface the generation method, model, and rank without forcing every clip open.

4.9.2 Project-Window Integration

Because every generated clip is written as a `.anim` asset under `Assets/MotionGen/Generated` (or another folder configurable in the simple-generation settings), the outputs are also visible

in Unity's Project window. This is the direct implementation of FR07. An animator can take a generated clip and edit it using Unity's existing animation tools, mix it into Unity animator-controller state machines, or feed it into Timeline sequences in exactly the same way as any hand-crafted clip.

This is the long-term pay-off of the decision in Section 4.3 to output ordinary assets instead of Timeline-only segments. The generated clip enters Unity's native content pipeline and does not need special handling once it has been imported.

4.9.3 Variant Ranking

To speed up comparison, `MotionGenQualityAnalyser` ranks variants using four visible failure cues: foot skating, motion jerk, root drift, and ground penetration. The weights are 0.35, 0.25, 0.25, and 0.15, and the normalised composite is used to sort clips in the library. The idea was simple: surface the cleaner clips first. Chapter 5 shows where that breaks down.

This is the implementation of FR20, and the per-metric scores shown in the clip detail panel are the implementation of FR21. The mechanism worked as a sorting aid in the interface, but Chapter 5 later shows why it could not be treated as a validated preference signal.

4.10 Cross-Cutting Defensive Behaviour

Figure 4.1 shows this section as a band rather than a single stage because the relevant checks apply across the whole request path. The same round-trip is guarded by backend validation, model-availability checks in the editor, background-thread dispatch, and health reporting, so failures become explicit instead of leaking through as freezes or silent no-ops. This is the defensive layer that keeps NFR01, NFR05, and NFR07 intact once the feature surface grows.

At the boundary, `app.py` rejects invalid prompts, frame rates, durations, seeds, and batch sizes before any model call is made. In the editor, model-dependent controls are gated by installation state, the backend status badge surfaces connectivity problems quickly, and long-running requests stay off the main thread. The model-availability checks are particularly important for NFR07: they are what allow a multi-model frontend to stay extensible without turning every missing dependency into an ambiguous runtime failure.

4.11 Implementation Summary

What got built here is not a new motion model. It is the workflow around them. The backend handles local inference, routing, and installation. The Unity side turns returned BVH into usable humanoid clips, stores them as project assets, and lets the user preview, edit, compare, and reuse them in one place. The rest of the tooling exists because first-pass generations are rarely enough on their own: in-betweening, path editing, post-processing, and composition let the user push a clip toward something usable.

Across Sections 4.2–4.10, the implementation realises FR01-FR21 with two important caveats: the large-displacement part of FR18 remains partial, and FR20-FR21 are implemented but later shown to be miscalibrated as preference signals. On the non-functional side, the chapter realises NFR01 and NFR03-NFR09 directly, while NFR02 remains hardware-dependent and is evaluated separately in Chapter 5.

The limits are real and visible. The pipeline is still BVH-centred, Windows-first, and the large-displacement anchor case is still open. The variant ranker also failed its own evaluation. Even with those caveats, the final system is a working Unity authoring loop that was substantial enough to evaluate properly in Chapter 5.

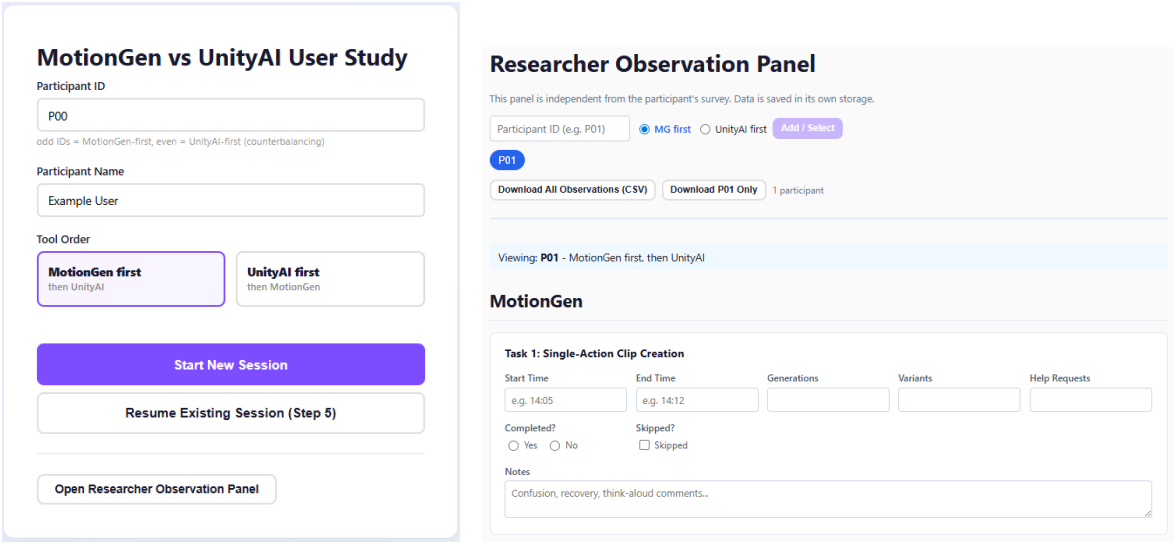
Chapter 5

Evaluation

This chapter documents the methods used to evaluate MotionGen as a Unity-integrated authoring workflow. Four evidence streams were run, each answering a different question and supporting a different design claim from a subjective user study against UnityAI's Muse Animate, an internal model-fit study across the three integrated motion backends, a local planner latency benchmark, and a calibration test of the heuristic variant ranker. Table 5.1 summarises the four strands and the specific role each one plays in Chapter 6.

Study	Question	Measures	Chapter 6 use
Comparative user study	Does MotionGen outperform UnityAI as a Unity authoring workflow?	SUS, raw NASA-TLX, duration, ease, direct comparison, help requests	Establishes the main workflow claim that MotionGen leads on usability, workload, duration, and ease.
Variant-ranker calibration	Does the shipped ranking heuristic reflect user preference?	Chosen visible rank, trust vs revealed behaviour, BVH expression replay	Explains why the shipped ranking signal cannot be treated as a reliable proxy for user preference
Internal model-fit study	Which backend is the strongest routing default per prompt family?	Best-of-N and mean-of-N composite scores plus blind review utility and rank	Supports family-aware routing defaults from combined quantitative and blind-review evidence.
Local planner benchmark	Is local decomposition practical when cached?	Startup, generation-1 end-to-end, steady-state request latency	Justifies preload plus KV cache as the practical editor-side default for local planning.

Table 5.1: Evaluation-strand summary and how each stream feeds Chapter 6.



(a) Welcome screen for the user-study participants.

(b) Researcher observation panel.

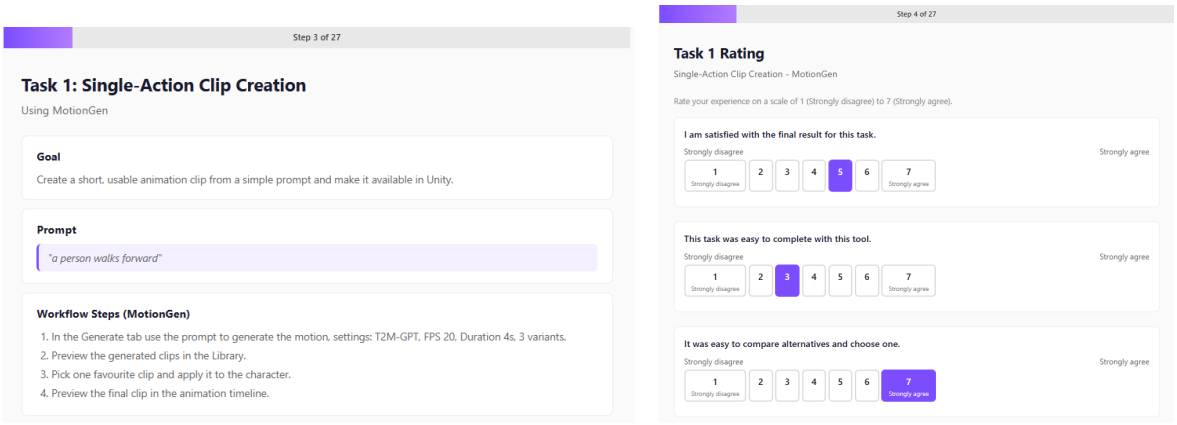
Figure 5.1: Parts of the react-based user-study tool. (a) The welcome page for the user study, defines the participant before they begin. (b) The researcher observation panel for recording times, number of generations/variants, help requests, completion, and notes, such as which variant was selected.

5.1 Comparative User Study

The comparative study used a within-subjects design with ten participants, each completing the same four tasks in MotionGen and UnityAI on the same workstation, Unity project, humanoid rig, and prepared scene. Tool order was counterbalanced (five MotionGen-first, five UnityAI-first), and the researcher provided non-directive support only when asked. Sessions ran for approximately 45 to 80 minutes, and the participant cohort was mixed-skill but student-heavy, with Unity Editor experience distributed as None 1, Beginner 3, Intermediate 4, Advanced 2. With $n = 10$, this design is intended to detect large within–participant effects; smaller differences are treated as consistency signals as opposed to population estimates.

The four shared tasks (Table 5.2) target distinct workflow stress points: simple prompt-to-clip creation, multi-action prompt stress-testing, spatial integration around an object, and nuanced prompt refinement. Each task was completed once with each tool. Due to UnityAI's limited feature set, MotionGen-only features such as segment regeneration were demonstrated separately and kept out of the shared comparative statistics.

The study combined the System Usability Scale and raw NASA-TLX once per tool with metrics: task-level ease, satisfaction, generation attempts, candidate review counts, help requests, and a post-study direct comparison block on a 1–7 scale. Researcher observation logs included measurements such as timing, help requests, and task completion because the participant-side observation fields in the survey export were largely empty.



(a) User-study pre-task brief. (b) User-study post-task questionnaire.

Figure 5.2: Presented screens for each task. (a) The brief given to participants to describe the process and provide the prompt. (b) The three-question questionnaire given after every task to rate to compare the result-satisfaction, completion-ease, and comparison-ease, while it was still fresh in their minds.

5.2 Internal Model-Fit Study

The internal model-fit study evaluates whether different prompt families justify the family-aware routing exposed in Section 4.7. It is build-specific, not a reproduction of published benchmarks. Twenty-one tasks were grouped into seven families: six generation families across all three models, plus one local semantic edit family (MDM and MoMask only). Six seeds were drawn for every task-model pair, resulting in 360 batch samples. Table 5.3 lists the scope.

Each sample was scored using a task-normalised composite. Generation tasks used $C_{\text{gen}} = 0.35\,S + 0.25\,J + 0.25\,D + 0.15\,P$, where S, J, D, and P are foot skating, motion jerk, root drift, and ground penetration. Local semantic edit tasks added a normalised boundary-gap term, $C_{\text{cont}} = 0.25\,S + 0.20\,J + 0.15\,D + 0.10\,P + 0.30\,B$, to weight boundary preservation alongside motion cleanliness at the edited window. Each family is summarised in two ways: mean-of-N (model robustness) and best-of-N (the practical view, because MotionGen exposes multiple candidates to the user).

A blind Unity reviewer pass was then run over a shortlist of the top motion per task using a custom built in-editor blind packet review tool. The reviewer scored each clip on semantic alignment, naturalness, physical plausibility, overall utility, and within-task rank using the same humanoid and review scene. The study targets the output quality of each model for each prompt family. The final routing decision combines the quantitative best-of-N table with reviewer Utility and within-task wins.

5.3 Local Planner Optimisation Benchmark

The planner benchmark measures whether the managed local Gemma decomposition path is practical for real-world workflow use. Four backend configurations were compared: no

Task	Title	One-line probe	Prompt
T1	Single-Action Clip Creation	Create a short, usable clip from a simple prompt and make it available in Unity.	a person walks forward
T2	Multi-Action Clip Creation	Create an accurate multi-beat motion from a written brief.	a person walks forward, waves with their right hand, then does a forward roll
T3	Spatial Object Fit and Interaction Placement	Create and position a clip so the character approaches a pre-placed door and performs an acceptable interaction.	a person walks up to the door and knocks on it
T4	Nuanced Prompt Refinement	Create a single action with specific nuance and refine the prompt until the result is acceptable.	a tired person limps forward slowly, favouring the left leg, with slumped shoulders

Table 5.2: Shared comparative study tasks used in the within-subject evaluation.

Family	Operation	Tasks	Models
Routine locomotion	generate	3	MDM, MoMask, T2M-GPT
Simple interactions / posture	generate	3	MDM, MoMask, T2M-GPT
High-amplitude athletic	generate	3	MDM, MoMask, T2M-GPT
Martial / abrupt	generate	3	MDM, MoMask, T2M-GPT
Expressive / rhythmic	generate	3	MDM, MoMask, T2M-GPT
Compound multi-beat	generate	3	MDM, MoMask, T2M-GPT
Local semantic edit	edit	3	MDM, MoMask

Table 5.3: Scope of the internal model-fit study. T2M-GPT was excluded from local semantic edit because its left-to-right generation cannot be conditioned on a fixed future window.

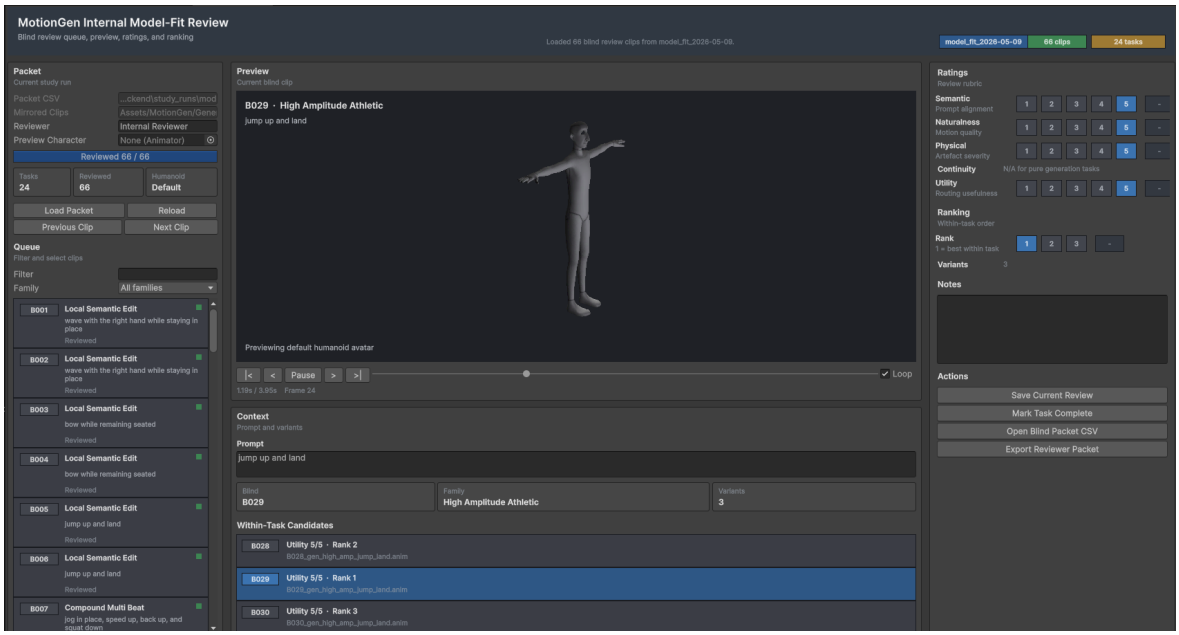


Figure 5.3: The custom-built blind model review tool.

optimisations, model preload only, prompt KV caching only, and preload combined with KV caching. Each configuration was run for two trials of three sequential decomposition requests against the same fixed user prompt and the same planner system prompt. The runtime was `llama-server.exe` hosting Gemma 4 E2B Q4_K_M with an 8192-token context window. The recorded measures are startup mean (warm-up cost), generation-2+ end-to-end mean (steady state), and generation-2+ request-only latency (steady state without startup amortisation). The fixed user prompt was a deliberately compound brief so that the planner had to emit a multi-segment plan rather than a trivial single-action one.

5.4 Variant-Ranker Calibration

The variant-ranker calibration uses two complementary slices to test whether the shipped composite (Section 4.9.3) reflects user preference. The behavioural slice draws on the user-study logs and counts which visible rank participants chose in clean three-variant single-generation cases on T1 and T4, comparing it against a uniform-pick baseline and against self-reported trust in the ranker. The motion-content slice replays the exact cached BVHs from the Unity generation-history asset for those same prompts and computes a within-session expression score over articulation energy, pose spread, and upper-body motion energy; a secondary gait-asymmetry signal is retained for T4 because the prompt explicitly describes a limp. Spearman correlations between rank and expression, paired rank 3 vs. rank-1 contrasts, and the binomial probability of the most expressive variant landing at rank 3 are then reported. The replay is run separately from the user study so that the ranker miscalibration can be

isolated from interface recency effects in a fixed-order list.

5.5 Validity Boundaries

Four boundaries apply throughout. The user-study sample is small and student-heavy, so results are evidence of consistent within-subject preference rather than population estimates. The model-fit study is build-specific and supports routing decisions, not external leaderboard claims, and it measures per-clip output quality. The planner benchmark is hardware- and runtime-configuration-specific. The variant ranker is a hand-designed heuristic, not a learned preference model, and the calibration slice tests whether that specific heuristic is well-calibrated, not whether learned ranking is feasible.

Chapter 6

Results and Discussion

6.1 Headline

MotionGen beat UnityAI on every measure in the comparative study, and the important part is that all four moved together. The supporting studies clarify what that means. Family-aware routing is justified, but high-amplitude athletic prompts exposed a disagreement in which the blind reviewer preferred MoMask over the quantitative MDM lead. The local planner is usable as an editor feature, not as a conversational tool, on non-CUDA hardware. The shipped ranker should not be trusted as-is because it suppresses expressive motion.

6.2 Comparative User Study

Metric	MotionGen	UnityAI	p	Effect (r_{rb})
SUS score	87.75	54.72	0.0117	0.91 (large)
Raw NASA-TLX	3.07	7.53	0.0098	−0.89 (large)
Mean task duration (min)	2.98	3.55	0.0430	−0.71 (large)
Ease rating	6.72	4.80	0.0020	1.00 (large)

Table 6.1: Headline comparative workflow outcomes. All paired comparisons use Wilcoxon signed-rank with rank-biserial effect sizes.

Table 6.1 and Figure 6.1 carry the headline. SUS Brooke (1996) rose from 54.72 to 87.75, comfortably above the usual 68-point average. Raw NASA-TLX Hart and Staveland (1988) more than halved, with the biggest gaps in Mental Demand, Performance, Effort, and Frustration (Figure 6.6). The useful part is that speed and ease improved alongside usability. Participants were not just happier with MotionGen; they were completing the tasks with less effort.

The per-task pattern (Figure 6.2) is consistent except on T4, where durations were nearly identical (2.0 vs 1.9 minutes) and the ease difference fell short of significance. The most

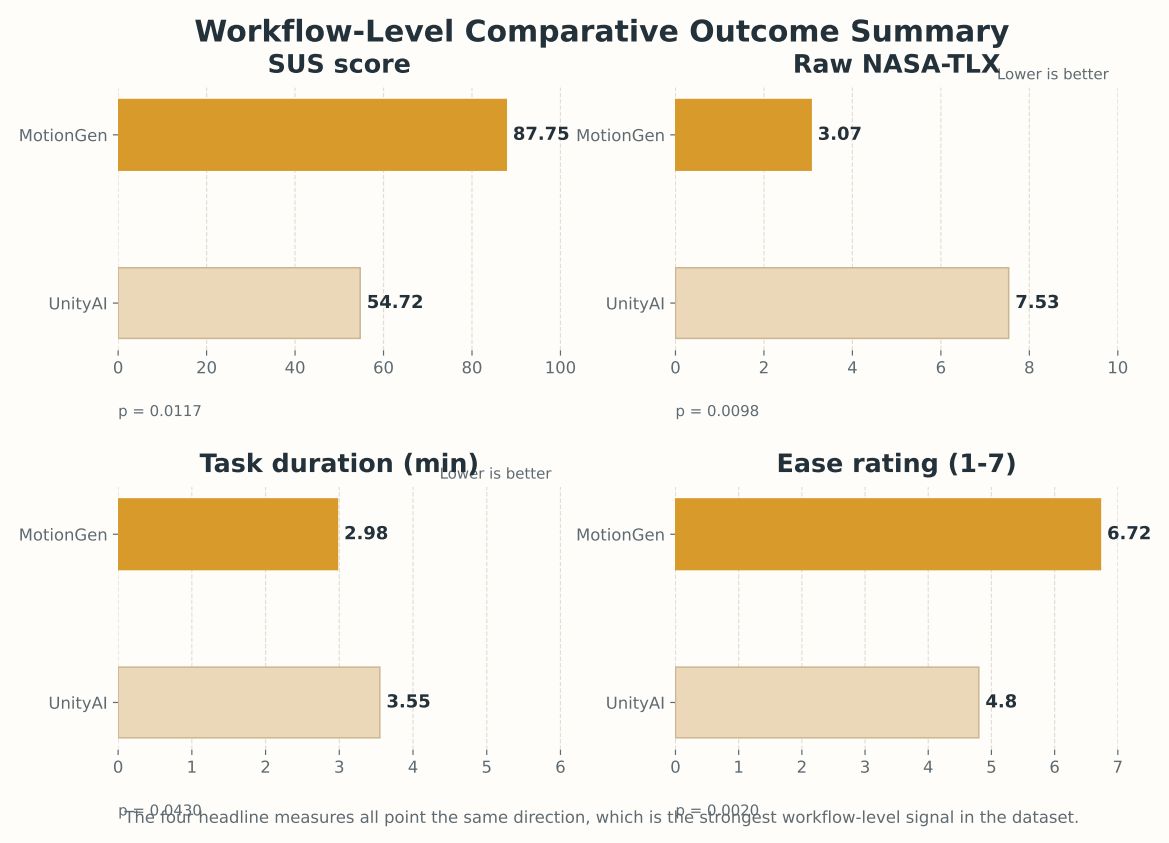


Figure 6.1: Workflow-level comparison across the four headline measures. MotionGen leads on usability, workload, mean task duration, and ease, which is the load-bearing comparative result of the dissertation.

consequential task is T3 (spatial fit), where MotionGen's average duration was a full minute lower (4.5 vs 5.5 minutes) with paired-significant ease. T3 stresses Unity-side placement against a marked target, which is where game engine workflows actually break, so the gap on this task matters more than the average across the four.

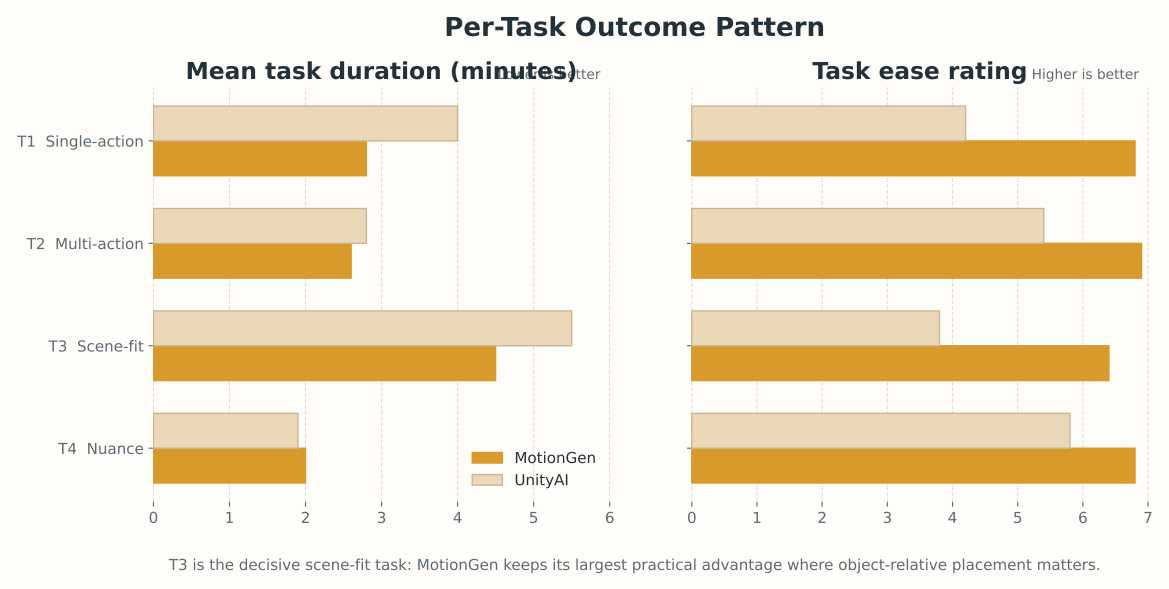


Figure 6.2: Per-task outcome pattern. The scene-fit task (T3) is the most important practical stress test because it exposes Unity-side placement friction rather than pure prompt ideation.

The direct-comparison block (Figure 6.4) explains why the workflow result took this shape. On a 1-7 scale, where 4 is neutral, MotionGen scored 6.7 for control, 6.5 for compound prompt authoring, 6.4 for real-project preference, and 6.1 for scene placement. These items describe specific generational affordances that participants attributed to MotionGen, not just preference proxies.

An illustrative five-frame strip from the T2 multi-action task makes the compound-prompt difference visible (Figure 6.3). MotionGen preserves the requested walk, wave, and forward-roll phases as distinct beats, whereas the UnityAI result remains closer to a generic locomotion sequence.

The mechanism is clearer once help requests are categorised (Figure 6.5). UnityAI accumulated eleven mentions of engine-setup friction against MotionGen's one, and thirteen mentions of output-quality dissatisfaction against MotionGen's two; both tools attracted similar counts of anchor and scene-fit friction. Equal counts are not equivalent to friction: MotionGen's requests advance the artefact (where does this clip attach?), whereas UnityAI's recover the workflow (how do I play it at all?). This is the substantive interpretation of the SUS gap.

Skill stratification refines this (Figure 6.7). Unity Editor fluency mainly reduces the rescue burden for both tools, with a steeper gradient for UnityAI ($\rho = -0.851$, $p = 0.0018$)

(a) UnityAI result for the T2 prompt.

(b) MotionGen result for the same prompt.

Figure 6.3: Illustrative five-frame comparison for the T2 multi-action prompt (*a person walks forward, waves with their right hand, then does a forward roll*). The strip is included as a qualitative example of the compound-prompt authoring difference reported in the direct-comparison ratings, not as a separate scored result.

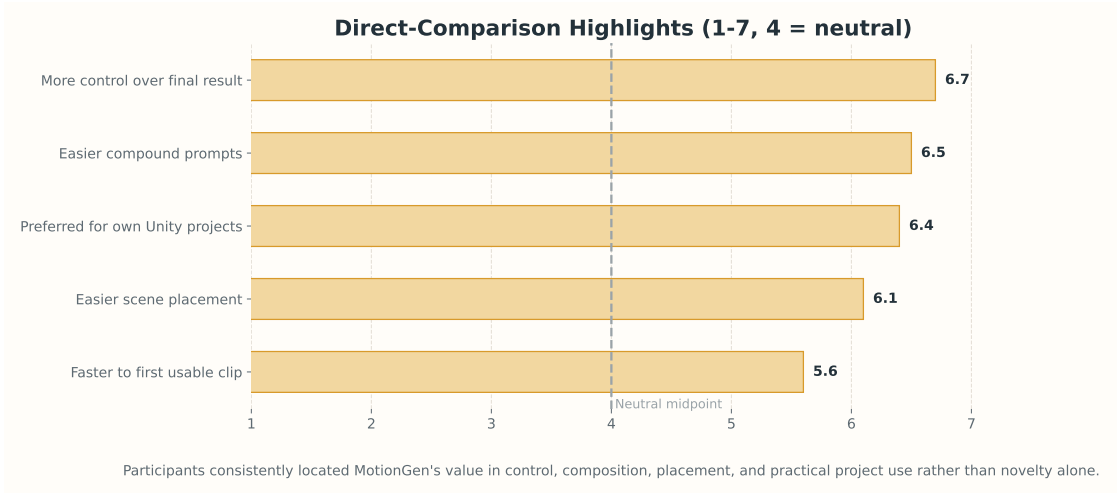


Figure 6.4: Direct-comparison items that most clearly explain why the overall workflow result favoured MotionGen. Scores use a 1-7 scale with 4 as neutral and higher values favouring MotionGen.

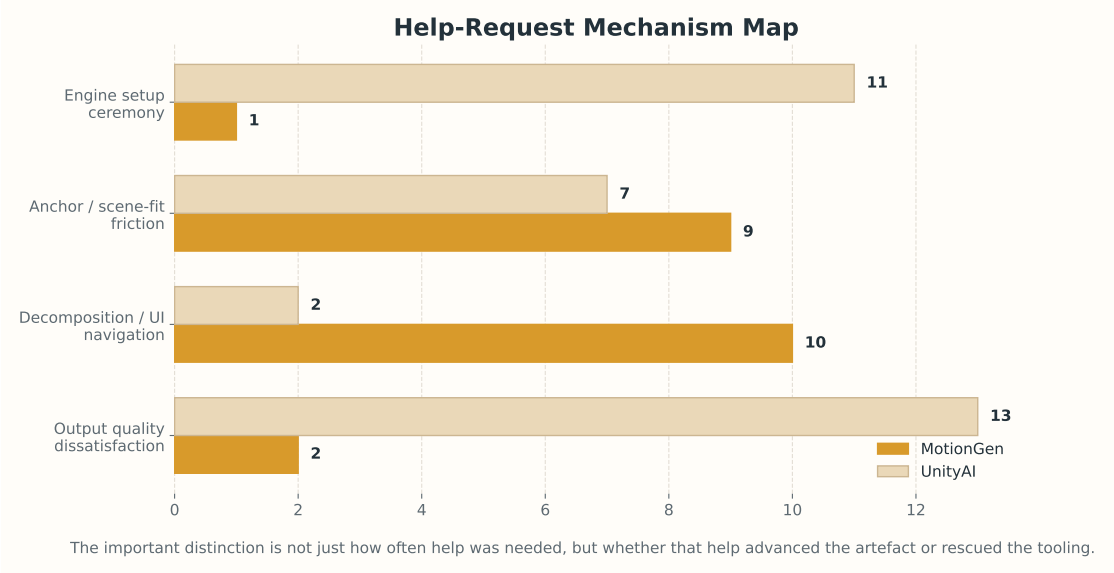


Figure 6.5: Help-request categories across the shared tasks. MotionGen help requests mainly advance scene integration, whereas UnityAI help requests disproportionately recover from engine setup and output dissatisfaction.

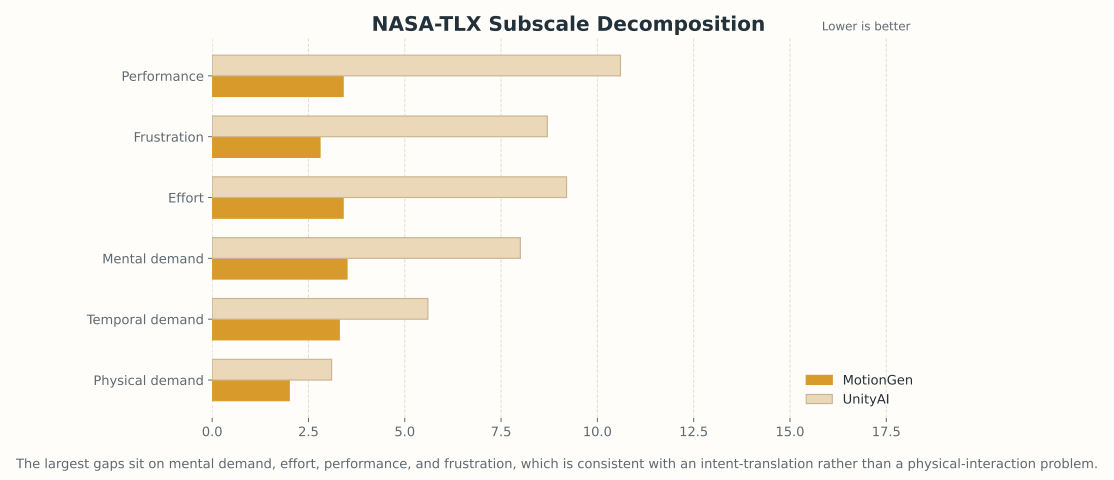


Figure 6.6: NASA-TLX decomposition showing the largest separation on mental demand, performance, effort, and frustration.

than for MotionGen ($\rho = -0.728$, $p = 0.017$). Animation literacy, by contrast, predicts how much value participants extracted from MotionGen specifically ($\rho = 0.78$ on both SUS and ease advantage) and is uncorrelated with Unity skill. Editor fluency sets a floor for both tools; animation literacy sets a ceiling for MotionGen. MotionGen's SUS advantage rises across skill bins (Beginner 23.3, Intermediate 28.1, Advanced 40.0): preference grows with experience rather than vanishing.

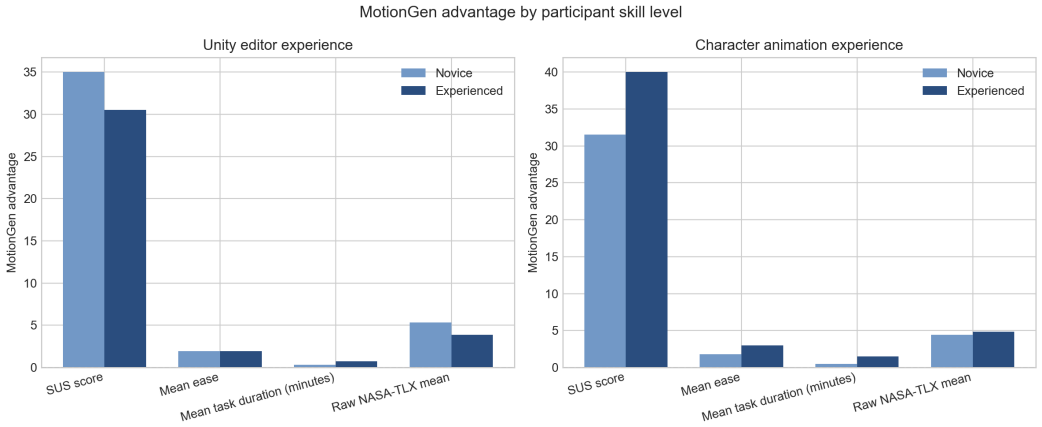


Figure 6.7: Skill-stratified MotionGen advantage. The SUS gap grows from Beginner to Advanced participants rather than disappearing with higher editor fluency.

An order-effect test flagged a duration carry-over at $p = 0.034$, but the direction strengthens rather than weakens the workflow claim: when participants used MotionGen first, its mean duration advantage was only 0.05 minutes, whereas, after UnityAI was used first, it rose to 1.1 minutes.

6.3 Internal Model-Fit Pre-Screen

The combined model-fit summary in Table 6.2 and Figure 6.8 reports the quantitative sweep, the blind reviewer pass, and the resulting per-family reporting position. Three generation families have strong combined evidence and a single routing default: routine locomotion, simple interactions and posture, and martial-arts or abrupt action all converge on MDM, which won best-of-N quantitatively and led on reviewer utility and within-task ranking. The most informative individual result is high-amplitude athletic prompts: the quantitative composite favoured MDM (best-of-N 0.103) but the blind reviewer preferred MoMask on both utility and per-task wins, so the routing default for that family is MoMask. This is direct qualitative evidence that the composite metric is mis-calibrated for at least one family.

The remaining two generation families are mixed. Expressive and rhythmic prompts default to MDM, but reviewer wins were split one-each across all three models, and the case is not decisive on this packet. Compound multi-beat prompts produced split quantitative wins; the reviewer ranked MoMask best overall, and T2M-GPT held the single strongest task

Family	Quantitative signal	Blind-review signal	Reporting position
Routine locomotion	Best-of-N: MDM (0.098); Mean-of-N: MoMask	Utility: MDM; wins: Split (1 each)	MDM strong
Simple interactions / posture	Best-of-N: MDM (0.101); Mean-of-N: MDM	Utility: MDM; wins: MDM 2/3	MDM strong
High-amplitude athletic	Best-of-N: MDM (0.103); Mean-of-N: MoMask	Utility: MoMask; wins: MoMask 2/3	MoMask strong
Martial / abrupt	Best-of-N: MDM (0.077); Mean-of-N: MDM	Utility: MDM; wins: MDM 2/3	MDM strong
Expressive / rhythmic	Best-of-N: MDM (0.125); Mean-of-N: MDM	Utility: MDM; wins: Split (1 each)	MDM mixed
Compound multi-beat	Best-of-N: MDM (0.124); Mean-of-N: MDM	Utility: T2M-GPT; wins: MoMask 2/3	MoMask mixed; T2M-GPT fallback
Local semantic edit	Best-of-N: MDM (0.023); Mean-of-N: MDM	Utility: MDM; wins: MDM 2/3	MDM

Table 6.2: Internal model-fit summary by family combining the quantitative sweep and the completed blind review.

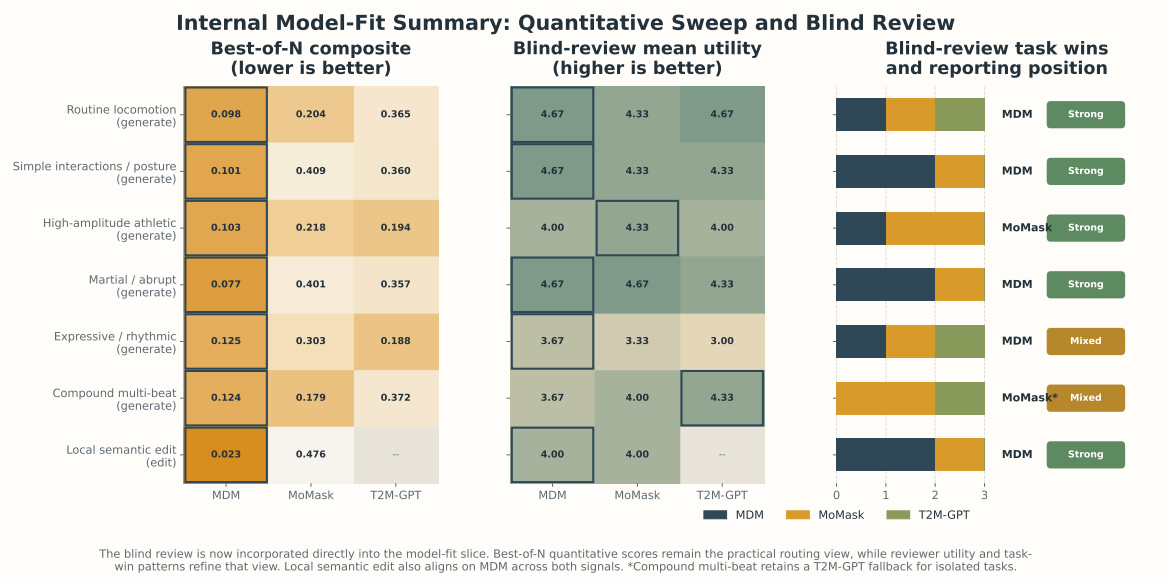


Figure 6.8: Combined model-fit summary. Left: best-of-N quantitative composite scores, with lower values indicating better practical routing under multi-variant selection. Middle: blind-review mean utility. Right: blind-review task wins and the resulting reporting position by family.

utility; the routing default is MoMask, with T2M-GPT as a fallback for compound prompts where the planner expects a discrete autoregressive prior to help. The local semantic edit family also resolved cleanly, favouring MDM on both the composite and the reviewer scores.

A single task makes the disagreement legible. On *do a kick spin to the left*, the lowest best-of-N composite came from T2M-GPT (0.079), with MoMask close behind (0.092); yet the blind reviewer ranked MoMask first and gave it 5/5 for both naturalness and physical plausibility; T2M-GPT was second and MDM third. The point is not that the quantitative winner is useless, but that a cleanliness-weighted composite is not sufficient when amplitude and articulation are the focus of the motion.

A residual caveat applies to the composite-driven half of this evidence. The same metric family that drives the variant ranker (Section 6.5) drives the quantitative composite here. The reviewer agreed with the composite on five of six decisive families but disagreed on high-amplitude athletic prompts, where the composite preferred MDM and the reviewer preferred MoMask. That single disagreement is the most concrete piece of evidence that the composite is a cleanliness signal: where the prompt rewards effort and amplitude rather than stillness, the quantitative best-of-N can rank a less plausible motion above a more useful one.

6.4 Local Planner Latency

The planner benchmark (Table 6.3, Figure 6.9) should be read in hardware context: the development machine had no CUDA-enabled GPU, so `llama.cpp` fell back to CPU execution. The measured minute-scale latency therefore describes the non-CUDA baseline rather than an inherent lower bound of the planner design. In that setting, the benchmark supports the design decision in Section 4.8 to engineer the planner prompt as a long fixed prefix with a small variable suffix, and to make the local runtime opt-in, but only for editor-side decomposition requests that can tolerate roughly a minute once the cache is warm. On CUDA-capable hardware, the same local path should be materially more manageable. KV cache alone cuts steady-state request latency from 130.7 s to 60.5 s (−53.7%) at no startup cost. Preload alone cuts startup from 16.0 s to 5.8 s but barely affects steady state. The combined configuration delivers the best startup (5.0 s) and the best generation-1 end-to-end (124.5 s).

Configuration	Startup (s)	Gen 1 end-to-end (s)	Gen 2+ request (s)	First / steady ratio
No optimisations	15.95	148.13	130.65	1.13×
Preload only	5.78	127.95	123.21	1.04×
KV cache only	5.80	134.17	60.53	2.22×
Preload + KV cache	5.04	124.47	60.90	2.04×

Table 6.3: Local planner latency summary across the four tested backend configurations on the non-CUDA test machine.

The first request remains expensive because the long prefix must be processed once before the cache becomes useful, which validates the architectural decision in Chapter 4: a stable rules-and-schema prefix with a small per-request suffix is what makes prompt caching a

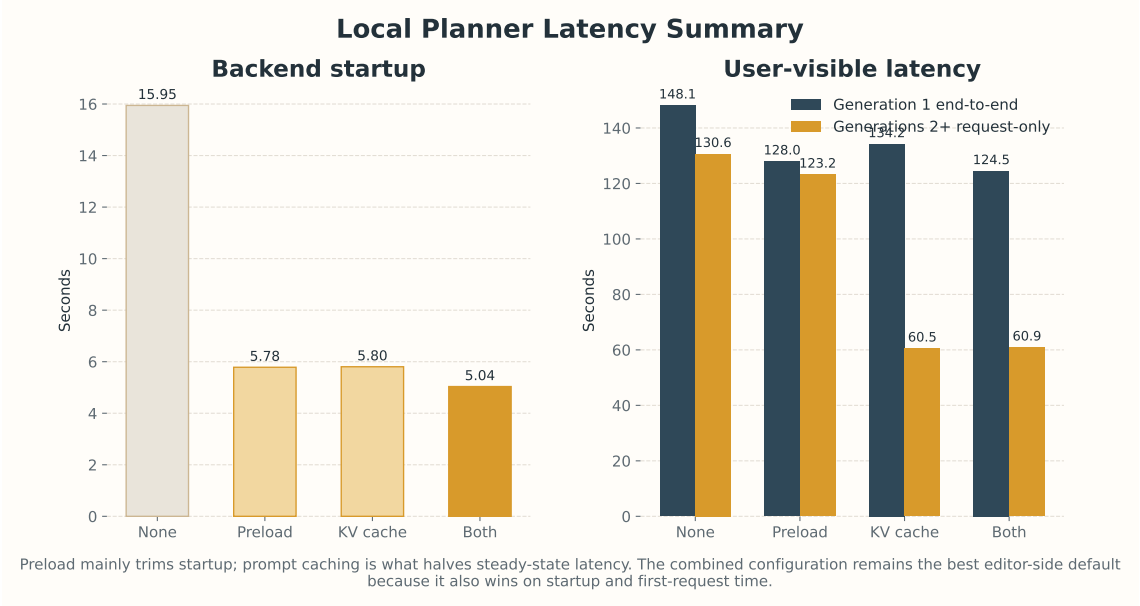


Figure 6.9: Planner latency across the four tested backend configurations on the non-CUDA test machine. Preload is mostly a startup optimisation; prompt caching is what halves steady-state request latency.

viable mitigation. The practical conclusion is therefore conditional. On machines without CUDA, the local planner is acceptable only for occasional use during an authoring session, so minute-scale latency has to be tolerable. On CUDA-backed machines, the same design should be much easier to live with, but it is still not aimed at a real-time conversational interface.

6.5 Variant-Ranking Calibration

The variant-ranking finding is the strongest negative result in the dissertation (Table 6.4). In 67 % (10 of 15) of clean three-variant single-generation cases on T1 and T4, participants chose the visible bottom-ranked variant against a uniform-pick baseline of 33 % (Figure 6.10). Stated trust averaged 5.25 out of 7, but the Spearman correlation between stated trust and the revealed choice ratio was -0.55: participants told the survey that the ranker was helpful, yet scrolled past its top pick. At the point of the original study, two explanations were still viable: either the metric family was implicitly rewarding stillness and therefore suppressing expressive motion, or the interface order was triggering a simpler display-order effect in which participants just favoured the last-presented option.

One participant described the same mechanism directly while choosing the bottom-ranked option: “the more animated variants tend to be at the end, as stillness increases the score with less foot skating, etc.” That matters because it points away from a pure recency explanation; the participant was not merely preferring the last slot but was explicitly identifying why the

lower-ranked clip looked better.

Slice	n	Key result
Behavioral choices	15 clean choices	67% chose visible rank 3; mean chosen rank = 2.53.
Stated vs revealed trust	8 paired cases	Spearman $\rho = -0.553$ between DIAG_MG1 and revealed choice ratio (mean trust = 5.25/7).
BVH replay	17 sessions / 51 variants	Spearman $\rho = 0.635$; the most expressive clip lands at rank 3 in 65% of sessions.

Table 6.4: Variant-ranking calibration summary for the behavioural and replay analyses.

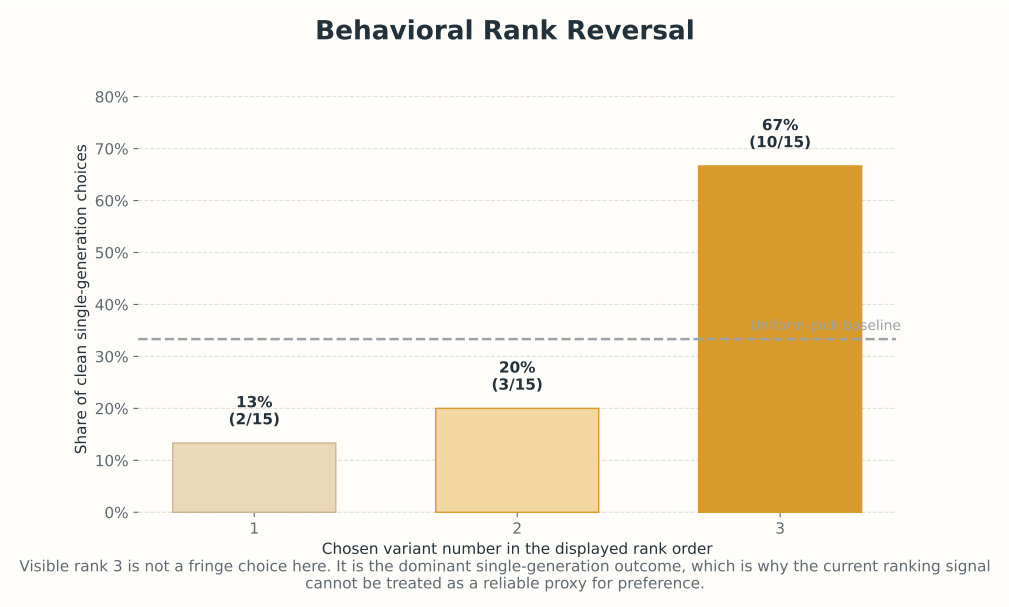


Figure 6.10: Behavioural rank reversal in the MotionGen variant picker. Visible rank 3 is chosen far more often than a uniform baseline would predict.

The behavioural slice cannot distinguish recency bias from ranker bias on its own because the interface always showed ranked outputs in a fixed order. The replay analysis (Figure 6.11) is what resolves that ambiguity in favour of stillness bias. Across 17 replayed sessions and 51 variants drawn from the cached generation history, the Spearman correlation between rank and expression was $\rho = 0.635$ ($p = 5.4 \times 10^{-7}$). The most expressive variant landed at rank 3 in 11 of 17 sessions (65 %, binomial $p = 0.008$), with a rank 3 minus rank 1 gap averaging more than one within-session standard deviation. On T4, the secondary gait-asymmetry signal also rose with worse rank (0.042 to 0.056); the bottom-ranked variants were not noisier; they preserved more of the requested limp. The simplest reading of the combined evidence is, therefore, that the current ranker is stillness-biased first and only secondarily vulnerable to

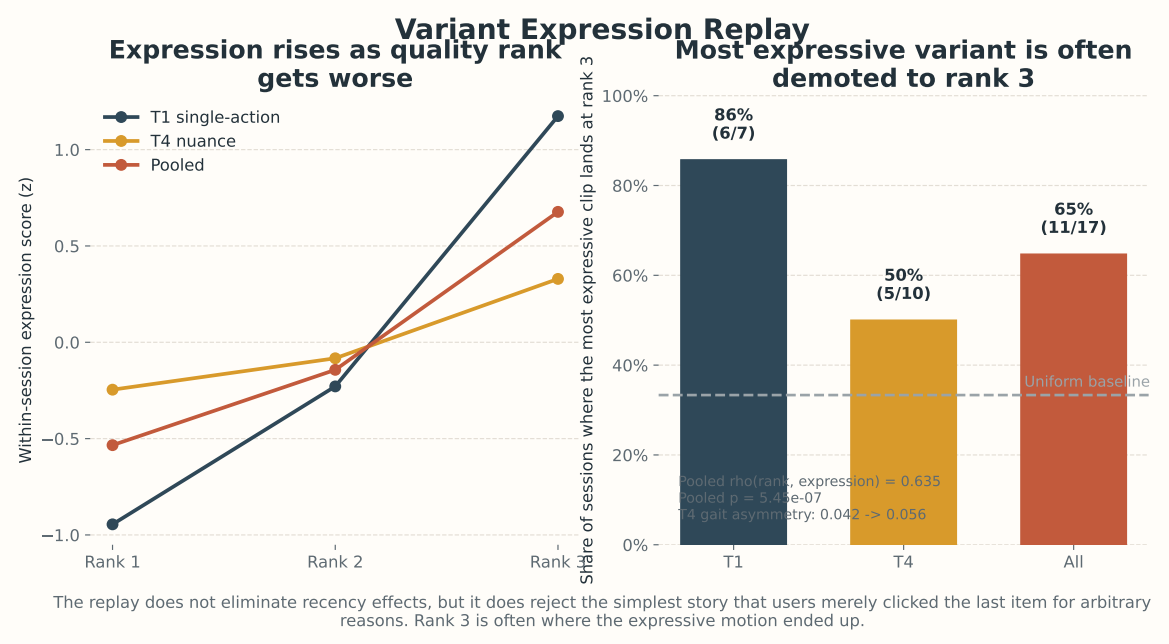


Figure 6.11: Replay analysis showing that expression tends to increase as the internal quality rank gets worse, which supports the stillness-bias interpretation of the ranker failure.

ordinary display-order psychology.

The mechanism follows directly from the metric weights. The composite reduces to four cleanliness signals (foot skating, jerk, drift, penetration), each monotonically preferring smaller and smoother motion. A clip that faithfully renders an expressive prompt scores worse on every term, and the composite has no compensating signal for prompt fidelity. The shipped ranker is therefore a cleanliness heuristic, not a quality one: useful where artefact suppression dominates, harmful when the prompt requires emphasis, asymmetry, or character. The composite must be augmented with a movement- or expressiveness-aware term, or replaced with a prompt-aware profile, before it can be relied on for sorting variants. A clean follow-up would keep the internal rank fixed but shuffle the displayed slot for each variant; if users still prefer the internally lowest-ranked clip when it appears in slot 1 or 2, the remaining human-psychology explanation would be narrowed further.

6.6 Integrated Discussion

The four strands resolve to a workflow claim as opposed to a model claim. The user study shows that previewability, controllability, and scene-fit support convert generation into production value; the model-fit study justifies availability-aware routing; the planner benchmark confines local decomposition to editor-side use on the non-CUDA baseline while indicating that CUDA-backed deployments should be materially more manageable; and the ranker miscalibration shows that cleanliness alone is the wrong proxy when the prompt

rewards expressive amplitude, with the replay evidence making stillness bias the more convincing explanation than simple display-order psychology.

6.7 Limitations

The user-study sample is small ($n = 10$) and student-heavy, so the inferential claims are evidence of consistent within-subject preferences rather than population estimates. The internal model-fit study is build-specific and measures per-clip output quality rather than full workflow performance. The planner benchmark is hardware- and runtime-configuration-specific: it was measured on a machine where `llama.cpp` had no CUDA-enabled GPU available and therefore fell back to CPU execution, so the minute-scale numbers should be read as a non-CUDA baseline rather than a universal expectation. The variant-ranker analysis tests a hand-designed heuristic, and the fixed-order interface in the original sessions means that residual display-order psychology cannot be ruled out completely, even though the replay evidence strongly favours stillness bias as the main failure mode.

Chapter 7

Conclusions

7.1 Summary of Contributions

This dissertation set out to bridge the gap between research-grade text-to-motion models and practical Unity authoring, and the artefact is best evaluated within that workflow framing rather than on raw generation quality. The comparative user study showed that MotionGen outperformed UnityAI's Muse Animate on all four headline measures, with large effect sizes, and the help-request taxonomy explained the mechanism: MotionGen's friction lies in artefact-advancing decisions, while UnityAI's friction lies in workflow recovery. Skill stratification reinforces this; editor fluency reduces the rescue burden for both tools, while animation literacy specifically increases the value participants extract from MotionGen's controllability. The substantive contribution is therefore not generation novelty but a Unity-native authoring workflow that exposes inspectable controls around generation, comparison, refinement, and scene placement.

7.2 What the Supporting Studies Add

The model-fit study, combining the quantitative sweep with a completed blind reviewer pass, supports family-aware routing and resolves five of the seven families on strong combined evidence: MDM is the default for routine locomotion, simple interactions and posture, martial-arts or abrupt action, and local semantic edit; MoMask is the default for high-amplitude athletic; expressive and compound prompts remain mixed. The single most informative result is the reviewer overriding the quantitative MDM lead on high-amplitude athletic prompts in favour of MoMask: this is independent confirmation that the shipped composite metric is a cleanliness signal rather than a quality one, and matches the variant-ranker replay benchmark which showed the same metric family monotonically rewards stillness ($\rho = 0.635$, most expressive clip ranked last in 65 % of sessions).

The local planner benchmark also clarifies the scope compared to merely reporting an optimisation win. Prompt caching halves steady-state request latency and validates the architectural choice of a long fixed prefix with a small variable suffix, but even the best

configuration remains a minute-scale editor operation after warm-up. That is acceptable for occasional decomposition requests inside an authoring workflow, but not for conversational back-and-forth.

7.3 The Variant-Ranker Negative Result

One contribution of the dissertation is negative but still important. I shipped a variant ranker intended to surface the best clip first, and the combined behavioural and replay evidence showed that it does not currently do that. Participants often chose the bottom-ranked variant, and the analysis proved that the composite only rewards terms such as skating, jerk, drift, and penetration, so it systematically punishes expressive motion. The appropriate fix is to redesign the variant ranker with an expression or movement metric built in.

7.4 Reflection

The hardest part of the project was not running the models but making their outputs behave like ordinary Unity animations. The BVH-to-humanoid bridge and the root-stability work consumed more engineering effort than the backend routing itself, and in hindsight, I would prototype retargeting before integrating multiple generators, as that was the real technical risk. The composition pipeline was comparatively straightforward, but fitting generated segments cleanly to anchors without foot sliding is where the remaining open work sits. With another three months, I would spend the time on the large-displacement anchor case, stronger composition-result evidence, and ranker redesign rather than on adding a fourth model.

7.5 Future Work

The first future-work priority is the ranker redesign. It needs either an explicit movement or expression term, or a prompt-aware semantic alignment profile to surface more aligned and coherent motion sooner. The next experiment should then keep the internal rank fixed but shuffle the variant listing to ensure complete detachment from the human psychology bias. A broader and more skill-balanced user study would test whether the controllability advantage generalises beyond a student-heavy cohort. A cloud-deployable backend would solve the Windows-first restriction; alternatively, the project could be ported to both Mac-OS and Linux. With either a cloud-based backend or compatible CUDA-enabled hardware, the project would benefit from re-opening the streaming online generation path that DART represented. The plugin's manifest-driven backend was deliberately engineered to make those extensions additive, so the workflow contribution should outlast the specific models it currently routes to.

7.6 Onward Use

Following the user-study results, my supervisor has begun arranging meetings with the University's software engineering team to prepare MotionGen for release on the Unity Asset Store. That outcome was not an aim of the project, but it is the most direct external indicator I have that the workflow contribution generalises beyond the evaluation setting.

Bibliography

Andreas Aristidou and Joan Lasenby. FABRIK: A fast, iterative solver for the inverse kinematics problem. *Graphical Models*, 73(5), 2011.

Nikos Athanasiou, Mathis Petrovich, Michael J. Black, and Gül Varol. Temporal action composition for 3D humans. In *International Conference on 3D Vision*, 2022. arXiv:2209.04066.

German Barquero, Sergio Escalera, and Cristina Palmero. Seamless human motion composition with blended positional encodings. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2024. arXiv:2402.15509.

John Brooke. SUS: A "quick and dirty" usability scale. In Patrick W. Jordan, Bruce Thomas, Bernard A. Weerdmeester, and Ian L. McClelland, editors, *Usability Evaluation in Industry*, pages 189–194. Taylor and Francis, 1996.

Samuel R. Buss. Introduction to inverse kinematics with jacobian transpose, pseudoinverse and damped least squares methods. Technical report, University of California, San Diego, 2009.

Setareh Cohan, Guy Tevet, Daniele Reda, Xue Bin Peng, and Michiel van de Panne. Flexible motion in-betweening with diffusion models. In *ACM SIGGRAPH 2024 Conference Papers*, 2024. arXiv:2405.11126.

DeepMotion Inc. SayMotion: Text-to-3D animation. `https://www.deepmotion.com/saymotion`, 2024. Open beta.

Gemma Team and Google DeepMind. Gemma 4 model card. `https://ai.google.dev/gemma/docs/core/model_card_4`, 2026. Product and technical documentation.

Georgi Gerganov and contributors. GGUF file format specification. `https://github.com/ggml-org/ggml/blob/master/docs/gguf.md`, 2024a.

Georgi Gerganov and contributors. llama.cpp. `https://github.com/ggml-org/llama.cpp`, 2024b. Software.

Michael Gleicher. Motion path editing. In *Proceedings of the 2001 Symposium on Interactive 3D Graphics*, pages 195–202. ACM, 2001. doi: 10.1145/364338.364400.

Chuan Guo, Shihao Zou, Xinxin Zuo, Sen Wang, Wei Ji, Xingyu Li, and Li Cheng. Generating diverse and natural 3D human motions from text. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2022.

Chuan Guo, Yuxuan Mu, Muhammad Gohar Javed, Sen Wang, and Li Cheng. MoMask: Generative masked modeling of 3D human motions. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2024. arXiv:2312.00063.

Sandra G. Hart and Lowell E. Staveland. Development of NASA-TLX (Task Load Index): Results of empirical and theoretical research. In *Human Mental Workload*, volume 52 of *Advances in Psychology*, pages 139–183. North-Holland, 1988.

Félix G. Harvey, Mike Yurick, Derek Nowrouzezahrai, and Christopher Pal. Robust motion in-betweening. *ACM Transactions on Graphics*, 39(4), 2020. SIGGRAPH 2020.

Daniel Holden, Taku Komura, and Jun Saito. Phase-functioned neural networks for character control. *ACM Transactions on Graphics*, 36(4), 2017.

Biao Jiang, Xin Chen, Wen Liu, Jingyi Yu, Gang Yu, and Tao Chen. MotionGPT: Human motion as a foreign language. In *Advances in Neural Information Processing Systems*, 2023. arXiv:2306.14795.

Biao Jiang, Xin Chen, Chi Zhang, Fukun Yin, Zhuoyuan Li, Gang Yu, and Jiayuan Fan. MotionChain: Conversational motion controllers via multimodal prompts. In *European Conference on Computer Vision*, 2024. arXiv:2404.01700.

Korrawe Karunratanakul, Konpat Preechakul, Supasorn Suwajanakorn, and Siyu Tang. Guided motion diffusion for controllable human motion synthesis. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2023. arXiv:2305.12577.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E. Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with PagedAttention. In *Proceedings of the 29th Symposium on Operating Systems Principles*, 2023. arXiv:2309.06180.

Jehee Lee and Sung Yong Shin. A hierarchical approach to interactive motion editing for human-like figures. In *Proceedings of SIGGRAPH 1999*, 1999.

Matthew Loper, Naureen Mahmood, Javier Romero, Gerard Pons-Moll, and Michael J. Black. SMPL: A skinned multi-person linear model. *ACM Transactions on Graphics*, 34(6), 2015. SIGGRAPH Asia 2015.

Naureen Mahmood, Nima Ghorbani, Nikolaus F. Troje, Gerard Pons-Moll, and Michael J. Black. AMASS: Archive of motion capture as surface shapes. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2019. arXiv:1904.03278.

Michael Meredith and Steve Maddock. Motion capture file formats explained. Technical Report CS-01-11, Department of Computer Science, University of Sheffield, 2001.

Nekki Limited. Cascadeur: AI-assisted keyframe animation. `https://cascadeur.com/`, 2025. Version 2025.3.

Matthias Plappert, Christian Mandery, and Tamim Asfour. The KIT motion-language dataset. *Big Data*, 4(4), 2016.

Abhinanda R. Punnakkal, Arjun Chandrasekaran, Nikos Athanasiou, Alejandra Quirós-Ramírez, and Michael J. Black. BABEL: Bodies, action and behavior with english labels. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2021. arXiv:2106.09696.

Yonatan Shafir, Guy Tevet, Roy Kapon, and Amit H. Bermano. Human motion diffusion as a generative prior. In *International Conference on Learning Representations*, 2024. arXiv:2303.01418.

Guy Tevet, Sigal Raab, Brian Gordon, Yonatan Shafir, Daniel Cohen-Or, and Amit H. Bermano. Human motion diffusion model. In *International Conference on Learning Representations*, 2023. arXiv:2209.14916.

Unity Technologies. Muse animate package documentation. `https://docs.unity3d.com/Packages/com.unity.muse.animate@1.2/manual/index.html`, 2024. com.unity.muse.animate, version 1.2.0-exp.4.

Unity Technologies. Muse animate retirement notice. `https://docs.unity3d.com/Packages/com.unity.muse.animate@1.2/manual/index.html`, 2025. Service retired on 1 October 2025.

Unity Technologies. Unity AI. `https://docs.unity.com/en-us/ai`, 2026. Product documentation for Unity 6.2+.

Li-Chun Tommy Wang and Chih Cheng Chen. A combined optimization method for solving the inverse kinematics problems of mechanical manipulators. *IEEE Transactions on Robotics and Automation*, 7(4), 1991.

Yiming Xie, Varun Jampani, Lei Zhong, Deqing Sun, and Huaizu Jiang. OmniControl: Control any joint at any time for human motion generation. In *International Conference on Learning Representations*, 2024. arXiv:2310.08580.

Jianrong Zhang, Yangsong Zhang, Xiaodong Cun, Yong Zhang, Hongwei Zhao, Hongtao Lu, Xi Shen, and Shan Ying. Generating human motion from textual descriptions with discrete representations. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2023. arXiv:2301.06052.

Kaifeng Zhao, Gen Li, and Siyu Tang. DartControl: A diffusion-based autoregressive motion model for real-time text-driven motion control. In *International Conference on Learning Representations*, 2025. arXiv:2410.05260.

Zixiang Zhou, Yu Wan, and Baoyuan Wang. AvatarGPT: All-in-one framework for motion understanding, planning, generation and beyond. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2024. arXiv:2311.16468.

Appendices

Appendix A

User Study Materials

This appendix records the supporting materials used to conduct the comparative user study described in Section 5.1. The full ethics application, the print-ready participant information sheet, the consent form, signed consent records, the raw participant and researcher exports, and the analysis workbook are retained alongside the dissertation submission package. The structure of each instrument is reproduced below; the full ethics application is also archived at `ethics/Ethics Application.pdf` for reference.

A.1 Ethics Approval

The study was reviewed and approved by the University of Sheffield through the School of Computer Science research ethics route.

Field	Value
Application number	072712
Application title	Interactive 3D Human Motion Generation and Editing Tool in Unity
Module	COM3610 Dissertation Project (Academic Year 2025–26)
Department	Computer Science, University of Sheffield
Programme	BSc Computer Science (FT) – G402
Applicant	Jack Smith (jsmith65@sheffield.ac.uk)
Supervisor / DSC	Dr Zhixiang Chen (zhixiang.chen@sheffield.ac.uk)
Head of Department	Professor Heidi Christensen (heidi.christensen@sheffield.ac.uk)
Ethics contact	researchintegrity@sheffield.ac.uk
Data controller	University of Sheffield (sole)
Academic review completed	Yes
Data collection start	3 April 2026
Anticipated project end	6 May 2026
Application signed	3 April 2026

Table A.1: Summary of the approved ethics application for the comparative user study.

The risk profile was low. The application confirmed: no NHS or social-care involvement,

no human tissue, no clinical trial, no special-category personal data, no vulnerable participants, no highly sensitive topics, no external funding, no off-site fieldwork, and no payment to participants. The departmental risk assessment was recorded as Not Applicable.

A.2 Participant Information Sheet

The two-page Participant Information Sheet was sent to each potential participant before they consented and was reviewed verbally at the start of each session. The print-ready master is stored at `ethics/Participant_Information_Sheet.pdf`; the document submitted to the ethics committee is filed as Document 1165218 (Version 1). The sheet covered, in plain English:

- Study title and purpose – evaluation of the MotionGen plugin's usability against a representative industry-standard text-to-motion workflow.
- Who is conducting and supervising the research, with contact addresses for the researcher, supervisor, Head of Department, and the University's research-ethics inbox.
- What participation involves: a single session of approximately 75–90 minutes, using two animation tools to complete a set of structured tasks, completing questionnaires, and taking part in a short interview.
- Voluntary participation and the right to withdraw at any time without giving a reason, before, during, or after the session, with data securely deleted on withdrawal.
- Separate, explicit consent for screen and audio recording. Participants could decline recording while still taking part.
- Data collected: pre-study demographics, SUS scores, NASA-TLX scores, Likert-scale ratings, open-ended responses, task-performance metrics recorded by the researcher, and (with consent) screen and audio recordings.
- Data handling: pseudonymisation (P01–P10), encrypted university-managed storage, no third-party processors, transcription performed by the researcher only.
- Legal basis under UK GDPR: legitimate interest for scheduling and contact, explicit consent for audio and video recording.
- Data-protection rights: access, rectification, erasure, and the right to complain to the Information Commissioner's Office.
- Retention and deletion timeline (see Section A.8).

A.3 Consent Form

A one-page consent form was signed electronically (digitally signed PDF or secure online form) before any data collection began. The print-ready master is stored at `ethics/Consent_Form.pdf`; the document submitted to the ethics committee is filed as Document 1165219 (Version 1). Each participant initialled separate confirmations covering:

- Having read and understood the Participant Information Sheet.
- Having had the opportunity to ask questions.
- Understanding that participation is voluntary and that participants could withdraw at any time without giving a reason.
- Consent to questionnaire responses and task-performance data being collected and used as described.
- Separate, optional consent for *screen* recording during the task phases.
- Separate, optional consent for *audio* recording during the think-aloud and semi-structured interview phases.
- Understanding of how their data will be stored and used.

The separation of recording consent from study consent ensured that declining recording did not exclude any participant from the study. Signed originals are held in encrypted university-managed storage and are not linked to the pseudonymized research dataset.

A.4 Session Script

Each session followed the fixed protocol set out in the ethics application so that researcher behaviour did not vary between MotionGen-first and UnityAI-first conditions. Sessions were conducted remotely, either through screen-sharing with a pre-configured workstation under the participant's remote control or with a study build of the plugin set up on the participant's own machine plus screen-sharing software.

1. **Welcome and consent** (5 min). The researcher introduced the study, walked through the information sheet, answered questions, obtained signed electronic consent, and reminded the participant of their right to withdraw.
2. **Pre-study questionnaire** (5 min). Demographics block PS1–PS8: age range, role/programme, Unity Editor experience, animation experience, prior generative-AI tool experience, prior motion-generation tool familiarity, tool order (counterbalance), session start time.
3. **First tool briefing** (10 min). Equal-depth guided walkthrough of the first tool's features and interface. Participant observed only, did not interact.

4. **First tool task phase** (20–25 min). The participant completed tasks T1–T4 (Table 5.2). Think-aloud was encouraged but not enforced. The researcher provided non-directive help only when explicitly asked and logged help requests live in the observation form (Section A.6).

5. **First tool questionnaire** (5 min). System Usability Scale (10 items) and the per-tool Likert block PT1–PT6 on a 7-point scale.

6. **Second tool briefing** (10 min). Equivalent walkthrough of the second tool.

7. **Second tool task phase** (20–25 min). Same task set on the second tool.

8. **Second tool questionnaire** (5 min). SUS and PT1–PT6 for the second tool.

9. **Post-study comparative block** (5–10 min). Raw NASA-TLX completed separately for each tool (six 21-point subscales), the eight direct-comparison items DC1–DC8 on a 1–7 scale with 4 as neutral and higher values favouring MotionGen, and the open-ended items OQ1–OQ6.

10. **Semi-structured interview** (5–10 min). The researcher followed a prepared schedule of open questions covering overall impressions, surprises, specific task comparisons, and suggestions; follow-up questions were used where useful. Audio-recorded only with consent.

Tool order was counterbalanced. The four task prompts and the shared scene were identical between tools, so the comparison was fair. The shared workstation, Unity project, humanoid rig, and prepared scene were reset between participants.

Scope reduction relative to the approved application. The approved application listed a broader prospective task pool (including reference-video imitation, multi-character scene population, and a timed-variations exercise) and a target of 10-15 participants. The final study restricted itself to the four shared tasks in Table 5.2 with 10 participants. This is a narrowing within the approved envelope, not an extension beyond it.

A.5 Survey Instrument

The survey was delivered through the standalone React user-study web application introduced in Section 3.4. Table A.2 summarises the instrument by block; full item wording is preserved in the raw participant CSV exports at `user-study/User_Study_Results/Participant_Survey/`.

A.6 Researcher Observation Form

Researcher observations were captured live in a parallel form per participant, exported to `user-study/User_Study_Results/Researcher_Observations/`. The form mirrored the

Block	Items
Background (PS1–PS8)	Age range, programme / role, Unity Editor experience (None/Beginner/Intermediate/Advanced), animation experience (None/Beginner/Intermediate/Advanced), prior tools used, Unity AI familiarity, prompt-tool familiarity, animation confidence (1–5).
Per-task (×4 per tool)	Start time, end time, generations attempted, variants reviewed, help requests, completed (yes/no), skipped, confusion notes, task questions TQ1–TQ3 (ease, satisfaction, confidence) on a 7-point Likert scale.
SUS (×2)	Standard SUS items SUS1–SUS10, 5-point Likert, scored 0–100 per the conventional formula.
NASA-TLX (×2)	Mental demand, physical demand, temporal demand, performance, effort, frustration on a 21-point scale, reported raw (un-weighted).
Post-tool (×2)	PT1–PT6: ease, control, output quality, prompt-translation, scene fit, willingness to use in a real project, 1–7 Likert.
MotionGen-specific (DIAG_MG1–7)	Trust in variant ranker, perceived usefulness of segment regeneration, anchor system, inbetweening, composition, ranker trust calibration. Asked of every participant after the MotionGen block.
Direct comparison (DC1–DC8)	1–7 scale, 4 neutral, higher values favouring MotionGen on speed to first usable clip, ease of comparing alternatives, scene placement accuracy, nuanced-prompt refinement, compound-prompt authoring, control over the final result, lower overall effort, and project preference.
Open questions (OQ1–OQ6)	Free-text follow-ups on most-improved workflow part, most confusing or frustrating part, where UnityAI was clearly stronger, most valuable MotionGen capability, what to change before real-project use, and any other comments.

Table A.2: Survey instrument layout. Full item wording is preserved verbatim in the raw participant CSV exports.

participant timeline and added researcher-only fields that the participant could not be reliably asked to self-report:

- Task-level wall-clock timing, redundant with the participant-side fields so that researcher and participant clocks could be reconciled. As noted in Section 5.1, the participant-side observation fields in the survey export were largely empty, so the researcher logs are the primary source of timing and help-request data used in Chapter 6.
- Help-request events: timestamp, task, free-text description, and the closest matching code from the help-request taxonomy in Section A.7.
- Task completion outcome: completed, partially completed, or skipped, with a free-text reason.
- Critical-incident notes: any moment when the participant abandoned an approach, expressed strong frustration, or discovered an unintended feature.
- Researcher impressions of which generative or scene-fit affordance the participant relied on most heavily.

A.7 Help-Request Taxonomy Codebook

The four-category help-request taxonomy that drives Figure 6.5 is defined as follows. Coding was carried out post-session by the researcher against the observation logs and was not visible to participants.

Category	Interpretation	Examples
Engine-setup ceremony	Workflow recovery	"How do I get this clip to play on the character?", "Where did the file go?", missing Animator controller, missing avatar mask.
Scene-fit / anchor friction	Artefact-advancing	"Where should this segment land relative to the door?", anchor placement, root-trajectory alignment, foot-floor mismatch.
Decomposition / UI navigation	Artefact-advancing	"Which panel exposes segment regeneration?", composition vs simple-generation routing, settings location.
Output-quality dissatisfaction	Workflow recovery	"This motion is not what I asked for", "Can I get a less floppy version?", repeated regeneration with unchanged prompt.

Table A.3: Help-request taxonomy used to categorise researcher observations across both tools.

A help request was logged only when the participant explicitly asked for or visibly waited for researcher input; spontaneous self-talk was not coded. Where a single utterance touched two categories, the dominant category was recorded and the secondary category was noted in free text.

A.8 Data Storage and Retention

Raw exports are stored alongside the dissertation submission under `user-study/User\_Study\_Results/` with the directory structure summarised below. Pseudonymised identifiers (P01–P10) are used throughout; the participant-to-pseudonym mapping was held separately in a password-protected file and was deleted within two weeks of the final session, in line with the commitment made in the ethics application.

- `Participant_Survey/user-study-P{01..10}-*.csv` – one row per participant covering the full survey instrument.
- `Researcher_Observations/researcher-obs-P{01..10}-*.csv` – one row per participant covering the observation form.
- `analysis/` – derived tables and figures (`summary_statistics.csv`, `inferential_statistics.csv`, `help_request_taxonomy.csv`, `participants_clean.csv`, the variant-expression replay outputs, the analysis workbook, and the rendered charts).

Retention timeline (per the approved application).

- Contact details (name, email addresses) are deleted within two weeks of the final session.
- Screen and audio recordings are deleted after transcription is complete, and in any case, no later than 30 July 2026.

Audio recordings. The recordings at `user-study/Audio_Recordings/P{01..10}.mp3` are identifying personal data and are not required to interpret the results reported in Chapter 6, which rely on the researcher summaries already recorded in the participant CSVs rather than verbatim quotations. For submission, they are therefore treated as working materials rather than dissertation artefacts, and any retained local copies are covered by the retention commitment above: they are kept only for transcription checking and deleted after transcription is complete, and in any case, no later than 30 July 2026.

Appendix B

System and Backend Artefacts

This appendix reproduces the load-bearing interfaces of the artefact described in Chapter 4. The intent is to make every contract that the dissertation reasons over directly inspectable, without requiring the reader to clone the repository.

B.1 gRPC Service Contract

The shared contract between the Unity plugin and the python backend is defined in `motion-backend/protos/motion.proto`. The relevant portions are reproduced below; comments not load-bearing for the dissertation have been removed.

```
syntax = "proto3";

package motion;
option csharp_namespace = "Motion";

service MotionService {
  rpc Ping (Empty)                        returns (PingResponse);
  rpc Generate (GenerateRequest)          returns (GenerateReply);
  rpc GenerateBatch (BatchGenerateRequest) returns (BatchGenerateReply);
  rpc Edit (EditRequest)                  returns (EditReply);
  rpc EditBatch (BatchEditRequest)        returns (BatchEditReply);
  rpc Inbetween (InbetweenRequest)        returns (GenerateReply);
  rpc InbetweenBatch (BatchInbetweenRequest) returns (BatchInbetweenReply);
  rpc Compose (ComposeRequest)            returns (ComposeReply);
}

enum MotionFormat { BVH = 0; JSON = 1; }

enum MotionModel {
```

```
  T2M_GPT = 0;
  MOMASK  = 1;
  MDM     = 2;
  reserved 3;          // DART placeholder, see Section 4.2
  reserved "DART";
}

message GenerateRequest {
  string prompt = 1;
  int32 fps = 2;
  float duration_seconds = 3;
  int32 seed = 4;
  MotionFormat format = 5;
  MotionModel  model  = 6;
}

message BatchGenerateRequest {
  string prompt = 1;
  int32 fps = 2;
  float duration_seconds = 3;
  int32 count = 4;
  bool  use_random_seed = 5;
  int32 seed = 6;
  MotionFormat format = 7;
  MotionModel  model  = 8;
}

message MotionJointSequence {
  int32 fps = 1;
  int32 frame_count = 2;
  int32 joint_count = 3;
  repeated float joint_positions = 4;
}

message EditRange { float start_seconds = 1; float end_seconds = 2; }

message EditRequest {
  string prompt = 1;
  int32 fps = 2;
  int32 seed = 3;
  MotionFormat format = 4;
```

```
  MotionModel  model  = 5;
  MotionJointSequence source_motion = 6;
  repeated EditRange edit_ranges = 7;
}

message InbetweenRequest {
  string prompt = 1;
  int32 fps = 2;
  float fill_duration_seconds = 3;
  int32 seed = 4;
  MotionFormat format = 5;
  MotionModel  model  = 6;
  MotionJointSequence start_pose = 7;
  MotionJointSequence end_pose   = 8;
}

message CompositionSegment {
  int32  id = 1;
  string prompt = 2;
  float  duration_seconds = 3;
  int32  seed = 4;
}

message CompositionTransition {
  int32  after_segment_id = 1;
  string prompt = 2;
  float  duration_seconds = 3;
}

message ComposeRequest {
  repeated CompositionSegment    segments    = 1;
  repeated CompositionTransition transitions = 2;
  int32 fps = 3;
  bool  use_random_seed = 4;
  int32 seed = 5;
  MotionFormat format = 6;
  MotionModel  model  = 7;
}
```

B.2 Composition Planner Prompt

The full system prompt used by the local Gemma 4 planner and the optional remote-API planner is reproduced verbatim below. The runtime substitutes {`anchor_dict`} with the live list of scene anchors and {`model_guidance`} with the availability-aware tier block described in Section 4.7. The canonical copy lives at `DissertationPluginPlayground/Assets/MotionGen/Editor/MotionGenCompositionPlannerPrompt.txt`; if the dissertation is read alongside a more recent build, the file at that path is authoritative.

```
You are a motion sequence planner for a character animation tool.

Decompose the user's motion description into atomic segments that can
each be generated independently by a text-to-motion AI model.

SCENE ANCHORS:
{anchor_dict}

MODEL GUIDANCE:
{model_guidance}

RULES:
 1. Each segment describes ONE primary action completeable in 1-8 seconds.
 2. Classify each segment: LOCOMOTION | INTERACTION | IDLE.
 3. Each segment must include 'model' using exactly one of the allowed
    values listed above.
 4. LOCOMOTION segments move the character through space.
 5. INTERACTION and IDLE segments are stationary or near-stationary.
 6. LOCOMOTION: assign anchorEnd, anchorStart, or both when the movement
    is explicitly between anchors.
 7. INTERACTION and IDLE: assign anchorStay when the action is tied to
    an anchor.
 8. ANCHOR CONTINUITY: If a segment ends at or stays at an anchor, and
    the NEXT segment is LOCOMOTION moving away from that location, set
    the next segment's anchorStart to that same anchor.
 9. Between each segment pair, provide a short transition prompt.
10. Segment prompt text must describe only the body action the motion
    model should generate. The motion models have NO knowledge of the
    scene; prompts must be self-contained body-action descriptions.
11. Do NOT include scene-relative phrases or anchor names in segment
    or transition prompts. Spatial placement is handled by anchors.
12. Generic locomotion prompts must state the travel direction
    explicitly, e.g. "a person walks forward".
```

```
13. Prefer simple model-friendly prompts. For INTERACTION segments,
    describe the body motion generically.
14. Compress phrases like "sit down and settle into the chair" into a
    single INTERACTION beat with prompt "a person sits down".
15. IDLE is only for true idle beats (standing, breathing, fidgets).
16. Anchors are optional. Never invent anchor names that are not in
    SCENE ANCHORS or in your own createdAnchors.
17. Re-evaluate model choice per segment using the MODEL GUIDANCE tiers.
18. Record stylistic tone in `styleContext`, not in segment prompts.
19. Use the exact camelCase field names shown in the schema below.
20. Output ONLY valid JSON matching the schema below.
21. WAYPOINT CREATION: When a motion needs intermediate positions not
    covered by existing scene anchors, add entries to `createdAnchors`
    with name, position [x,y,z], and approachDirection [dx,dy,dz].
22. Only create waypoints when intermediate routing is genuinely needed.
23. WAYPOINT PLACEMENT: shift sideways by 1.5 on the non-travel axis for
    "beside", shift +1.5 along the travel axis for "past", and keep at
    least 1.0 unit from the obstacle centre.
24. WAYPOINT APPROACH DIRECTION: approachDirection points from the
    previous stop toward the new waypoint; must not be perpendicular to
    travel.
25. NO TELEPORTATION: segments must form an unbroken spatial chain.
    Insert an intermediate LOCOMOTION segment whenever consecutive
    anchors differ without an explicit transition.
26. TRANSITIONS ARE BLENDS ONLY: transition prompts describe subtle body
    blending and must not reference the actions of adjacent segments.
27. EVERY DESCRIBED ACTION IS A SEGMENT: do not skip or merge actions
    that the user explicitly named.

OUTPUT SCHEMA:
{
  "sequenceSummary": "...",
  "styleContext":    "...",
  "createdAnchors": [
    { "name": "box_wp1",
      "position": [3.0, 0.0, 4.5],
      "approachDirection": [1, 0, 0] }
  ],
  "segments": [
    { "id": 0,
      "type": "LOCOMOTION",
```

```
      "model": "T2M_GPT",
      "prompt": "a person walks forward",
      "durationHintSeconds": 3.0,
      "anchorEnd":   "door",
      "anchorStay":  null,
      "anchorStart": null }
  ],
  "transitions": [
    { "afterSegmentId": 0,
      "prompt": "a person slows to a stop",
      "durationHintSeconds": 0.6 }
  ]
}
```

B.3 Segment Plan JSON Schema

The planner's emitted JSON is parsed into the in-editor segment model by `MotionGenLLMDecompose`. The field semantics are:

- `sequenceSummary` - one-sentence human-readable plan summary, shown above the composition panel.
- `styleContext` - emotional or stylistic descriptor used to bias the variant-ranker tie-breaker, never injected back into per-segment promts.
- `createdAnchors[]` - planner-introduced waypoints with `name`, world-space `position`, and `approachDirection`. Resolved into Unity `Motion Anchor` runtime objects on plan acceptance.
- `segments[]` - ordered list of atomic action units with stable integer `id`, segment `type` (`LOCOMOTION`, `INTERACTION`, `IDLE`), generator `model`, body-action `prompt`, `durationHintSeconds`, and anchor references (`anchorStart`, `anchorEnd`, `anchorStay`).
- `transitions[]` - per-segment boundary bridging hints with `afterSegmentId`, `prompt`, and `durationHintSeconds`. Used only when the inter-segment pose distance exceeds the configured threshold; below the threshold, transitions are skipped, and segments are stitched by direct interpolation.

B.4 Manifest Schemas

Both motion-model and local-LLM installation flows are driven by the same manifest-driven pattern (Section 4.2). The two manifest files share the structure summarised here; the live files (`backend_manifest.json`, `local_llm_manifest.json`) sit under `motion-backend/` and `Dissertation Plugin Playground/Assets/Plugins/MotionGen/` respectively.

Top-level fields.

- `schemaVersion` - integer, incremented on breaking schema changes.
- `components[]` - one entry per installable component (model, runtime binary, or auxiliary asset bundle).

Per-component fields.

- `id` - canonical identifier matching the editor-side enum (`T2M_GPT`, `MOMASK`, `MDM`, `GEMMA4_E2B_Q4_K_M`, `LLAMA_SERVER_WIN`).
- `displayName` - user-facing label shown in the settings pane.
- `installRoot` - relative path under the plugin or backend installation root where the component is unpacked.
- `artifacts[]` - ordered list of downloadable files. Each artefact has `name`, `url` (typically GitHub Releases or Hugging Face), `sha256`, `sizeBytes`, and optional `archive` hint (`none`, `zip`, `tar.gz`).
- `expectedPaths[]` - post-unpack paths the verifier checks for existence, each with an optional `sha256`.
- `requires[]` – optional list of component ids that must be installed first.

Verification. `model_manager.py` (backend side) and `MotionGenLocalLlmManager` (editor side) walk `components[]`, check `expectedPaths`, validate `sha256` where present, and trigger downloads on mismatch. The verifier is idempotent and is the single source of truth for "is this model installed" decisions throughout the rest of the workflow.

B.5 Variant-Ranker Composite Formula

`MotionGenQualityAnalyser` ranks variants in the library using a weighted composite of four per-clip cleanliness metrics. The weights live in `MotionGenEditorSettings` and were set as below before the user study and have not been retuned since; the user-study findings on this heuristic are reported in Section 6.5.

$$C = w_S \tilde{S} + w_J \tilde{J} + w_D \tilde{D} + w_P \tilde{P} \tag{B.1}$$

- $\tilde{S}$, foot skating: mean horizontal foot velocity during detected contact frames, normalised by clip duration. Higher is worse.
- $\tilde{J}$, motion jerk: mean magnitude of the third time derivative of joint positions across the clip, root-mean-square aggregated over joints. Higher is worse.

- $\tilde{D}$, root drift: total horizontal displacement of the root not accounted for by the locomotion prompt, normalised by clip duration. Higher is worse.
- $\tilde{P}$, ground penetration: per-frame negative-Y excursion of foot-contact joints, integrated over frames. Higher is worse.

Each component is min-max normalised within the current batch before weighting, so C is session-relative rather than absolute. The shipped weights are $w_S = 0.35$, $w_J = 0.25$, $w_D = 0.25$, and $w_P = 0.15$. The local-semantic-edit family adds a boundary-gap term $\tilde{B}$ with $w_B = 0.30$ and reweights the cleanliness terms to $0.25, 0.20, 0.15, 0.10$ respectively, as reported in Section 5.2. Lower C is preferred; this preference direction is the source of the stillness bias diagnosed in Section 6.5.

Appendix C

Reproducibility and Environment

This appendix records the precise hardware, software, and model configurations under which the artefact was developed and evaluated. The intent is for a third party to be able to reproduce the headline measurements in Chapter 6 without recourse to the author.

C.1 Development and Evaluation Hardware

All development, the comparative user study, the internal model-fit study, the planner benchmark, and the variant-ranker replay were carried out on the same Windows workstation. The user study reused the same machine for every participant (remote screen-sharing or remote control) to keep latency, screen layout, and Unity build state constant.

Component	Specification
CPU	AMD Ryzen 5 2600, six-core / twelve-thread (Zen+, AM4)
GPU	AMD Radeon RX 580 Series. No CUDA support; this is the development-hardware constraint that ruled out DART in Section 2.2.
RAM	16 GB
Storage	Four drives: 1.8 TB SATA HDD, 932 GB SATA HDD, 466 GB NVMe SSD, 119 GB SATA SSD. The plugin, backend, and model checkpoints reside on the NVMe SSD; raw study exports on the 932 GB HDD.
OS	Windows 11 Home 10.0.26200
Display	ASUS monitor (exact model not recorded).

Table C.1: Evaluation workstation. The CPU-only inference path used for MDM, MoMask, and T2M-GPT is the relevant constraint for the latency numbers in Chapter 6; the RX 580 provides no CUDA acceleration to any component of the pipeline.

C.2 Software Versions

The Python backend ran inside a project-local virtual environment at `motion-backend/env/.venv`; versions below are the resolved installs from that environment. Unity Editor versions are recorded for both the primary build target and the secondary Unity AI evaluation target.

Component	Version
Unity Editor (primary)	6000.3.10f1 (Unity 6.3 LTS)
Unity Editor (Unity AI eval)	6000.5.0b4 (Unity 6.5 Beta) – only used to run Unity AI in the comparative study
.NET target	.NET Standard 2.1 (Unity 6 default)
Python (backend venv)	3.10.11
PyTorch	torch 2.11.0, torchvision 0.26.0, torchaudio 2.11.0
NumPy	1.23.5
SciPy	1.15.3
gRPC	grpcio 1.80.0, grpcio-tools 1.80.0
protobuf	6.33.6
PyTorch Lightning	2.6.1
torchmetrics	1.9.0
spaCy	3.7.4
SMPL-X	0.1.28
CLIP	OpenAI CLIP (git, installed from `github.com/openai/CLIP.git`)
llama.cpp build	b8665, Windows x64, CUDA 12.4 distribution (CPU fallback used on this hardware; the RX 580 is not CUDA-capable)
Unity AI / Muse Animate	Build available during the user-study window 23–30 April 2026
React user-study app	Vite + React + TypeScript; see `user-study/package.json` for pinned versions
Analysis stack	Python 3.10 with pandas, scipy, matplotlib; entry points in `user-study/analysis/` and `report/build_evaluation_assets.py`

Table C.2: Software versions. Pinning Python (3.10.11) and PyTorch (2.11.0) is necessary for the motion-model outputs to be reproducible to within the expected stochastic envelope of the seeds recorded in `motion-backend/study_runs/model_fit_2026-05-09/manifest_used.json`.

Note on the llama.cpp build. The local-LLM manifest pins `b8665`; Section 2.6 mentions `b8775` in the related text. The manifest is authoritative for what the installer fetched and what the planner benchmark in Section 6.4 measured; the body-text reference should read `b8665`.

C.3 Models and Asset Hashes

The motion models and the local language-model weights are downloaded on demand by the manifest-driven installer (Section 4.2, Appendix B.4). The hashes below are reproduced verbatim from the manifest revisions used at evaluation time: `motion-backend/packaging/`

backend_manifest.json (version 0.2.0) and motion-backend/packaging/local_llm_manifest.json (version 0.1.0).

Motion-backend archive hashes. Each motion model is distributed as a single runtime-asset archive hosted on GitHub Releases. The SHA-256 is verified against the archive bytes before extraction. Per-file hashes inside each archive are intentionally left blank in the manifest; the archive hash is the integrity boundary.

Model	Artefact	SHA-256
T2M-GPT (runtime assets)	t2m_gpt-runtime-assets.zip	60aeb433...b8508b5783a9df4d
MoMask (runtime assets)	momask-runtime-assets.zip	5ed2163f...391f5b4c6bdd62f
MDM (runtime assets)	mdm-runtime-assets.zip	5694dd84...3308bca11ba5fcc

Table C.3: Motion-model archive hashes (first 8 / last 16 hex characters of the SHA-256 digest). Full digests are recorded in backend_manifest.json.

- T2M-GPT: 60aeb4330c0aaa8ed2447134817e26ac10cd965b62bc0991b8508b5783a9df4d
- MoMask: 5ed2163fe14588d194f9f0447d3852fc23be1522abc737c66391f5b4c6bdd62f
- MDM: 5694dd84e8a62276cd8e560894071f5e7379224766575530c3308bca11ba5fcc

The HumanML3D feature extractor and the SMPL neutral body model used by MDM (SMPL_NEUTRAL.pkl, J_regressor_extra.npy) are bundled inside the MDM runtime archive, so the MDM archive hash above is their integrity boundary. The plugin verifies each archive's SHA-256 on download and refuses to extract on mismatch.

Local-LLM artefacts. The local-LLM manifest pins the source URLs of the llama.cpp runtime binary, the CUDA support binary, and the GGUF planner weights, but does not currently record SHA-256 digests for these three artefacts (sha256 fields are present and left blank in version 0.1.0 of the manifest). The pinned URLs are reproduced below; integrity is therefore source-pinned rather than hash-pinned at this manifest version.

- llama-server.exe (b8665, Win x64, CUDA 12.4): https://github.com/ggml-org/llama.cpp/releases/download/b8665/llama-b8665-bin-win-cuda-12.4-x64.zip
- CUDA support binaries (b8665): https://github.com/ggml-org/llama.cpp/releases/download/b8665/cudart-llama-bin-win-cuda-12.4-x64.zip
- Gemma 4 E2B Q4_K_M GGUF: https://huggingface.co/unsloth/gemma-4-E2B-it-GGUF/resolve/main/gemma-4-E2B-it-Q4_K_M.gguf

C.4 Install, Build, and Run

The artefact is reproducible on a fresh Windows machine using the steps below. Steps that the editor performs automatically through the manifest-driven installer are marked *(auto)*; the user only ever runs the first two manual steps.

1. Clone the repository: `git clone <repo-url>` (submodules included: `motion-backend`, `user-study`, `Dissertation Plugin Playground`).
2. Open `Dissertation Plugin Playground` in the Unity Editor version recorded in Table C.2 (Unity 6.3 LTS `6000.3.10f1` for the primary build; Unity 6.5 Beta `6000.5.0b4` for the Unity AI comparison only).
3. *(auto)* On first launch, the plugin provisions the Python virtual environment under `motion-backend/env/.venv` from `env/requirements.txt` and installs the pinned dependencies recorded in Table C.2.
4. *(auto)* The user opens the MotionGen settings pane and installs the desired motion models. The installer downloads each archive listed in `backend_manifest.json`, verifies the SHA-256 against Table C.3, and unpacks under `motion-backend/models/`.
5. *(auto, optional)* If local-LLM decomposition is required, the user installs the Gemma 4 runtime through the same settings pane. The installer fetches `llama-server.exe` (b8665), the CUDA support binaries, and the GGUF weights from the pinned URLs in Section C.3, and writes the endpoint configuration into the editor settings. On non-CUDA hardware (such as the RX 580 used here) the CUDA build falls back to CPU inference, which is the path measured in the planner benchmark.
6. *(auto)* The plugin starts the backend on the first generation request and health-checks it through `Ping`. The `Ping` response is the canonical record of the resolved backend version, CUDA availability, and device name on the host.

The user study and model-fit analyses are reproducible by running the scripts under `user-study/analysis/` and `motion-backend/study\_runs/` respectively. `report/build_evaluation_assets` is the canonical entry point for re-rendering every figure in Chapters 5 and 6 from the raw exports.

C.5 Known Reproducibility Caveats

- **Seed reproducibility** for the motion backends depends on PyTorch CUDA/CPU determinism flags, which the published checkpoints do not pin. Outputs are reproducible to within the expected stochastic envelope, not byte-identical.
- **CUDA-less hardware.** The development workstation has an AMD Radeon RX 580, so every PyTorch motion model and the llama.cpp runtime executed on the CPU.

Latency numbers in Section 6.4 are CPU-bound and will improve substantially on a CUDA-capable host without invalidating the relative comparisons between the four planner configurations.

- **Unity AI/Muse Animate** is a cloud service whose behaviour and quality may change over time; the user-study results characterise the version available during the 23 April 2026 to 30 April 2026 window, accessed through Unity 6.5 Beta `6000.5.0b4`, and not the service in general.
- **Local-LLM hashes not pinned.** Version 0.1.0 of `local_llm_manifest.json` pins URLs but not SHA-256 digests for the llama.cpp binary and the Gemma GGUF weight. Integrity is therefore source-pinned only; a future manifest revision should record the digests for full reproducibility.
- **Variant-ranker weights** were fixed before the user study and not retuned; later builds may diverge from the values in Appendix B.5 if the `MotionGenEditorSettings` asset is changed.