

# Multimodal Open-Source Video Gen Testing Plan (Vast GPU)

## 1 Scope and Hard Restrictions

- Models (only these 3):
  - Wan 2.2 TI2V-5B
  - HunyuanVideo-I2V
  - CogVideoX1.5-5B-I2V
- Input: per clip, **text prompt + reference image** (normal quality; not huge; used for story context and inspiration).
- Output: **silent video only** (no audio generation or syncing in scope).
- Target length: **30–60 seconds** final video per model run.
- Resolution: **720p-class** output.
- Clip strategy: generate **short clips** (typically 5–10 seconds) and stitch into 30–60 seconds.
- Continuity: **last-frame chaining** between clips; overlap window at boundaries.
- Prompting: **strict per-clip shot prompts** (shot cards), not vague story-only prompts.
- Concurrency: **one generation job at a time** per GPU instance.

## 2 Prompt Structure

### 2.1 Global Constants (fixed for an entire run)

- Style and realism level
- Era/location and environment constraints
- Lighting and visibility constraints (example: bright daytime; clear visibility)
- Lens language and camera style constraints (example: cinematic; handheld vs stabilized)
- Aspect ratio and composition constraints

## 2.2 Per-Clip Shot Card (changes per clip)

- Subject and key visual elements
- Action and motion direction
- Environment and scene context
- **One camera move max** (pan, dolly, tilt, push-in, orbit)
- Duration target (5–10 seconds)

## 3 Generation Loop (Runs on the Vast Instance)

Let the model generate a temporally coherent **clip** per prompt; do **not** prompt per frame.

1. Build `prompt_i = global constants + shot card_i`.
2. Choose input image:
  - Clip 0 uses the initial reference image.
  - Clip  $i > 0$  uses `last_frame_i-1` as the next input image.
3. Run inference to produce `clip_i.mp4`.
4. Save artifacts for each clip:
  - `clips/clip_i.mp4`
  - `frames/last_frame_i.png` (and optionally a small keyframe set)
  - `logs/log_i.json` containing:
    - full text prompt (global + shot card)
    - seed and all generation params (steps, fps, frames, resolution, CFG, sampler)
    - input image hash and last-frame hash
    - runtime + model id/version + pipeline git commit
5. **Hard-stops (cost guardrails):**
  - Abort if any single clip takes  $> 10$  minutes.
  - Abort if total wall time exceeds  $X \times 10$  minutes, where  $X$  is the number of planned clips.

## 4 Stitching and Export (Runs on the Vast Instance)

- Stitching is done on the GPU instance for speed and deterministic results.
- **Deterministic stitch requirements:** fixed overlap length, fixed transition recipe, fixed encoder settings, fixed naming.
- Optional boundary smoothing: apply motion-bridging (for example RIFE) only at clip boundaries to hide seams.

## 5 Final Download Package (Must Include Everything)

At the end of each model run, download a single compressed bundle to your PC that contains:

- `final/final_stitched.mp4`
- `clips/clip_000.mp4 ... clip_X-1.mp4`
- `frames/last_frame_000.png ... last_frame_X-1.png`
- `logs/log_000.json ... log_X-1.json`
- `manifest.json` with run summary: model id/version, pipeline commit, environment info, stitch settings, clip count  $X$

This ensures you can recreate the final video locally later, using the same clips, last frames, prompts, and parameters.

## 6 Setup Beforehand to Reduce Time and Wasted Spend

### 6.1 Avoid re-downloads

- Use HF cache environment variables (example: `HF_HOME` or `HF_HUB_CACHE`) to keep weights cached between steps.

### 6.2 Pre-bake the environment

- Maintain a pinned setup (Docker image or a reproducible setup script) so a new instance can be ready quickly.

### 6.3 Prefetch models once (download-only)

- Run a download-only stage for all 3 models before running any generation jobs.

### 6.4 Keep the model loaded

- Use one long-lived worker process per run; avoid restarting Python between clips.

## 6.5 Standardize baseline configs

- Lock baseline fps, frames, steps, CFG, sampler per model; change one variable at a time for comparisons.

## 6.6 Make stitching deterministic

- Fixed overlap, fixed transition type, fixed encoder settings, fixed output naming, fixed manifest schema.

## 6.7 Minimize bandwidth

- Generate, chain, and stitch on the instance; download a single bundle at the end.

## 6.8 Hard-stop rules

- Clip timeout:  $> 10$  minutes abort.
- Run timeout:  $> X \times 10$  minutes abort.

# 7 Pipeline Chronology (Brief)

1. **Local:** Pick a 4090 offer and create an instance; then SSH into it.
2. **Instance:** Install system deps and create the Python environment (pinned).
3. **Instance:** Configure HF cache env vars; optionally mount persistent storage if needed.
4. **Instance:** Copy pipeline code to the instance (git clone or scp).
5. **Instance:** Run download-only prefetch for all 3 models.
6. **Instance:** Run a job spec for model A; pipeline generates clips, chains last frames, stitches, and produces the final bundle.
7. **Local:** Download the single bundle to your PC.
8. **Local:** Stop or destroy the instance immediately after verifying artifacts.

# 8 Billing Timeline Reference Diagram

This diagram is included as a visual reference to understand how time can be billed across startup, model loading, active requests, and idle time. Vast.ai bills base rental per second while an instance is active/connected, plus storage and bandwidth metering; see the billing reference for exact definitions.<sup>1</sup>

---

<sup>1</sup>[Vast.ai Billing Documentation](#).

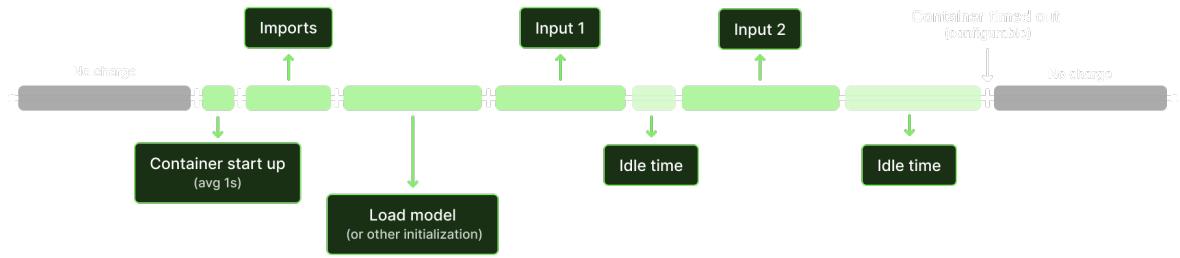


Figure 1: GPU instance billing timeline illustration