PlayTrace: Collecting Verified Human Gameplay with Synchronized Actions for Training AI

A consented, multi-game corpus of screen video paired with keyboard and mouse input on one clock, checked for live play before it is used

The PlayTrace Team

PlayTrace · labs@playtrace.fun

PUBLISHED	VERSION	LENGTH	READING TIME
October 2026	2.0	10 pages	~20 min

ABSTRACT

Models that act, such as game agents, interactive world models and computer-use agents, need training data in which an observation is paired with the action a person took in response. Public video contains the observations but not the actions, so the largest efforts to date have had to infer actions with a model or collect them privately for a single game. We present PlayTrace, a system for collecting this data from ordinary players at scale. A Windows client records one chosen game window as video and logs keyboard and mouse events on the same clock, so every frame can be matched to the inputs around it. Each session is analysed on the server before it is accepted: 30-second windows are labelled as gameplay, menus, idle or non-game content, and a response test checks that the picture actually reacts to the player's inputs, which rejects replayed footage, video playback and scripted input. Accepted sessions carry a per-session quality score and per-window labels, and are stored in a documented, self-describing format with a pseudonymous contributor code. Contributors are rewarded in $TRACE for verified play, and revenue from licensing the data to AI labs buys $TRACE back for future rewards. We describe the capture method, the verification pipeline, the dataset format and its intended uses, the incentive design, and the limitations of the approach.

Keywords: action-labelled video, inverse dynamics, behaviour cloning, world models, human gameplay data, data provenance

1 Introduction

For a decade, progress in AI has followed a simple recipe: more parameters, more compute and more data [1]. For language and images, the data came almost free from the public web. That supply is

now close to exhausted. Epoch AI estimates that the stock of high-quality public human text will be fully used for training somewhere between 2026 and 2032 [2].

At the same time, the frontier is moving from models that *describe* the world to models that *act* in it. An agent that plays a game, a model that simulates an interactive world, or an assistant that operates a computer must learn a mapping from what it observes to what it does next. Text and still images do not contain this mapping. Ordinary video contains the observations but not the actions: a gameplay stream shows the screen, but not which key was pressed or how the mouse moved.

Games are an unusually good source of this missing data. Hundreds of millions of people play on PCs. Every session is a long, continuous record of a person perceiving a rich, fast-changing environment and responding with precise, measurable inputs, across thousands of distinct games, genres and control schemes. Today almost none of it is kept with the inputs attached.

Collecting it from the public is not just an engineering problem. If contributors are rewarded for time, some will try to submit fake time: idle screens, looped video, recordings of other people's streams, or input generated by macros. A corpus collected this way is only useful if each recording can be shown to be a person playing live. PlayTrace is built around that requirement. Our contributions are:

1. A capture client that records a single game window and the player's keyboard and mouse events on one shared clock, without reading or modifying game files (Section 3).
2. A server-side verification pipeline that labels every 30-second window and runs a *response test*, measuring whether the picture moves in step with the player's inputs, to separate live play from replays, media playback and scripted input (Section 4).
3. A documented session format with per-window labels and a per-session quality score, designed for training inverse dynamics models, behaviour cloning and action-conditioned world models (Section 5).
4. An incentive design that rewards verified play rather than raw hours and returns data revenue to contributors through $TRACE (Section 6).

2 Background

2.1 Learning to act from video

The clearest evidence for the value of action labels comes from work that had to reconstruct them. OpenAI's Video PreTraining (VPT) [3] trained a Minecraft agent on about 70,000 hours of online gameplay video. Because that video carried no inputs, the authors first paid contractors to record gameplay *with* keyboard and mouse, trained an inverse dynamics model on it to guess which actions produced each frame, and then used that model to label the larger dataset. The small, action-labelled set was the key that made the large, unlabelled one useful. The same idea, learning an inverse model from a small labelled set and imitating from observation, goes back to behavioural cloning from observation [8].

Genie [4] took the opposite route: it learned an interactive world model from internet platformer videos with no action labels at all, inferring a small set of *latent* actions instead. The result is

controllable, but the latent actions do not correspond to real controls and have to be interpreted after the fact.

2.2 Games as training environments

Recent systems show what becomes possible when real human actions are available. DeepMind's SIMA [5] trained a single agent across many commercial 3D games using human gameplay recorded with keyboard and mouse input. Microsoft's WHAM [6], published in *Nature*, trained a generative world-and-action model on human gameplay data from a single multiplayer game, equivalent to about seven years of continuous play. GameNGen [7] showed that a diffusion model can act as a real-time game engine when it is trained on frames paired with actions, building on the earlier framing of learned world models as environments an agent can be trained inside [9].

These results share one constraint: the action-labelled data was collected privately, for one lab, often for one game. There is no open, growing, multi-game supply of human play with inputs attached.

Table 1. How prior work obtained action data, compared with PlayTrace.

Work	Source of video	Actions	Games
VPT [3]	Online video + paid contractors	Inferred by a model	One
Genie [4]	Online video	Latent, not real controls	One genre
SIMA [5]	Recorded human play	Real keyboard and mouse	Several, private
WHAM [6]	Recorded human play	Real controller input	One
PlayTrace	**Consented player recordings**	**Real keyboard and mouse, same clock**	**Any PC game**

2.3 Why this data is scarce

Players already produce this data every time they play, and much of their footage ends up in public videos that models learn from. The most valuable part of the record, the inputs, is thrown away at the source, because nothing on a normal PC keeps it, and players have no reason to keep it. Collecting it requires consent, a client that records both streams on the same clock, and a reason for people to run that client. A system that asks for consent and rewards complete, genuine recordings can produce better data than scraping, because both sides gain from the recording being real.

3 Data collection

3.1 Capture

The PlayTrace client runs on Windows 10 and 11. The player chooses a single game window from live previews, and only that window is captured. By default the video is scaled so its longer side is at most 1024 px (aspect ratio kept) and encoded as WebM (VP8, falling back to VP9) at 30 frames per second and a target bitrate of about 1 Mbps, roughly 450 MB per hour. The recording is uploaded in

10-second slices during play, so a crash loses at most a few seconds. The client never reads or modifies game files and does not inject into the game process.

While recording, the desktop client logs keyboard and mouse events system-wide through an operating-system input hook: key down and up with the physical key code, button down and up, cursor position (normalised to the screen, at most every 50 ms) and wheel movement. A browser-only capture path also exists; it can only see input inside its own tab, and its sessions are marked as such.

3.2 One clock

The most important property of the data is that video and input share a timeline. Every input event carries t, the number of milliseconds since the first video frame, with paused time removed from both streams. Input captured by the operating-system hook is time-stamped in wall-clock time and converted onto the same timeline using the wall-clock instant of $t = 0$, which is also stored. An event at time t belongs to video frame

$$k = \lfloor\, t \cdot f \,/\, 1000 \,\rfloor \tag{1}$$

where f is the recorded frame rate. This lets a training pipeline cut any clip and recover exactly the inputs that drove it.

3.3 Session format

Each accepted session becomes a self-describing directory in the format `playtrace.session/v1` (Figure 1).

```
<session>/
├─ manifest.json    # games, timings, quality, contributor code
├─ video.webm       # the game window, WebM
├─ inputs.jsonl     # one input event per line, time-ordered
├─ analysis.json    # per-window labels, motion, input counts, checks
└─ frames/          # one 512 px still every 15 s

# inputs.jsonl (t = ms since the first video frame)
{"t":48210,"type":"keydown","code":"KeyW"}
{"t":48231,"type":"mousemove","x":0.5741,"y":0.5963}
{"t":48312,"type":"mousedown","button":0}
{"t":48390,"type":"keyup","code":"KeyW"}
```

Figure 1. Layout of one recorded session and a sample of its input log.

The manifest records which game or games were detected and for which time ranges, the capture source and resolution, the input scope (*system* for the desktop hook, *browser tab* otherwise), the paid and recorded durations, the quality score and any flags. The per-window analysis records the label,

screen state, motion and counts of keys, clicks and mouse moves for every 30-second window, so a buyer can filter at the level of minutes rather than whole sessions.

3.4 Consent and privacy

Every contributor creates an account, is at least 18, and accepts terms that state that their gameplay recordings are licensed to AI labs. The player stays in control of what is recorded:

- **Only the window they pick.** Nothing outside the chosen window is filmed. The recording carries the sound the PC plays, and the player's voice only when they switch the microphone on; it is off by default. PlayTrace never records the webcam.
- **Only while recording.** The player starts, pauses and stops. Keys pressed while recording are logged in any program, so the app asks players to pause before typing anything private.
- **Hands off the game.** Capture uses ordinary Windows screen-capture and input APIs and never reads or changes game files or memory.
- **No names in the data.** Each session carries a stable pseudonymous contributor code derived from a hash of the account. Labs never receive the player's email, name or account.

4 Verifying live play

A recording is only useful for learning to act if the actions in it caused what is on screen. Every session is therefore analysed on the server before it is accepted, and the player's computer is never trusted to report its own duration or content.

4.1 Window labelling

The video is split into 30-second windows. For each window the server measures on-screen motion (the mean brightness difference between frames half a second apart, on a 160 px greyscale copy) and counts input events. One frame every 15 seconds is classified by a vision-language model into one of seven screen states: gameplay, menu, loading, cutscene, someone else's gameplay video being watched, desktop, or other. The model also names the game. Each window then receives one label and a weight w (Table 2).

Table 2. Window labels and weights.

Label	Meaning	Weight w
Active	Live gameplay with motion and input	1.0
Menu	Menus, lobbies, loading screens, cutscenes	0.2
Idle	Screen not changing: paused, away, alt-tabbed	0
Other	Not a game: desktop, browser, black screen, watched video	0
Unverified	Failed the response test (Section 4.2)	0

4.2 The response test

Labels alone cannot tell live play from a recording of play: a replayed match has motion, looks like gameplay, and can be accompanied by random key presses. The response test asks a different question: *does the picture change in step with this player's inputs?* When system-wide input is available, four probes are run on the decoded video:

- **Look.** The whole image shifts within about half a second after the mouse moves, as a camera does in a 3D game.
- **Steer.** A held movement key, or a cursor pushed to the edge of the screen, makes the picture scroll.
- **React.** A key press or click changes part of the screen within roughly a quarter of a second.
- **Point.** A click changes the region under the cursor, as in strategy and point-and-click games.

Each probe is scored against a null distribution built from the same input stream shifted ten or more seconds out of line, giving a z-score. A session with any probe at $z \geq 3$ is treated as responsive. One with plenty of input and motion but every probe below 1.5 is unresponsive, and so is one in which the screen consistently moves *before* the input, which is what happens when someone presses keys along to a video. By default a session with enough input must show positive proof; a weak result is not enough. Windows in a failing session are relabelled *unverified* and excluded. If half of the session or more fails, the whole session is rejected and the reason is recorded (unresponsive, media app, or video playback).

4.3 Other checks

- **Duplicates.** A 64-bit difference hash is computed for one frame every 15 seconds, skipping flat frames, and compared with earlier sessions. If 40% or more of the hashes match another session, the session is rejected.
- **Not a game.** If at least 90% of a session is classified as non-game content, it is rejected.
- **Scripted input.** Human key timing is irregular. If the gaps between key presses are too uniform (coefficient of variation below 0.08 over at least 50 gaps), the session is flagged for review and its weight is halved.
- **Length.** Sessions with under one minute of accepted play are rejected, and at most eight hours per session are counted.
- **Human review.** A random sample of sessions, and any that were flagged (scripted input, weak response, input following the screen, clock mismatch, or accounts linked by device or network), are checked by a person before they are accepted.

4.4 Quality score

Accepted time is the sum of window lengths weighted by their labels:

$$T_{\text{acc}} = m \cdot \Sigma_i \, w_i \cdot d_i \tag{2}$$

where d_i is the length of window i in seconds, w_i its weight, and m a penalty multiplier (1 by default, 0.5 if scripted input is detected). The quality score $q = T_{\text{acc}} / T_{\text{recorded}}$ lies between 0 and 1 and is stored in the manifest. Together with the per-window labels it lets buyers keep only the active minutes of high-scoring sessions.

Worked example. A 1 h 30 min session contains 1 h 20 min of active play, 6 min of menus and 4 min idle, and passes the response test. Then $T_{\text{acc}} = 4{,}800 + 0.2 \times 360 + 0 = 4{,}872$ s and $q \approx 0.90$. For training, the 80 active minutes are usable directly; the menu minutes are kept but labelled.

5 The dataset

5.1 What one hour contains

One accepted hour is about 108,000 video frames at 30 fps, each aligned to the keys held, buttons pressed and cursor position at that moment, plus 120 labelled windows and 240 still frames with a detected game and screen state. Because sessions come from many players and many games, the corpus covers a wide range of genres (shooters, racing, strategy, open-world and others), control schemes, skill levels and hardware, which single-game collections cannot.

5.2 Data packs

Sessions are grouped into data packs for licensing. A pack is selected by game or genre, minimum quality score, input scope and time range, and contains for each session the video, the input log, the analysis and the manifest. Contributor codes are kept so that a buyer can split training and evaluation data by player, but they cannot be linked back to a person. Tooling to assemble packs from stored sessions is in development; the per-session format in Section 3.3 is the interchange format and is available now.

5.3 Intended uses

- **Inverse dynamics models.** Predict the action between two frames, then use the model to label much larger amounts of unlabelled public video, as VPT did [3].
- **Behaviour cloning and agents.** Train policies that map pixels to keyboard and mouse actions, across many games rather than one [5].
- **Action-conditioned world models.** Learn how a game world responds to input, the setting of WHAM and GameNGen [6, 7], with real controls instead of latent ones [4].
- **Human reaction and timing.** Study how quickly and how precisely people respond to what they see, using the exact time offsets between frames and inputs.
- **Evaluation.** Hold out players or games to measure how well an agent or world model generalises to people and titles it has not seen.

5.4 Why players rather than contractors

Paid contractors produce clean data, but they play the games they are assigned, for the hours they are paid, at a cost that limits scale. People playing their own games choose what they enjoy, play the way they normally do, and spread across far more titles. The verification in Section 4 is what makes

this open collection usable: it replaces the trust that a contractor relationship provides with measurements on every session.

6 Contributor rewards

6.1 Reward function

Contributors are rewarded in $TRACE for accepted time. Let U be the accepted time the player has already had in the previous 24 hours. Counting on from U, the part of T_{acc} that stays below F hours is T_{full}, the part between F and C hours is T_{reduced}, and anything past C is not rewarded:

$$R = r \cdot (T_{\text{full}} + \rho \cdot T_{\text{reduced}}) \,/\, 3600 \;\; \$\text{TRACE} \tag{3}$$

with base rate $r = 100$, $F = 1$ h, $C = 6$ h and $\rho = 0.2$ at the time of writing. The current values are always shown in the app. In the worked example of Section 4.4, with no earlier play that day, the first 3,600 of the 4,872 accepted seconds earn the full rate and the other 1,272 the reduced rate, so $R = 100 \cdot (3{,}600 + 0.2 \times 1{,}272) \,/\, 3{,}600 \approx 107$ $TRACE.

6.2 Why a daily rhythm

The schedule rewards playing a little on most days more than one long session, for data reasons as much as fairness. Long, tired sessions produce repetitive, lower-quality data, while many shorter sessions across more days and games give a more varied corpus. The daily ceiling also limits what any single abuser can extract. New rewards are held for seven days before they can be paid out, so that human review can still remove them. Players also earn for community events and for inviting others, which grows the contributor base; these bonuses are paid from the same player allocation.

6.3 The incentive loop

The system is a closed loop with four steps (Figure 2). Players record verified play and are rewarded in $TRACE. Recordings are packaged into data packs and licensed to AI labs. Revenue from those licences buys $TRACE on the market. Bought-back tokens fund future player rewards.



Figure 2. The PlayTrace incentive loop.

Data contracts take time to sign, while contributors need to be rewarded from the first day. The model therefore has two phases. In the first, the token launch funds development, liquidity and early rewards while the first verified corpus is built. In the second, recurring data revenue replaces

launch funding as the source of buybacks. The loop is sustainable only if the second phase arrives; Section 7 discusses what happens if it does not.

6.4 Token supply

Table 3. $TRACE allocation. Total supply is fixed at 10,000,000,000.

Allocation	Tokens	Share	Purpose
Player rewards	5,000,000,000	50%	Rewards for verified play
Launch sale	5,000,000,000	50%	Funds the app and liquidity

At the base rate r, the player allocation covers at least 50,000,000 hours of verified play before buybacks are needed. Balances are currently kept in an internal ledger and will be paid out on-chain once on-chain payouts go live.

7 Limitations and risks

- **Action space.** Only keyboard and mouse are recorded; controller input is not yet supported. Mouse movement is logged as cursor position at up to 20 Hz rather than raw relative motion, which under-describes fast camera turns in games that lock the cursor.
- **Input scope.** The response test needs system-wide input from the desktop client. Browser-only captures have weaker proof of live play and are marked so buyers can exclude them.
- **Verification is imperfect.** The classifier and motion checks can mislabel unusual games, and determined abusers will look for gaps. Thresholds are settings, not constants, and will change as abuse is observed.
- **Dataset bias.** Contributors are self-selected adult Windows players, so the data over-represents some games, skill levels and regions.
- **Sensitive input.** Keys pressed while recording are logged in any program. Players are warned to pause before typing anything private, but mistakes can happen.
- **Content rights.** Recorded games are the intellectual property of their publishers. Licensing terms for commercial use of footage vary and must be respected when packaging data.
- **Demand is not guaranteed.** The loop depends on labs licensing the data. If data revenue is small, buybacks are small, and rewards rely on the fixed player allocation alone.
- **Token risk.** $TRACE is a memecoin. It carries no promise of profit, yield or return and can go to zero. Reward rates and rules may change. Nothing in this paper is financial advice.

8 Roadmap

When	Milestone
Q4 2026 NOW	Windows client with system-wide input, server verification and response test live. Session format v1 published to labs. $TRACE launches with liquidity.
Q1 2027	First data packs licensed. Quality tiers per game. First buybacks from data revenue.
2027	Raw mouse motion and controller input. Recurring lab contracts. Public revenue and buyback reports. More platforms.

9 Conclusion

AI is running out of passive data and moving toward models that act. The data those models need, human perception paired with human action, is produced constantly by people playing games and is almost entirely discarded. PlayTrace captures it with consent, on one clock, across any PC game, and verifies on the server that each recording is a person playing live before it is accepted. The result is an action-labelled corpus that grows with its contributors and returns its revenue to them through $TRACE. If this data is as valuable as recent research suggests, players become paid contributors to the next generation of AI rather than unpaid sources of it.

References

1. J. Kaplan et al. Scaling laws for neural language models. *arXiv:2001.08361*, 2020.
2. P. Villalobos et al. Will we run out of data? Limits of LLM scaling based on human-generated data. *ICML*, 2024. arXiv:2211.04325.
3. B. Baker et al. Video PreTraining (VPT): Learning to act by watching unlabeled online videos. *NeurIPS*, 2022.
4. J. Bruce et al. Genie: Generative interactive environments. *ICML*, 2024.
5. SIMA Team. Scaling instructable agents across many simulated worlds. *arXiv:2404.10179*, 2024.
6. A. Kanervisto et al. World and human action models towards gameplay ideation. *Nature*, 2025.
7. D. Valevski et al. Diffusion models are real-time game engines. *arXiv:2408.14837*, 2024.
8. F. Torabi, G. Warnell and P. Stone. Behavioral cloning from observation. *IJCAI*, 2018. arXiv:1805.01954.
9. D. Ha and J. Schmidhuber. World models. *arXiv:1803.10122*, 2018.