

The Home That Grades Itself

A household simulator that lives daily life against a fresh build, files replayable failures, and sits behind a boundary the automated developer cannot edit.

Pieter Meyer · Elysium Labs · Zürich, Switzerland

ELY-TR-2026-02 · Version 1.4 · First published 11 July 2026 · Revised 11 October 2026

Self-published technical report · not peer reviewed

Abstract

A home assistant can pass every test in its repository and still fail the people who live with it. Conventional tests certify code against a specification an engineer wrote down. They cannot report that a resident was cold this morning, or that a stated preference was forgotten by the afternoon. Elysium narrows this measurement gap with a second system that lives outside the product: a language-model-driven household simulator that conducts simulated daily life against a fresh build of the product, judges its conversations and device outcomes against fixed rules, and files self-contained, replayable tickets where the home failed a genuine human need. The tickets drive a development loop in which an AI developer proposes fixes, independent checks and a human review gate them, and the same simulator grades the result again.

The design depends most on an administrative boundary. The grader lives in its own repository, a required check rejects any product change that imports it, and the automated developer has no write access to it. We describe the method, the verdict rule, the boundary and the capabilities that followed from the simulator's tickets, report what the loop has produced in numbers, and set out its known limitations. We believe the pattern generalizes: it is a small organization of specialized AI systems arranged so that they hold one another accountable, with a human at the final gate.

1. The measurement gap

Software testing answers the question "does the system do what we specified?" For a home assistant, the question that matters is different: "did the home serve the people in it?" The two come apart constantly. A thermostat API can return 200 on every call while the household's stated preference is silently forgotten between conversations. A lighting system can pass its unit suite while a resident walks into a dark room the house knew was occupied. Nothing in a conventional pipeline is positioned to notice either failure, because nothing in it uses the product the way a resident does, day after day, with needs that no test case lists.

The standard industrial answer is user research and telemetry. Both are valuable and both arrive late, after real residents have been failed, and telemetry in particular sits awkwardly with a privacy-first product that deliberately collects little. We wanted a source of resident-shaped failure signals that is available before and between real deployments and does not depend on watching anyone.

2. Method: a simulated household against the product

The simulator maintains a household of language-model personas [1] with persistent traits, routines, moods and needs across a simulated day: waking, leaving, returning, cooking, hosting a guest, falling ill, going to sleep. The simulated residents use the interfaces a household uses. They chat with the assistant, call the same API the app calls, and move through rooms whose sensors report over the home's device protocol. The grader has more access than a resident: it reads product state through an internal service interface to check outcomes, and it re-seeds stated preferences when it replays a ticket. Simulated days also run faster than real ones. The product takes the simulated time from sensor reports, so its scheduled behavior follows the simulation's clock, and the few behaviors timed in real seconds can disagree with it (Section 8).

Two design choices matter more than the personas' sophistication.

First, each simulated day runs against a fresh build of the product's own source, in a disposable environment with simulated devices in place of real ones. A digital twin supplies devices, rooms, sensor events and weather, while every decision the assistant makes comes from the product's own code and model, so a recorded failure belongs to the product.

Second, we use the simulator to look for needs the product does not serve. Checking that existing features still work remains the job of the product's own test suite. The simulator files a ticket wherever a need goes unmet, whether a capability is missing or a request is handled badly, and those tickets feed the roadmap. Section 5 lists what this has produced.

3. The verdict rule

Language-model judges have documented biases, including a preference for their own outputs [2, 3], so judgment is structured in layers.

The first layer is deterministic: assertions about world and product state that either hold or do not. Did the room the resident entered light up within the window? Does the thermostat setpoint match the standing preference? These checks are cheap and cannot be talked round, though they can be wrong: each encodes our reading of what the home should do.

Above them sits a judge panel of two language models from two vendors [4] (Figure 1). The judges read one conversation at a time, with a description of the household, and answer one question: did the product leave a genuine need unmet? When two residents make competing requests of the same device, the judges also read both conversations together and judge how the assistant mediated. They read the conversations the simulated resident was unhappy with, up to a fixed number per run, plus a one-in-ten sample of conversations the resident accepted, so that easy acceptance is audited too. A third model then ranks the resulting tickets against a versioned constitution, a short statement of what the home owes its residents:

comfort, convenience, information, security, and the household's social life and routines. We borrow the word from Bai et al. [5]; our constitution orders the work queue and decides nothing about pass or fail, and it gives low priority to a change that placates the simulated residents without serving the need. We assume Goodhart's law holds [6, 7], so any measure the developer can see and change will eventually be optimized in place of the outcome it stands for.

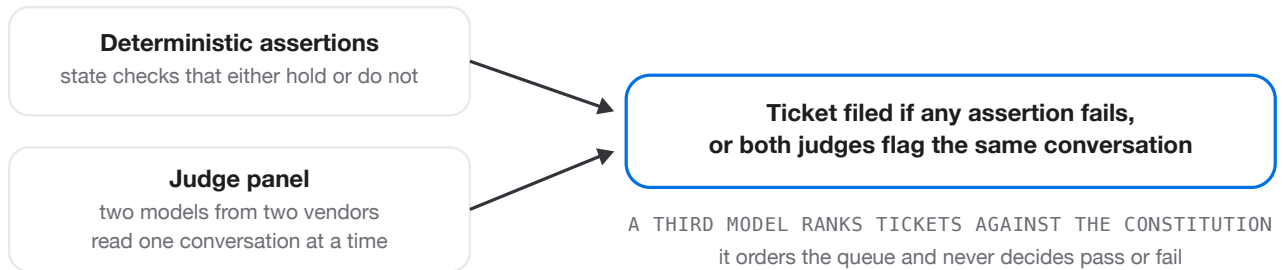


Figure 1. The verdict rule. Deterministic assertions and a two-judge panel feed one rule: a ticket is filed if any assertion fails, or if both judges flag the same conversation. A third model ranks tickets against the constitution and never decides pass or fail.

A conversation or a device outcome produces a ticket only under a conservative rule: a deterministic assertion failed, or both judges flagged the same conversation. A complaint from one judge is recorded and creates no work. The rule trades sensitivity for precision, because each false ticket costs a person time to triage.

We tested how the panel treats a privacy-preserving refusal. For one generated household, both judges graded the same constructed conversation, in which the assistant declined a weather request, once with the home's internet access enabled and once with it disabled. With access enabled, the refusal was graded as a missing capability. With access disabled, as in the Sovereign placement described in our companion report [10], the identical refusal was graded as correct behavior. A later run of the real assistant with internet access disabled produced such a refusal, and that recorded reply is now a regression test for the grader.

4. The boundary

A developer that can change its evaluator will, under enough pressure, raise its score by changing the evaluator [8]. We prevent this with access control and treat instructions to the developer as a second line. The grader lives in a separate repository under separate access control, and a required check fails the product build if product code imports from the grader. The automated developer never holds a write credential. It proposes a patch, and a separate publishing step checks that the patch touches only product code before opening the change. To diagnose a failure it may read a pinned copy of the grader, and it has no way to change the copy that grades it.

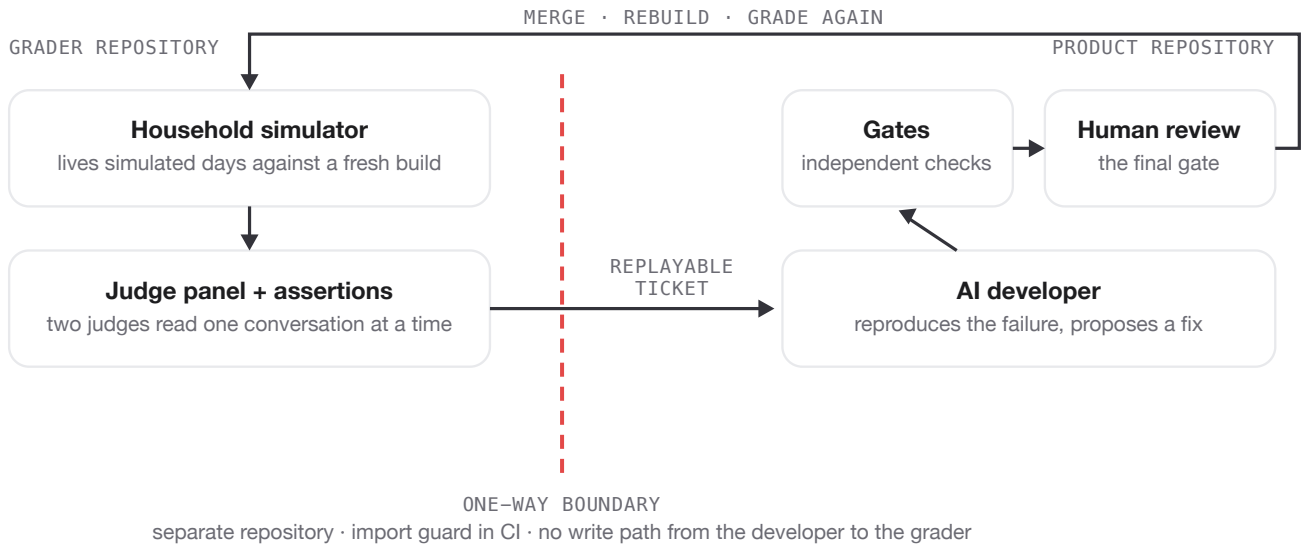


Figure 2. The accountability loop. Tickets cross the boundary in one direction. Fixes are checked by independent gates and merged by a human; the merged source is then rebuilt and graded again by the independently run simulator, which observes the product and gives the developer no write path into the grader.

The loop then runs as follows (Figure 2). The simulator lives its days and files tickets. An AI developer, operating in the product repository, picks up a ticket, reproduces the failure and proposes a fix as an ordinary reviewed change. Before anyone merges it, independent gates check it: the conventional suite, contract checks, a re-run of the failing scenario on the patched build and, for physical failures, the same check in a second household the fix was not written for. The developer's copy of the grader leaves that household out (Section 8). A human reviews and merges. Every night the simulator then rebuilds the merged source and grades it in a newly generated household, and the held-out checks run again. A ticket closed as fixed is reopened if its failure returns. Automatic merging is switched off, and a person merges every change.

The boundary also disciplines the humans. Because the grader's verdicts are produced outside the product's repository, the product team cannot rewrite them. It can decline them: a person can close a ticket as not planned, which retires that check for good, or withdraw a check the team no longer trusts, and the automated developer can dispute a ticket with cited evidence. Each of these acts is recorded where the whole team can see it, and changing what the home is expected to do requires a pull request to the grader's repository.

5. What the loop has produced

Table 1 lists capabilities that began as filed tickets describing a resident need the product could not serve, and ended as shipped capabilities. Each passed a re-check when it shipped: a replay of the original ticket where the failure was physical, and a manual check where no simulated household yet exercised the path. Some of the same checks have since fired again in newly generated households, and we triage each recurrence to find whether the product or the grader is at fault; the standing-preference account below describes product defects found this way, and Section 8 describes faults in the grader. We anticipated

several of these needs ourselves and wrote them into the simulator as checks before they first fired. What the simulator added was measurement: it is scheduled to check them every night, in households we did not design by hand, and it ranks them against everything else it finds.

Table 1. Capabilities that began as simulator tickets, and how each fix was made.		
RESIDENT NEED, AS FILED	SHIPPED CAPABILITY	HOW THE FIX WAS MADE
A stated preference had to be repeated every day	Standing preferences, remembered and enforced across days	Owner-directed AI session, then automated developer
A resident entered a dark, occupied room	Presence-driven lighting, on by default	Owner-directed AI session, then automated developer
Empty rooms stayed lit for hours	Vacancy switch-off, on by default	Automated developer
"Will it rain on the school run?" went unanswered	Live weather and forecasts from an open weather service built on national weather-service models [9]	Automated developer
The home did not know the family's day	Read-only calendar integration over the standard subscription format	Owner-directed AI session
Residents asked for warmth; nothing changed	Thermostat and climate setpoint control	Owner-directed AI session

Two of the six were written end to end by the automated developer and passed its independent gates. The other four were first built in sessions a person directed, with an AI coding assistant, outside those gates; for presence lighting and standing preferences the automated developer later contributed follow-up fixes. A person merged every change.

The standing-preference case shows how the parts work together, and where people were still needed. The requirement came from us: a preference stated once, such as keeping the bedroom at 21 degrees, should hold until the resident changes it. We taught the simulated residents to state such preferences in their own words and added a deterministic check that compares each stated preference with the device timeline on later evenings; a resident who has to ask again counts as a failure. Early runs failed, because the product treated the request as a one-time command. We added durable preferences at the end of June, yet the check kept failing in later simulated households. Over the next six weeks those failures exposed three more defects. The automated developer fixed one in July. We found the other two at the end of July: a room-

name comparison that silently ignored any preference whose capitalization differed from the room name, and a database constraint that made every restated preference fail on the production database while the unit tests, which ran on a different database, passed.

5.1 The loop in numbers

Table 2 summarizes the loop's record to 10 October 2026. It counts discovery runs from the first scheduled run, on 12 June 2026, and every change the automated developer has proposed, the first of them on 10 June.

Table 2. The loop's record to 10 October 2026.	
MEASURE	VALUE
Discovery runs (of which scheduled)	142 (121)
Scheduled nights in the last 30 whose evidence passed the quality gate	4
Open tickets	24
Open tickets about how a request was handled	19
Open tickets for a missing capability	4
Open tickets for reliability	1
Changes proposed by the automated developer	37
Of those, merged	6
Most recent merge of an automated-developer change	13 July 2026

Most open tickets now concern how a request was handled, such as the wording of an answer or an assumption the assistant made without asking. The earlier tickets behind Table 1 were missing capabilities. One of the four open missing-capability tickets is the false ticket described in Section 8, which stays open while that check's verdicts are withheld. Most proposals from the automated developer were closed without merging. Several were repeat attempts at the same gap, including three attempts at the standing temperature preference before a fourth merged; we have not yet classified the remaining closures by cause. The record of the earliest tickets is incomplete because their issues were later deleted.

6. Replayability

Every ticket is a self-contained artifact: the scenario, the household state, the recorded persona behavior and the observed failure. A replay re-drives that record against a fresh product instance. The human side of the interaction is pinned at record time, down to the exact utterances and the exact simulated sensor traffic, and the physical layer is judged by deterministic assertions over the observed device timeline. The product's side is deliberately fresh: the assistant answers anew, and conversational outcomes are re-judged by the same fail-closed panel that graded the original run. A judgment that cannot be grounded is reported as inconclusive. A developer, human or AI, can therefore re-experience the recorded failure itself, and an engineer with access to the bundle can re-run the evidence.

The same structure is what makes the re-grade meaningful. When a fixed scenario goes green, it does so on the recorded day, household and inputs that failed it, so the comparison isolates the change under test from scenario variation. Judgments of free-form conversation keep the variance of the models that produce them, while the physical assertions are deterministic. For failures in the physical home, a fix is credited only when the recorded failure reproduces on the build before the fix and disappears on the build after it. Conversational fixes are checked against the verdict recorded when the ticket was filed, because a fresh conversation on the old build would vary by itself. Some multi-day tickets cannot be replayed faithfully, because the state they depend on from earlier days was not captured; we mark those uncertifiable and keep them out of the automated queue.

7. An organization of accountable AIs

The system is easiest to describe as an organization with four roles. The grader has product sense and writes no code. The developer writes code and cannot grade its own work. The judges come from two competing vendors, and the constitution used to rank their findings is versioned and changed only by people. A human owns every merge. Each role holds one power and lacks the others, and the design depends on keeping them separate.

We find this framing more durable than "AI-assisted development," a phrase that suggests a single assistant that writes its own work and approves it. Here those duties are split across agents whose outputs can be checked against each other, much as an audit separates the people who prepare accounts from those who verify them. Nothing in the arrangement is specific to smart homes. Any product whose success is defined by lived outcomes could pair a simulator of its users, and rules for judging how well it serves them, with an access boundary that keeps its automated developer from editing either.

8. Status and limitations

This section records the state of the loop as of October 2026. The discovery run is scheduled every night. Simulator tickets lie behind every capability in Section 5, the automated developer wrote two of those fixes end to end, and a person made every merge.

Each night's evidence must pass a quality gate before it counts, and many nights do not. In the 30 scheduled nights to 10 October 2026, four passed. None of the other 26 failures was a verdict on the product. On 17 nights, a simulated day included a resident's expectation that the home would remember something overnight; the check that grades such expectations is the one whose verdicts we withdrew (described next), and the gate counted each withheld verdict as missing coverage. On the 8 nights from 29 September to 6 October, the cloud model account the simulation uses was out of credit, a held-out check could not stage its case, and the grader reported that failure under the wrong cause. On one night the simulated stack failed to start. Each of these nights was marked failed. The gate runs after a night's tickets are filed and does not withdraw them, so 9 of the 24 open tickets in Table 2 were filed on 8 of the failed nights. A grader change merged on 10 October 2026 stops counting withheld verdicts as missing coverage and reports a held-out check that could not stage its case under that cause. During the credit outage the simulated days themselves were recorded as complete, because no simulated resident ever spoke to the assistant and so no check could fail; a second grader change, merged later the same day, marks a day lived during a model-provider outage as inconclusive and files nothing from it. Both changes apply from the next scheduled night.

Deterministic checks can be wrong. One compared remembered preferences by overlapping words. It filed a ticket against a home that had behaved correctly, and in testing, a patched version of it would have passed a stored preference that reversed the resident's wish. We withdrew its verdicts until it compares values with their units. Another read a small color rounding difference as a conflict between two residents. The automated developer disputed that ticket with evidence, the dispute was upheld, and the check was narrowed; its verdicts are now withheld until the simulator can stage the conditions the product's shared-device rules depend on.

Simulated time and product time can disagree. The simulator advances in hour-long steps while some product behavior is timed in real seconds, and at least one recurring ticket comes from this kind of disagreement.

The residents are steered. During nightly discovery they receive cues that send them toward product capabilities, so how often a need appears in simulation measures our coverage and says nothing about how often real households would have it.

Coverage is partial. Nightly runs use the cloud-connected configuration; the fully local configuration is exercised in targeted runs. The simulated residents talk to the assistant in text, so speech recognition and synthesis are outside the simulator's reach.

Judging is sampled. The judges read the conversations a resident was unhappy with, plus a one-in-ten control sample, up to a fixed number per run. A conversational fix is credited only when, across at least three repeated samples, the two judges never again agree that it fails; if some samples still fail and others do not, the ticket stays unresolved. Generalization checks and before-and-after checks exist only for failures in the physical home.

The panel has two judges, one per vendor, and both must agree. One of them comes from the same vendor as the assistant's cloud model, and is the same model the assistant falls back to when its first choice fails. Because a conversational ticket needs both judges, that judge can veto any finding about the assistant's replies, and language-model evaluators are known to favor their own generations [3]. A judge from a third vendor would remove that veto and let us require a majority. The model that ranks tickets against the constitution is the assistant's own default model; it orders the queue and cannot pass or fail anything.

Fixes for physical failures are also graded in a second, held-out household, with a different layout and residents, that the fix was not written against. Until 10 October 2026 the developer could read that household's definition in its copy of the grader, and an earlier version of it in the product repository's history, so up to then this check guarded against fixes tied to one home's layout more than against deliberate overfitting. Since that date, the copies of the grader that models in the development loop read contain neither that household's own files nor its details in any shared file, each job checks its copy before a model reads it, the developer's working copy of the product carries no history from before the grader moved to its own repository, and the check's results reach those models only as verdicts. The secrecy holds only as long as no other channel shows a model that household. Conversational failures have no held-out check yet.

Simulated residents are not people. The personas exercise the product across realistic routines, but they inherit the biases and blind spots of the models that animate them. The simulator finds many failures a specification misses, and it will still miss failures a real household would surface, so it does not replace testing with real residents.

Finally, the boundary is administrative. It keeps the product and its automated developer from editing the grader, but the people who own both repositories can still change it. Weakening the grader requires a pull request to its repository, and retiring a single check requires a person to close its ticket.

References

- [1] Park, J. S., O'Brien, J. C., Cai, C. J., Morris, M. R., Liang, P., and Bernstein, M. S. Generative Agents: Interactive Simulacra of Human Behavior. *Proceedings of the 36th Annual ACM Symposium on User Interface Software and Technology (UIST '23)*, 2023. <https://doi.org/10.1145/3586183.3606763>
- [2] Zheng, L., Chiang, W.-L., Sheng, Y., Zhuang, S., Wu, Z., Zhuang, Y., Lin, Z., Li, Z., Li, D., Xing, E. P., Zhang, H., Gonzalez, J. E., and Stoica, I. Judging LLM-as-a-Judge with MT-Bench and Chatbot Arena. *Advances in Neural Information Processing Systems 36 (NeurIPS 2023), Datasets and Benchmarks Track*, 2023. arXiv:2306.05685
- [3] Panickssery, A., Bowman, S. R., and Feng, S. LLM Evaluators Recognize and Favor Their Own Generations. arXiv:2404.13076, 2024. <https://arxiv.org/abs/2404.13076>
- [4] Verga, P., Hofstätter, S., Althammer, S., Su, Y., Piktus, A., Arkhangorodsky, A., Xu, M., White, N., and Lewis, P. Replacing Judges with Juries: Evaluating LLM Generations with a Panel of Diverse Models. arXiv:2404.18796, 2024. <https://arxiv.org/abs/2404.18796>
- [5] Bai, Y., Kadavath, S., Kundu, S., et al. Constitutional AI: Harmlessness from AI Feedback. arXiv:2212.08073, 2022. <https://arxiv.org/abs/2212.08073>
- [6] Goodhart, C. A. E. Problems of Monetary Management: The U.K. Experience. In *Monetary Theory and Practice: The UK Experience*. Macmillan, 1984 (first presented 1975). https://doi.org/10.1007/978-1-349-17295-5_4

- [7] Strathern, M. 'Improving ratings': audit in the British University system. *European Review* 5(3):305–321, 1997. doi:10.1002/(SICI)1234-981X(199707)5:3<305::AID-EURO184>3.0.CO;2-4
 - [8] Amodei, D., Olah, C., Steinhardt, J., Christiano, P., Schulman, J., and Mané, D. Concrete Problems in AI Safety. arXiv:1606.06565, 2016. <https://arxiv.org/abs/1606.06565>
 - [9] Open-Meteo. Free open-source weather API. <https://open-meteo.com>
 - [10] Meyer, P. *Sovereign by Construction*. Elysium Labs technical report ELY-TR-2026-01, 2026. elysium-labs.ai/research/sovereign-by-construction (<https://elysium-labs.ai/research/sovereign-by-construction>)
-

How to cite

Meyer, P. (2026). The Home That Grades Itself. Technical report ELY-TR-2026-02, version 1.4. Elysium Labs, Zürich, Switzerland. <https://elysium-labs.ai/research/the-home-that-grades-itself>

Revision history

Version 1.4 · 11 October 2026

Corrected one statement of version 1.3. In testing, the stored preference that reversed the resident's wish passed a patched version of the withdrawn memory check, a patch that was never merged; the check as shipped failed it.

Version 1.3 · 10 October 2026

Corrected five statements of version 1.2. It said the simulator holds no privileged access, where the grader reads product state through an internal interface; it had the judges score each day against the constitution, where they decide whether a need went unmet and the constitution only ranks tickets; it presented the privacy refusal as a verdict from the corpus, where it was a constructed test; it described fixes graded again on the same scenario after merge, where a fix is re-run before merge and graded nightly afterwards in new households; and it described the held-out scenarios as withheld from the developer, which could read the held-out household, in its copy of the grader and in the product's history, until 10 October 2026. Also stated which fixes the automated developer wrote, corrected the standing-preference account, added the loop's record to date, reported the grader's own errors and the nightly evidence gate, and redrew Figures 1 and 2 to match the text. Added in-text citations and complete references; the page also gained a report number, citation formats, a PDF, table captions, figures renumbered from 3 and 4 to 1 and 2, and references linked to their records.

Version 1.2 · 26 July 2026

Rewrote Section 6 to separate the parts of a replay that are pinned at record time from the parts that run fresh, and stated in Section 5 that a minority of the listed capabilities were verified by manual re-checks. Named the author and labeled the report as self-published and not peer reviewed.

Version 1.1 · 12 July 2026

Reordered the capabilities table, described the simulator as running against the shipped product build, and annotated the loop figure so the re-grade reads as the grader observing the product.

Version 1.0 · 11 July 2026

First published.