# CHOOSING, NOT GENERATING: OPTION SELECTION, LEARNED PRECONDITIONS, AND TEMPORAL CONSISTENCY CHECKS FOR SMART-HOME AGENTS

**Anonymous authors**
Paper under double-blind review

## ABSTRACT

Language-model agents that operate a smart home by issuing one tool call at a time often fail on requests that involve timing, dependencies between devices, or a device or time that the user states incorrectly. Such an agent infers again at every step, from its growing history, what each command requires and when each action should happen. We observe that many of the decisions it makes are selections among options that the tool interface already lists, such as which device a phrase refers to, and that the times follow by computation from values the devices report. We therefore use text generation only to convert the request into a list of atomic goals, in one language-model call. A second model makes these selections: given a question and a list of options, it returns a probability distribution over the options instead of generating text. It maps each goal to a device, or to none when the named device does not exist, and chooses the goal's action and arguments. Code computes all times, checks with a simple temporal network whether the request's timing constraints are consistent, and either executes the request or explains the conflict. Skills learned from demonstrations, annotated with the preconditions of their actions, add the actions that others require first on SimuHome, a smart-home benchmark, and serve as the plans on TimeArena, a benchmark of concurrent time-consuming tasks. On SimuHome, the agent reaches 92.8% and 89.4% success with Gemma 4 12B and Qwen 3.5 9B, 30.5 and 31.6 points above the strongest baseline that uses the same demonstrations; with Gemma 4 12B, it makes about one backbone call per episode instead of about seven for ReAct and uses 13 times fewer backbone tokens. On TimeArena, it completes every episode that can be completed without violating the task instructions.

## 1 INTRODUCTION

Consider a request to a smart-home assistant: "Twenty minutes after the dishwasher finishes, turn on the kitchen light and set the purifier to 60%." To fulfil it, the agent must know which devices the words refer to, what each command requires (the purifier must be on before its fan setting matters), and when the dishwasher will finish (the dishwasher reports a countdown, so the agent can read this time instead of guessing it). If the user adds "at 1:39 PM", the agent must also notice whether that clock time agrees with the countdown.

LLM agents that choose one tool call at a time often fail on such requests. On SimuHome, a smart-home benchmark built on the Matter protocol, the best model in the benchmark's evaluation reaches 54% overall success, and its success on scheduling requests falls well below its success on perception and explicit commands (Seo et al., 2026). On TimeArena, where an agent must carry out several time-consuming tasks at once, agents wait when they could act, rarely run actions in parallel, and often violate dependencies between actions (Zhang et al., 2024). A step-by-step agent such as ReAct (Yao et al., 2022) infers again at every step, from its own history, which procedure to follow, what each action requires, when each action should happen, and whether an action has finished. Agents that reuse experience store it as text (Zhao et al., 2024; Wang et al., 2024) or as programs (Wang et al., 2023), but these records do not state when a procedure applies or how procedures relate in time.



Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

We observe that many decisions such an agent makes are *selections*: which device a phrase refers to, which of a device's commands achieves a goal, which mode a word names. On SimuHome the options can be enumerated from the tool interface, and on TimeArena from programs learned from demonstrations. Times require neither selection nor generation: they follow by computation from the countdowns that the devices report, and so does the answer to whether a stated clock time agrees with them. Only one step requires generating text: turning the free-text request into a structured list of goals, which we call the *goal checklist*.

We therefore give each kind of decision to a different component. A language model writes the goal checklist in one call. Grounding each goal to a device and choosing its action and enumerated arguments are posed as choices from explicit lists of options, and a fast *typed decision model* (TypeSafe AI, 2026) answers them by returning a probability distribution over the options instead of generating text. Code does the arithmetic and time reasoning, checks the request's temporal constraints with a simple temporal network (Dechter et al., 1991), executes the actions, and reads the results back, following fixed rules for action order, power, room conditions and repair. Skills learned from demonstrations describe the options and carry *contracts*: the preconditions an action requires, and the call that succeeded after a call of that form failed. Unlike experience stored as text, the contracts state when an action can be applied. On TimeArena, the learned programs are the plans that the agent schedules.

We make three contributions. (1) An agent built on this division of work that detects requests whose temporal constraints are inconsistent, executes none of the requested actions, and explains the conflict from the values the devices report (Sections 3.2–3.5). (2) A learner that turns demonstrations into skills with contracts. On SimuHome it learns from 50 successful demonstrations and 258 error-free segments of unsuccessful attempts: it selects composite skills by description length, hypothesises preconditions from failures followed by recoveries, and records the alternatives demonstrated after failing calls. On TimeArena it learns per-task programs with durations and dependency contracts, which a partial-order scheduler executes (Sections 3.1 and 3.6). (3) An evaluation on SimuHome with two backbones and on TimeArena with one, in which all methods are scored with the same tools, runner and judges. The evaluation includes a device type that no demonstration reads or operates, three tasks without demonstrations, and ablations that remove one component at a time (Section 4).

On SimuHome, the agent reaches 92.8% overall success with Gemma 4 12B and 89.4% with Qwen 3.5 9B, against 62.3% and 57.8% for the strongest baseline that uses the same demonstrations. With Gemma 4 12B, it makes about one backbone call per episode instead of about seven for ReAct. On TimeArena, it completes every one-task episode with a held-out task and every two- and three-task episode except the cooking combinations in which two tasks share an ingredient name; a defect of the benchmark makes these combinations impossible to complete without violating the task instructions.

## 2 Problem Formulation

An agent serves requests in an environment that it can reach only through a tool interface, whose specification lists the tools, their arguments, and the entities (devices, rooms) that the agent can refer to. A request $q$ is a natural-language instruction that may involve several devices, information the agent must report, and temporal requirements such as deadlines or orderings. The environment changes over time, through the agent's calls and through device operation, and the agent does not see the environment's internal state. We call everything the agent can learn from tool results *public*.

Before evaluation, a learner builds a skill library for the agent from a corpus $\mathcal{D}$ of demonstration trajectories. Each trajectory consists of public timestamps, tool calls and observations, and carries a binary outcome label from the benchmark's automatic check. An episode succeeds when the benchmark judges it correct, from the state of the environment (at the required times, for scheduled actions) or, for reports and for requests that cannot be fulfilled, from the agent's response.

**Assumptions.** The learner and the agent use only public information: they never read hidden state, evaluator rules, or reference solutions, and, of the benchmark's evaluation, the learner uses only the outcome label of each demonstration. The tool interface is given and fixed, and the library is frozen before evaluation. Beyond the interface's documentation and the control rules it publishes (which

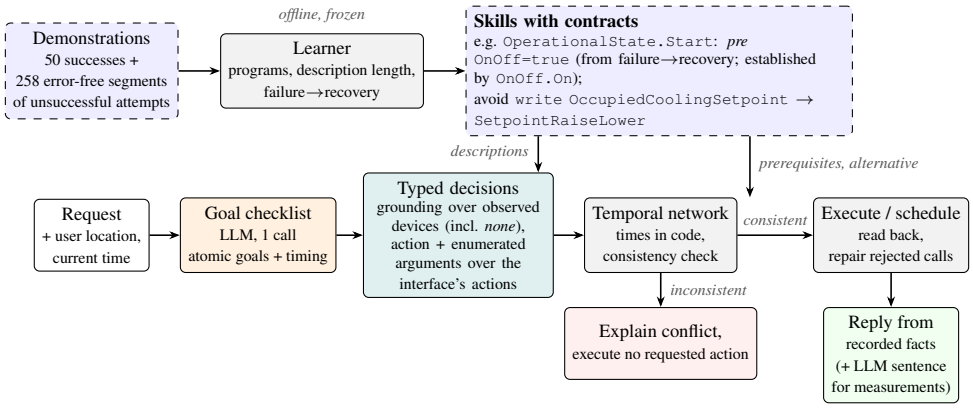


Figure 1: Overview (SimuHome). Offline (top), a learner turns demonstrations into skills with contracts, which supply the descriptions of the options and, at execution, prerequisites and an alternative for a rejected call. One language-model call writes the checklist; a typed decision model grounds each goal (with a "no such device" option) and picks the action and enumerated arguments from the device's interface; code resolves times with a temporal network and executes, or explains the conflict.

device types affect which room measurements), the agent's knowledge of devices comes from the demonstrations and from rules that we wrote on development episodes (Appendix Table 3). During execution, the agent reads outcomes back from the environment.

## 3 METHOD

The agent assigns each kind of decision to a different component (Figure 1). In one call, the language model writes a checklist of the goals in the request (Section 3.2). Grounding and the choice of action and enumerated arguments are typed choices from explicit lists of options, made by the typed decision model (Section 3.3). Code, with the description rules of Appendix Table 3, makes the remaining decisions, computes all times, checks the request's temporal constraints, and executes (Sections 3.4 and 3.5). Offline, a learner turns demonstrations into skills with contracts (Section 3.1); on SimuHome the agent uses them to describe its options, to insert learned prerequisites, and to repair rejected setpoint writes, and on TimeArena they are the plans themselves (Section 3.6). The agent sees the environment only through tool results: it lists the rooms, devices and measurements at the start, and reads a device's structure when a goal concerns it.

### 3.1 SKILL LEARNING WITH CONTRACTS

**Programs.** The corpus $\mathcal{D} = \{\tau_n\}_{n=1}^{N}$ holds trajectories $\tau_n = ((t_i, a_i, o_i))_{i=1}^{T_n}$ of public timestamps, tool calls and observations, each labelled only as a successful demonstration or as an error-free segment of an unsuccessful attempt (on SimuHome, a segment in which every call was accepted). The learner parses every state-changing call, including the steps inside scheduled workflows, with its outcome (HTTP status and error type). Each call is abstracted over the device and its values into an *elementary skill*: a device parameter typed by the capability it needs, namely a cluster with a given command or writable attribute (in Matter, a cluster groups related attributes and commands of a device), and value parameters whose argument keys, types and unit scales come from successful calls (Appendix B.2). Recurring ordered sequences of calls on one device are candidates for *composite skills*.

**Library selection.** We keep a composite only if it recurs in at least two records and lowers the total description length (Rissanen, 1978)

$$\mathcal{J}(\mathcal{G}) = L(\mathcal{G}) + L(A_{\mathcal{D}} \mid \mathcal{G}), \tag{1}$$

where $\mathcal{G}$ is the library and $A_{\mathcal{D}}$ the calls in $\mathcal{D}$: the bits needed to write the library plus the bits needed to write the corpus's calls with it (Appendix B.1). The accepted composites are procedures that recur

across requests, such as powering a device before setting its fan speed, or selecting a wash mode before starting a cycle.

**Contracts.** Each skill receives a contract with capability preconditions (checked against a device's structure), state preconditions, observed effects, and an avoid set. A state precondition is hypothesised from a failure followed by a recovery: when a call fails and then succeeds after another call on the same device, the attributes of that other call's cluster that held at the failure become candidate preconditions; we report how often each holds on executed successes (Appendix B.2). In our corpus this yields one state precondition, that a cycle start (`OperationalState.Start`) requires `OnOff.OnOff = true`, established by `OnOff.On`. Effects come from call responses (effects that no response reports remain unknown). The avoid set lists call forms that failed, or that the scheduler accepted but that conflict with the observed device structure or command schema; for a call form that failed, it also records any call observed to succeed after the failure (e.g. a rejected cooling-setpoint write replaced with `SetpointRaiseLower`). Apart from the outcome label of each demonstration, learning reads no benchmark evaluation data and no simulator code.

### 3.2 Goal Checklist

Given the request $q$, the user's location and the current time, the language model writes an ordered checklist $\mathcal{Q}(q) = (g_1, \ldots, g_m)$ in one schema-constrained call. Each goal is atomic: one measurement to report, one setting or command on one device, or one room condition to change. It records an exact quote, a kind, the room, device, property and action *as written*, a numeric and a non-numeric target, and, for room conditions, a direction. Its timing record takes one of five forms: now, $N$ minutes from now, $N$ minutes after the previous action, at a clock time, or relative to when another appliance finishes (after, before, when it finishes, finishing together, or finishing by a time); it also keeps any clock time the user states. The model copies mentions and numbers; it does not resolve identifiers or add times.

Deterministic rules handle phrasing that the small backbones get wrong (Appendix B.3): number words in times become digits, explicit phrases such as "at 2:57 PM, which is 9 minutes after dishwasher 1 finishes" are parsed into the timing record, and actions in one sentence share its timing. In a request that contains device commands, room-condition goals that name no device are dropped: such goals typically restate the situation or purpose that the commands address ("It is a bit muggy in the study, so turn study room dehumidifier 1 on ..."), and executing them could operate devices that the user did not name. After these rules, the checklist is not revised.

### 3.3 Typed Decisions over the Available Actions

A step-by-step agent spends many of its calls deciding which of a few options to take next, from a prompt that grows with its history. We make these options explicit and hand each choice to a *typed decision model*, a model that returns a distribution over the options of a question instead of generating text. We use Jev (TypeSafe AI, 2026), which answers Choice, Score and yes/no questions over a given state with probabilities. The agent sends at most two question sets per episode, one for grounding and one for action selection. Each set asks independent questions at once over one compact state (the request, the user's location, and each goal's fields), and devices, rooms and actions appear as the options.

**Grounding.** For each goal, the grounding set asks one Choice over the devices of the named room (or of the home), labelled by kind, number and room, with an explicit *none* option ("no listed device matches: the named kind of device with that number does not exist"). Where needed, it also asks Choices over rooms, room measurements, the direction of a room condition, and the device that an anchored timing refers to. The *none* option is how the agent detects requests for devices that do not exist. Code then checks the device number literally: among the devices of the chosen type, it selects the one that carries the mentioned number, or *none* if no such device exists; a naming rule maps a plain "light $N$" to the on/off light.

**Action selection.** For each control goal, code lists the grounded device's interface primitives: its commands and writable attributes, except a fixed list of informational clusters and configuration attributes. Because the options come from the interface rather than from the learned skills, a device

type that no demonstration operates still has actions. Each option is described by the learned skill's description when one exists and otherwise by a one-line description of the command or attribute (in the supplementary material). One Choice picks the action. In the same set, further Choices select every enumerated argument of every candidate action (modes, fan modes, thermostat modes, dryness levels, channels), and code keeps only the answers for the chosen action. For a goal that asks about a device, one Choice picks the attribute that answers it. Numbers are never chosen by the decision model: they come from the checklist and are converted to device units by fixed per-argument rules (hundredths for temperatures, setpoints and lift positions). The distribution over actions orders the alternatives used for repair.

### 3.4 Temporal Network

Code turns the timing records into a simple temporal network (Dechter et al., 1991). Such a network represents each event as a time point and each requirement as a bound on the difference between two time points; the requirements can all hold exactly when the network's distance graph has no negative cycle. Each goal $g_j$ has a time point $x_j$ measured from the request time $T_0 = 0$; each anchoring appliance $k$ has a completion time $e_k$, fixed from its public `CountdownTime`; each constraint bounds a difference, $c^{\min} \le y - x \le c^{\max}$. Constraints that define a goal's timing are hard: $x_j = N$ for "in $N$ minutes"; $x_j - e_k = \pm N$ for "$N$ minutes after/before $k$ finishes"; $x_j = c$ when a clock time is the goal's only timing; $x_j - x_i = N$ for "$N$ minutes after the previous action", where $x_i$ is the previous action on the same device; no action lies in the past; and to finish together with $k$, a cycle of duration $d$ starts at $e_k - d$. The duration $d$ is measured by applying the settings, starting the cycle, reading its `CountdownTime`, and stopping it; these probe calls precede the consistency check and are not undone. The remaining timing statements are *soft*: a clock time stated in addition to another timing, or a stated completion time, must hold within one minute (Appendix B.5). If the network is consistent, the earliest assignment of the hard constraints gives the execution times. If not, the agent executes none of the requested actions, reports for each stated time the time that the other constraints imply ("stated for 13:39, but the other constraints put it at 13:25:54") with the anchor's `CountdownTime`, and asks which was meant.

### 3.5 Execution

Immediate actions are ordered by a fixed rule (power on, then settings, then start or resume, then pause, stop or off). Before an action that requires power according to the learned skills, the agent inserts `OnOff.On` unless the device is already on. The learned state precondition covers cycle starts; for the actions that follow `OnOff.On` in a learned composite (fan speed and mode, thermostat mode, level, wash mode), we assume the same prerequisite, a heuristic reading of the composites. Three media commands, which the demonstrations do not cover, are given the same prerequisite. Timed actions are compiled into scheduled workflows, one per device and time, so that a failing step cannot block another device's steps. After an immediate write or power, operational-state, mode, level or position command, the agent reads the target attribute back and records the reading in its reply; a mismatch is reported, not repaired. When an immediate call fails, the agent powers the device on and retries. If the call still fails, it replaces a rejected thermostat setpoint write with `SetpointRaiseLower`, the demonstrated alternative in the avoid set; if that does not apply or also fails, it tries the next two actions in the decision model's distribution. For a room condition that names no device ("the office is too warm"), an effect model that mirrors the interface's public control rules decides which devices can help and what to set, or why none can (Appendix B.6). The reply is rendered by templates from the recorded facts (reported values, what was done or scheduled and when, what cannot be done and why, and the public readings used); when the reply consists only of room measurements, the language model adds one or two interpretive sentences.

### 3.6 Instantiation for TimeArena

TimeArena exposes templated actions ("cook OBJ1 in OBJ2") over listed objects, one minute per step. From the demonstrations the learner extracts, per task, the program of grounded actions and, per action, its duration, whether it occupies the agent or runs in the background, the objects it occupies, and dependency contracts, including those that the environment's refusal messages reveal ("Because X is Y"). For a task without demonstrations, the language model proposes the program

once, given two learned programs of the same scenario as examples, and code completes and orders it with contracts learned for each verb rather than for each action. The agent merges the tasks' programs into one partial order and executes it. At each decision, among the actions whose dependencies are met and whose objects are free, it simulates the rest of the plan after each candidate with a base rule (background actions and long chains first) and picks the candidate whose simulated completion is earliest within the 40-decision budget. A refusal adds the missing dependency, and the decision model resolves the choices that code leaves open (Appendix E).

# 4 EXPERIMENTS

We ask four questions. **(Q1)** Does the agent outperform agents that act step by step and agents that reuse the same demonstrations without contracts, including on a device type that no demonstration reads or operates and on tasks without demonstrations? **(Q2)** Which components matter: on SimuHome, the learned contracts, the typed decision model, the explicit "no such device" option, the temporal consistency check, and read-back with repair; on TimeArena, the learned programs, concurrent execution and the rollout? **(Q3)** What does this design cost and save, in language-model calls, tokens and time? **(Q4)** Where does the agent still fail?

## 4.1 SETUP

**Benchmarks.** SimuHome (Seo et al., 2026) simulates a Matter-based smart home whose room measurements change as devices operate, in virtual time that runs with the wall clock. Its 600 episodes cover perception (QT1), implicit intent (QT2), explicit intent (QT3), and future, dependency and concurrent scheduling (QT4-1, QT4-2, QT4-3), each feasible or infeasible; episodes are judged from the simulator's state or by a panel of LLM judges (Table 1). TimeArena (Zhang et al., 2024) has 30 cooking, household and laboratory tasks whose actions take time, depend on each other, and occupy the agent or objects; episodes combine one to three tasks, with at most 40 decisions.

**Demonstrations and splits.** We use the same GPT-5.1 demonstrations as the demonstration-based baselines (Appendix Table 4): on SimuHome, 50 successful demonstrations and 258 error-free segments of 205 unsuccessful attempts, on episodes generated with seeds disjoint from the 600 evaluation episodes, with every request or call that concerns a dehumidifier removed; we evaluate on all 600 episodes and report separately the 58 episodes whose goals involve a dehumidifier (*held-out*). On TimeArena, one task per scenario is held out, demonstrations are single-task episodes of the other 27, and we evaluate on the 90 test configurations used for the baselines (10 single-task, 10 two-task and 10 three-task per scenario). The agent was developed only on episodes and configurations outside the evaluation sets, which were run once per backbone and seed after the agent's logic was frozen (Appendix C).

**Approaches.** All agents use the same backbone, the same pinned SimuHome tools, runner and LLM judges, and the same TimeArena environment loop; the baselines were run separately with the same harness (Appendix D). Budgets differ: our agent may make 400 tool calls per episode, the step-by-step baselines 40 steps. *ReAct* (Yao et al., 2022) makes one tool call at a time; *Reflexion* (Shinn et al., 2023) adds self-reflection over up to three trials; *Self-plan* is TimeArena's planning prompt, which we report only on TimeArena, since on SimuHome it scored below ReAct (42.9% and 39.2%); *ExpeL* (Zhao et al., 2024) and *Agent Workflow Memory* (Wang et al., 2024) (*Workflow Memory* in the tables) add insights or workflows extracted from the same demonstrations we use. *Ours* uses the backbone for the checklist and `jev-1.13.0` (TypeSafe AI, 2026) for typed decisions; it is thus the only method that also calls a second model, accessed through an API, and the only one with description rules (Appendix Table 3); the ablation *LLM decisions* replaces that model with the backbone.

**Metrics.** On SimuHome, success is the fraction of episodes the benchmark judges successful, per query type, separately for feasible and infeasible episodes. On TimeArena we report the benchmark's average progress score (AS), completion speed (CS, progress per minute), completion rate (CR), and average completion time of completed episodes (CT, minutes); the *Oracle* is a schedule computed offline from the tasks' dependency graphs, not a simulator run. *Precision* is the fraction of

Table 1: Main results. (a) SimuHome success (%) by query type (QT) on feasible (F) and infeasible (IF) episodes, in the benchmark's format; superscripts mark evaluation by the simulator's state ($S$) or by a panel of LLM judges ($J$). *Held-out*: the 58 episodes involving the dehumidifier (Section 4.1). Means over three seeds; the standard deviation of Ours is at most 4.1 points per cell and 0.3 overall (all cells in Appendix F). (b) TimeArena, averaged over the three scenarios (metrics in Section 4.1); as in the baselines' reports, a scenario without completed episodes enters the CT average as 0, so the agents' three-task CT covers household and laboratory only and is not comparable with the Oracle's. One-task episodes use the held-out tasks. Bold: best among agents; for CT, among the agents with the highest CR.

**(a) SimuHome**

| | QT1 | | QT2 | | QT3 | | QT4-1 | | QT4-2 | | QT4-3 | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Method | $F^J$ | $IF^J$ | $F^S$ | $IF^J$ | $F^S$ | $IF^J$ | $F^S$ | $IF^J$ | $F^S$ | $IF^J$ | $F^S$ | $IF^J$ | Held-out | Overall |
| *Gemma 4 12B* | | | | | | | | | | | | | | |
| ReAct | 87.3 | 62.7 | 46.0 | 34.7 | 84.0 | 68.0 | 48.7 | 20.7 | 62.0 | 0.0 | 17.3 | 6.7 | 54.6 | 44.8 |
| Reflexion | 88.0 | 80.0 | 62.0 | 50.7 | 91.3 | 74.7 | 50.0 | 25.3 | 62.7 | 0.0 | 36.7 | 11.3 | 62.1 | 52.7 |
| ExpeL | 88.7 | 73.3 | 68.7 | 42.7 | 87.3 | 70.7 | 46.0 | 58.0 | 70.7 | 54.7 | 64.0 | 22.7 | 70.1 | 62.3 |
| Workflow Memory | 85.3 | 62.7 | 39.3 | 33.3 | 86.7 | 67.3 | 44.7 | 34.0 | 63.3 | 0.0 | 20.7 | 11.3 | 56.3 | 45.7 |
| Ours | **92.7** | **96.7** | **90.7** | **78.0** | **98.0** | **100.0** | **97.3** | **100.0** | **92.0** | **95.3** | **89.3** | **83.3** | **98.3** | **92.8** |
| *Qwen 3.5 9B* | | | | | | | | | | | | | | |
| ReAct | 87.3 | 82.0 | 38.0 | 41.3 | 83.3 | 82.0 | 28.7 | 15.3 | 46.0 | 0.0 | 9.3 | 14.0 | 46.6 | 43.9 |
| Reflexion | **92.0** | 88.0 | 68.7 | 60.0 | 94.0 | 92.0 | 44.7 | 17.3 | 61.3 | 1.3 | 22.0 | 32.0 | 57.5 | 56.1 |
| ExpeL | 87.3 | 87.3 | 56.7 | 38.7 | 84.7 | 85.3 | 32.7 | 48.7 | 55.3 | 30.7 | 51.3 | 35.3 | 56.9 | 57.8 |
| Workflow Memory | 85.3 | 84.7 | 40.0 | 46.0 | 84.7 | 78.0 | 24.7 | 28.7 | 31.3 | 8.0 | 13.3 | 27.3 | 47.1 | 46.0 |
| Ours | 76.0 | **96.7** | **91.3** | **79.3** | **95.3** | **100.0** | **90.7** | **100.0** | **90.7** | **95.3** | **85.3** | **72.7** | **96.6** | **89.4** |

**(b) TimeArena, Gemma 4 12B**

| | One task (held-out) | | | | Two tasks | | | | Three tasks | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Method | AS↑ | CS↑ | CR↑ | CT↓ | AS↑ | CS↑ | CR↑ | CT↓ | AS↑ | CS↑ | CR↑ | CT↓ |
| ReAct | **100.00** | 5.55 | **100.00** | 18.22 | 94.10 | 2.93 | 71.11 | 33.66 | 87.76 | 2.00 | 33.33 | 29.68 |
| Reflexion | **100.00** | 5.57 | **100.00** | 18.11 | 97.03 | 2.97 | 84.44 | 34.37 | 92.43 | 2.14 | 61.11 | 28.68 |
| Self-plan | 69.78 | 4.89 | 33.33 | 5.67 | 70.61 | 2.88 | 36.67 | 30.75 | 59.81 | 1.87 | 20.00 | 28.56 |
| ExpeL | 94.67 | 5.24 | 88.89 | 17.00 | 88.73 | 3.10 | 73.33 | 31.99 | 80.77 | 2.09 | 46.67 | 28.27 |
| Workflow Memory | 98.78 | 5.26 | 88.89 | 17.56 | 91.77 | 2.91 | 71.11 | 34.21 | 87.46 | 1.96 | 44.44 | 29.96 |
| Ours | **100.00** | **6.01** | **100.00** | **17.00** | **97.97** | **3.62** | **90.00** | **28.19** | **96.37** | **2.45** | **66.67** | **24.16** |
| Oracle | 100.00 | 6.72 | 100.00 | 15.00 | 100.00 | 3.73 | 100.00 | 27.67 | 100.00 | 2.54 | 100.00 | 40.87 |

episodes that the benchmark judges successful among those whose reply reports no failure, refusal or missing reading.

**Implementation.** The backbones are Gemma 4 12B and Qwen 3.5 9B, served with vLLM and sampled at temperature 0.7; GPT-5.1 only collected demonstrations. We report means and standard deviations over three seeds (42, 43, 44); seeds change the backbone's sampling, and identical decision-model inputs receive cached answers. Appendix D lists all settings.

### 4.2 Results

**(Q1) Main comparison.** On SimuHome (Table 1a), our agent reaches 92.8% overall success with Gemma 4 12B and 89.4% with Qwen 3.5 9B, 30.5 and 31.6 points above the strongest baseline that uses the same demonstrations (ExpeL) and 48.0 and 45.5 points above ReAct. Most of the gain comes from the scheduling query types (QT4), where the baselines fail most: on dependency scheduling with an inconsistent request (QT4-2 IF), where the agent must read the anchor's countdown and notice that a stated clock time contradicts it, success rises from at most 54.7% (ExpeL) to 95.3% with both backbones. Every infeasible query type improves: requests for missing devices are caught by grounding (the *none* option and a literal number check), inconsistent timing by the temporal network, and impossible room-condition changes by the effect model, rather than by the

Table 2: Ablations. (a) SimuHome, Gemma 4 12B, seed 42 (600 episodes): success on feasible (F), infeasible (IF) and infeasible scheduling (QT4 IF) episodes and overall (%), precision (%), and agent time per episode (s); the ablations ran together on shared endpoints, so compare their times with each other. (b) TimeArena, Gemma 4 12B, three seeds: CR (%) and CT (minutes).

**(a) SimuHome**

| Variant | F | IF | QT4 IF | Overall | Precision | Time (s) |
|---|---|---|---|---|---|---|
| Ours | 92.7 | 92.0 | 92.0 | 92.3 | 90.4 | 12.0 |
| LLM decisions (backbone answers the questions) | 91.7 | 92.0 | 92.0 | 91.8 | 89.3 | 21.5 |
| No learned contracts | 89.0 | 92.0 | 92.0 | 90.5 | 87.6 | 14.6 |
| No *none* option in grounding | 93.0 | 64.7 | 91.3 | 78.8 | 77.2 | 14.9 |
| No temporal consistency check | 93.3 | 48.7 | 5.3 | 71.0 | 63.7 | 15.1 |
| No read-back or repair | 91.3 | 92.0 | 92.0 | 91.7 | 90.0 | 15.1 |

**(b) TimeArena**

| Variant | Two tasks | | Three tasks | |
|---|---|---|---|---|
| | CR↑ | CT↓ | CR↑ | CT↓ |
| Ours | 90.0 | 28.2 | 66.7 | 24.2 |
| All tasks unseen (no learned per-task programs) | 75.6 | 30.2 | 54.4 | 26.8 |
| Sequential (no concurrency) | 83.3 | 40.5 | 56.7 | 34.0 |
| No rollout (base rule only) | 90.0 | 29.4 | 66.7 | 25.0 |
| No template completion for unseen tasks | 90.0 | 28.2 | 66.7 | 24.2 |
| No probing of uncertain dependencies | 90.0 | 28.2 | 66.7 | 24.1 |
| No typed decisions | 90.0 | 28.2 | 66.7 | 24.2 |

language model's judgement. On the held-out episodes the agent scores 98.3% and 96.6%. Its actions on the dehumidifier come from the device's interface and, for room conditions, from the effect model. The held-out episodes are also easier for the Gemma baselines (7.8 to 10.6 points above their overall rates), so this result shows that the agent does not degrade on a device type without demonstrations, rather than that it transfers. The one cell in which the baselines are better is QT1 feasible with Qwen (76.0% against 85.3–92.0%), where Qwen's checklist turns "brand" into "product ID" and its appended interpretive sentence can contradict the reported value (Appendix F.1). The agent reports every goal achieved in about half of the episodes; its precision is 91.2% (Gemma) and 86.2% (Qwen). It refuses 97.2% of the infeasible episodes and 1.7% (Gemma) and 2.7% (Qwen) of the feasible ones.

On TimeArena, the agent completes every one-task episode with a held-out task (three tasks, nine episodes; ReAct and Reflexion also complete all of them), 90.0% of two-task and 66.7% of three-task episodes. The rest are exactly the cooking combinations in which two tasks share an ingredient name, which no agent completes without adding an ingredient to another task's dish, against the instructions (Appendix C). By completion speed it is the fastest agent at every task count. On the two-task episodes that it completes, its completion time averaged over scenarios is 28.2 minutes, against 27.6 for the Oracle's schedules of the same episodes (the Oracle's 27.7 in Table 1 also covers the three cooking pairs that no agent completes); on laboratory three-task episodes the two completion times are equal (Appendix Tables 8 and 9).

**(Q2) Components.** The two components that turn infeasible requests into correct refusals matter most (Table 2a). Without the temporal consistency check the agent schedules whatever the request says, and success on infeasible scheduling requests falls from 92.0% to 5.3% (71.0% overall). Without the *none* option the decision model must pick an existing device, and infeasible success falls from 92.0% to 64.7% (78.8% overall), mostly on requests for missing devices; both also lower precision. The learned contracts add 1.8 points (90.5% without them; net 11 of 600 episodes), mostly in concurrent scheduling, where a cycle start needs the learned power precondition, and in room conditions, where a rejected setpoint write needs the demonstrated alternative. Repair adds 0.7 points (net 4 episodes); read-back alone cannot change the outcome, since a mismatch is only reported. Replacing the decision model with the backbone answering the same typed questions changes success little (91.8% against 92.3%, a net difference of 3 of 600 episodes with one seed) but makes the agent 1.4 to 1.5 times slower than the other ablations (21.5 s against 14.6–15.1 s) and adds 1.6

language-model calls per episode; with this backbone, the decision model mainly makes the agent faster. On TimeArena, withholding the learned per-task programs, so that every task is planned from its description, lowers three-task completion from 66.7% to 54.4%; executing actions in sequence lowers three-task completion to 56.7%, because waiting for background actions exhausts the 40-decision budget, and raises two-task completion time from 28.2 to 40.5 minutes; the rollout shortens completion time by about a minute; and the other three ablations in Table 2(b) change no completion rate and change completion time by at most 0.11 minutes.

**(Q3) Cost.** With Gemma 4 12B, our agent makes 1.06 language-model calls per episode (ReAct 6.85, ExpeL 6.44) and uses 3.35K backbone tokens (ReAct 44.7K, ExpeL 80.3K), 13 and 24 times fewer tokens; invalid tool calls fall from 0.23 (ReAct) to 0.01 per episode (Appendix Table 7). It sends 1.63 decision-model requests per episode, 0.65 of which reached the API (the rest were cache hits, mostly seed-42 answers for identical checklists), at 0.22 s and 1.9K input tokens each: \$0.094 for all 1,800 episodes (\$0.24 without the cache). Its agent time per episode is 10.3 s, against 23.0 s for ReAct and 66.2 s for ExpeL, whose times come from separate runs (Appendix F.3).

**(Q4) Remaining failures.** Of the 46 failures of seed 42 with Gemma, 14 start at the checklist, 13 are correct refusals whose template reason the judges found insufficient, and 7 have a benchmark label that contradicts the request; grounding by the decision model causes none (one of 66 with Qwen; Appendix F.1).

## 5 Related Work

Smart-home agents let an LLM choose devices and actions (King et al., 2024; Rivkin et al., 2023), and benchmarks report that such agents fail on invalid, ambiguous, multi-device and time-dependent requests (Li et al., 2025; 2026; Seo et al., 2026; Zhang et al., 2024; Gonzalez-Pumariega et al., 2025); in these agents the language model infers dependencies, applicability and times itself. Agents that reuse experience store insights, workflows or programs (Zhao et al., 2024; Wang et al., 2024; 2023; 2025; Zheng et al., 2025), but these records do not state when a procedure applies. We select skills by compression, as in library learning (Ellis et al., 2021; Bowers et al., 2023; Grand et al., 2024), and attach preconditions that, unlike action-model learners such as SAM (Juba et al., 2021), we hypothesise only from failures followed by recoveries. Instead of generating every decision as text, possibly constrained to a schema (Willard & Louf, 2023), we assign the selections to a typed decision model (TypeSafe AI, 2026) and check the request's timing constraints with a simple temporal network (Dechter et al., 1991). Appendix A discusses each line of work in detail.

## 6 Conclusion

We presented an agent that assigns each kind of decision to a different component: a language model writes the goal checklist once, a typed decision model grounds each goal and selects its action and enumerated arguments from explicit lists of options, code computes and checks times and executes, and skills learned from demonstrations supply prerequisites and a repair on SimuHome and the plans on TimeArena. On SimuHome this raises success from 62.3% and 57.8% for the strongest demonstration-based baseline to 92.8% and 89.4% with about one language-model call per episode, and on TimeArena the agent completes every episode that can be completed without violating the task instructions; most remaining failures come from the checklist or from the wording of refusals.

**Limitations.** Both benchmarks are simulated and the demonstrations come from one model. Some of the agent's SimuHome interface knowledge is description (Appendix Table 3), and the ablations do not remove it. The checklist is the least reliable step, and its normalization rules were written for these benchmarks' phrasing. The agent also relies on a second model that the baselines do not use; we report its cost and an ablation that replaces it with the backbone. The SimuHome ablations use one seed, TimeArena was run with one backbone, and the held-out SimuHome subset is easier for every method with Gemma.

### AI USE STATEMENT

We used generative AI tools to assist in the implementation of the methods and to assist with translation, and not for any other task whose disclosure the ICLR policy requires. We also used them to create Figure 1 and to identify related literature. The authors reviewed all AI-assisted work and take responsibility for the final content of this work.

### REPRODUCIBILITY STATEMENT

Section 4.1 and Appendices C–E describe the demonstrations, splits, development protocol, budgets, and baselines; Appendix H gives the prompts and Appendix G shows learned skills. The supplementary material contains the code of the agent, the skill learners, the learned libraries, the scripts that select the development episodes and configurations, and our evaluation harnesses, together with the per-episode records of all runs of our agent, including the ablations. The typed decision model is accessed through a public API with a pinned version (`jev-1.13.0`); all its answers used in our runs are included in the cache in the supplementary material.

## REFERENCES

Matthew Bowers, Theo X Olausson, Lionel Wong, Gabriel Grand, Joshua B Tenenbaum, Kevin Ellis, and Armando Solar-Lezama. Top-down synthesis for library learning. *Proceedings of the ACM on Programming Languages*, 7(POPL):1182–1213, 2023.

Rina Dechter, Itay Meiri, and Judea Pearl. Temporal constraint networks. *Artificial intelligence*, 49 (1-3):61–95, 1991.

Kevin Ellis, Catherine Wong, Maxwell Nye, Mathias Sablé-Meyer, Lucas Morales, Luke Hewitt, Luc Cary, Armando Solar-Lezama, and Joshua B Tenenbaum. Dreamcoder: Bootstrapping inductive program synthesis with wake-sleep library learning. In *Proceedings of the 42nd acm sigplan international conference on programming language design and implementation*, pp. 835–850, 2021.

Gonzalo Gonzalez-Pumariega, Leong Yean, Neha Sunkara, and Sanjiban Choudhury. Robotouille: An asynchronous planning benchmark for llm agents. In *International Conference on Learning Representations*, volume 2025, pp. 83757–83792, 2025.

Gabriel Grand, Lio Wong, Maddy Bowers, Theo X Olausson, Muxin Liu, Joshua B Tenenbaum, and Jacob Andreas. Lilo: Learning interpretable libraries by compressing and documenting code. In *International Conference on Learning Representations*, volume 2024, pp. 30399–30446, 2024.

Lin Guan, Karthik Valmeekam, Sarath Sreedharan, and Subbarao Kambhampati. Leveraging pretrained large language models to construct and utilize world models for model-based task planning. *Advances in Neural Information Processing Systems*, 36:79081–79094, 2023.

Brendan Juba, Hai S Le, and Roni Stern. Safe learning of lifted action models. *arXiv preprint arXiv:2107.04169*, 2021.

Daniel Kahneman, Patrick Egan, et al. *Thinking, fast and slow*, volume 1. Farrar, straus and Giroux New York, 2011.

Sehoon Kim, Suhong Moon, Ryan Tabrizi, Nicholas Lee, Michael W Mahoney, Kurt Keutzer, and Amir Gholami. An llm compiler for parallel function calling. *arXiv preprint arXiv:2312.04511*, 2023.

Evan King, Haoxiang Yu, Sangsu Lee, and Christine Julien. Sasha: creative goal-oriented reasoning in smart homes with large language models. *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies*, 8(1):1–38, 2024.

Jia Li and Guoxiang Zhao. T3 planner: A self-correcting llm framework for robotic motion planning with temporal logic. *arXiv preprint arXiv:2510.16767*, 2025.

Kuan Li, Shuo Zhang, Huacan Wang, Fangzhou Yu, Zecheng Sheng, Yi Gu, Weipeng Ming, Lei Xue, Chen Liu, Sen Hu, et al. Smh-bench: Benchmarking llm agents for environment-grounded reasoning and action in smart homes. *arXiv preprint arXiv:2606.01912*, 2026.

Silin Li, Yuhang Guo, Jiashu Yao, Zeming Liu, and Haifeng Wang. Homebench: Evaluating llms in smart homes with valid and invalid instructions across single and multiple devices. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 12230–12250, 2025.

Yichao Liang, Nishanth Kumar, Hao Tang, Adrian Weller, Joshua B Tenenbaum, Tom Silver, João F Henriques, and Kevin Ellis. Visualpredicator: Learning abstract world models with neuro-symbolic predicates for robot planning. In *International Conference on Learning Representations*, volume 2025, pp. 60416–60444, 2025.

Yichao Liang, Dat Nguyen, Cambridge Yang, Tianyang Li, Joshua B Tenenbaum, Carl Edward Rasmussen, Adrian Weller, Zenna Tavares, Tom Silver, and Kevin Ellis. Exopredicator: Learning abstract models of dynamic worlds for robot planning. In *International Conference on Learning Representations*, volume 2026, pp. 49174–49214, 2026.

Bo Liu, Yuqian Jiang, Xiaohan Zhang, Qiang Liu, Shiqi Zhang, Joydeep Biswas, and Peter Stone. Llm+ p: Empowering large language models with optimal planning proficiency. *arXiv preprint arXiv:2304.11477*, 2023.

Jorma Rissanen. Modeling by shortest data description. *Automatica*, 14(5):465–471, 1978.

Dmitriy Rivkin, Francois Hogan, Amal Feriani, Abhisek Konar, Adam Sigal, Steve Liu, and Greg Dudek. Sage: Smart home agent with grounded execution. *arXiv preprint arXiv:2311.00772*, 2023.

Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. Toolformer: Language models can teach themselves to use tools. *Advances in neural information processing systems*, 36:68539–68551, 2023.

Gyuhyeon Seo, Jungwoo Yang, Junseong Pyo, Nalim Kim, Jonggeun Lee, and Yohan Jo. Simuhome: A temporal-and environment-aware benchmark for smart home llm agents. In *International Conference on Learning Representations*, volume 2026, pp. 40070–40097, 2026.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. *Advances in neural information processing systems*, 36:8634–8652, 2023.

Tom Silver, Rohan Chitnis, Nishanth Kumar, Willie McClinton, Tomás Lozano-Pérez, Leslie Kaelbling, and Joshua B Tenenbaum. Predicate invention for bilevel planning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 37, pp. 12120–12129, 2023.

TypeSafe AI. Jev: A System One model for typed decisions. `https://docs.typesafe.ai`, 2026. Model `jev-1.13.0`; accessed September 2026.

Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. Voyager: An open-ended embodied agent with large language models. *arXiv preprint arXiv:2305.16291*, 2023.

Zora Zhiruo Wang, Jiayuan Mao, Daniel Fried, and Graham Neubig. Agent workflow memory. *arXiv preprint arXiv:2409.07429*, 2024.

Zora Zhiruo Wang, Apurva Gandhi, Graham Neubig, and Daniel Fried. Inducing programmatic skills for agentic tasks. *arXiv preprint arXiv:2504.06821*, 2025.

Brandon T Willard and Rémi Louf. Efficient guided generation for large language models. *arXiv preprint arXiv:2307.09702*, 2023.

Qiang Yang, Kangheng Wu, and Yunfei Jiang. Learning action models from plan examples using weighted max-sat. *Artificial Intelligence*, 171(2-3):107–143, 2007.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models. *arXiv preprint arXiv:2210.03629*, 2022.

Yikai Zhang, Siyu Yuan, Caiyu Hu, Kyle Richardson, Yanghua Xiao, and Jiangjie Chen. Timearena: Shaping efficient multitasking language agents in a time-aware simulation. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 3894–3916, 2024.

Andrew Zhao, Daniel Huang, Quentin Xu, Matthieu Lin, Yong-Jin Liu, and Gao Huang. Expel: Llm agents are experiential learners. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pp. 19632–19642, 2024.

Boyuan Zheng, Michael Y Fatemi, Xiaolong Jin, Zora Zhiruo Wang, Apurva Gandhi, Yueqi Song, Yu Gu, Jayanth Srinivasa, Gaowen Liu, Graham Neubig, et al. Skillweaver: Web agents can self-improve by discovering and honing skills. *arXiv preprint arXiv:2504.07079*, 2025.

Siyu Zhou, Tianyi Zhou, Yijun Yang, Guodong Long, Deheng Ye, Jing Jiang, and Chengqi Zhang. Wall-e: World alignment by neurosymbolic learning improves world model-based llm agents. *Advances in Neural Information Processing Systems*, 38:65352–65391, 2026.

Wang Bill Zhu, Ishika Singh, Robin Jia, and Jesse Thomason. Language models can infer action semantics for symbolic planners from environment feedback. In *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pp. 8751–8773, 2025.

## APPENDIX CONTENTS


- A Extended related work
- B Method details: B.1 library selection, B.2 learned contracts, B.3 checklist and normalization, B.4 typed decisions, B.5 temporal network, B.6 room conditions
- C Benchmarks, demonstrations and development protocol
- D Implementation and baselines
- E TimeArena details
- F Additional results
- G Learned skill library (examples)
- H Prompts


## A EXTENDED RELATED WORK

**LLM agents for smart homes and time-dependent tasks.** Smart-home agents such as Sasha (King et al., 2024) and SAGE (Rivkin et al., 2023) let an LLM choose devices and actions. Benchmarks report where such agents fail: on invalid and multi-device instructions in HomeBench (Li et al., 2025), on scheduling and ambiguous requests in SMH-Bench (Li et al., 2026), and on time-dependent requests in SimuHome (Seo et al., 2026). TimeArena (Zhang et al., 2024) and Robotouille (Gonzalez-Pumariega et al., 2025) report similar failures when actions take time and overlap. LLM-Compiler (Kim et al., 2023) runs independent calls in parallel from a dependency graph that the LLM writes. In these agents, the language model infers the dependencies between actions, whether an action applies, whether it has finished, and when it should happen. In our agent, code computes times, the learned contracts supply action dependencies on TimeArena and power prerequisites on SimuHome, and the agent reads the state back after immediate actions.

**Learning reusable skills from experience.** ExpeL (Zhao et al., 2024) and Agent Workflow Memory (Wang et al., 2024) reuse insights and workflows from past trajectories as text, and Voyager (Wang et al., 2023), Agent Skill Induction (Wang et al., 2025), and SkillWeaver (Zheng et al., 2025) store skills as programs. Library learning selects abstractions by compression (Ellis et al., 2021; Bowers et al., 2023; Grand et al., 2024). We apply the compression criterion to tool-call trajectories and attach contracts to the learned skills: preconditions hypothesised from failures followed by recoveries, observed effects, and demonstrated alternatives to failing calls. On SimuHome, however, the agent chooses among the actions of the tool interface, not among the learned skills; the library describes these actions, adds power prerequisites, and supplies the alternative for rejected setpoint writes.

**Learning action models for planning.** Action-model learning infers the preconditions and effects of given actions from trajectories (Yang et al., 2007; Juba et al., 2021). SAM (Juba et al., 2021) keeps as preconditions the facts that held before every observed occurrence of an action, whereas we hypothesise a precondition only from a failure followed by a recovery. LLMs build or refine such models from documentation and feedback (Liu et al., 2023; Guan et al., 2023; Zhu et al., 2025; Zhou et al., 2026), and predicate invention also learns the state abstraction (Silver et al., 2023; Liang et al., 2025; 2026). Our learner reads no documentation on SimuHome, only the tool calls and observations of the demonstrations, which include the device structures and control rules that the tools returned; on TimeArena, it also takes pre-state words from the action descriptions in the task instructions. In both benchmarks the contracts are stated over the states that the environment reports (device attributes on SimuHome, object-state words on TimeArena), and we invent no new predicates. T3 Planner (Li & Zhao, 2025) verifies LLM-proposed plans with temporal logic; we check the constraints stated in the request with a simple temporal network (Dechter et al., 1991).

**Fast, typed decisions.** Agents usually make every decision by generating text, possibly constrained to a schema (Willard & Louf, 2023) or formatted as a tool call (Schick et al., 2023). Following the distinction between fast judgments and deliberate reasoning (Kahneman et al., 2011), we assign the selections of a tool-using agent to a typed decision model that returns distributions

over enumerated options (TypeSafe AI, 2026), and keep the generative model for the one step that converts the free-text request into a list of goals.

# B METHOD DETAILS

## B.1 LIBRARY SELECTION

The corpus for Eq. 1 consists of 299 blocks (the successful state-changing calls of a trace on one device, or the steps of one accepted workflow; 429 calls on 36 devices). An elementary skill costs $\log_2(\#\text{tools}) + 8(|\text{cluster}| + 1) + 8(|\text{name}| + 1)$ bits (cluster, command and attribute names are spelled out at 8 bits per character, plus a terminator); a composite of $k$ steps costs an Elias-gamma code for $k$ plus $k \log_2(\#\text{elementary skills})$ bits, with steps sharing the device parameter. Each block is segmented optimally (dynamic programming) into library tokens; the token stream, including an end-of-block token, is coded prequentially with a Krichevsky–Trofimov estimator, each token carries one device reference of $\log_2(\#\text{devices})$ bits, and argument values are coded per (skill, argument) with a literal escape. We accept composites greedily: among contiguous $n$-grams ($2 \leq n \leq 4$) that recur in at least two records, we add the one that most reduces the total and stop when none does. Of 22 candidates, 16 composites are accepted; the total falls from 10150.9 to 9699.3 bits (library 3612.5 $\rightarrow$ 3810.6; corpus 6538.3 $\rightarrow$ 5888.7). The first two accepted composites are `LaundryWasherMode.ChangeToMode` + `OperationalState.Start` ($-124.6$ bits) and `OnOff.On` + `FanControl.PercentSetting` ($-98.5$ bits).

## B.2 LEARNED CONTRACTS

Value ranges of elementary skills come from the device structure or a naming rule, and unit scales are inferred by matching raw values against the numbers in the demonstrator's own reasoning and answer (e.g. 2631 and "26.31 °C"); this matching is the only use of the demonstrators' text. The SimuHome library has 18 elementary and 16 composite skills, 12 informational skills and one control skill (cancelling a workflow), and 13 avoid entries (Table 4). State preconditions are hypothesised from a failure followed by a recovery: a call fails, another call on the same device succeeds, and the attributes of the recovering call's cluster that held at the failure are taken as candidates (a hypothesis counts fully when the failed call is later retried and succeeds, and at half weight when the trace contains no such retry). This yields one state precondition: `OperationalState.Start` requires `OnOff.OnOff = true`, established by `OnOff.On`; it explains 2 failures and holds in all 12 executed successes where the state was known (11 of them established by a prior call), although 11 accepted workflow steps started a washer that had not been powered in the same workflow (their outcome was not observed). Effects come from call responses (e.g. the returned `new_value`); the power effect of `OnOff.On` is inferred from the recovery, and two written attributes are assumed rather than confirmed. Of the 13 avoid entries, 2 involve executed failures (the cooling-setpoint write and a string-valued fan-mode write); the other 11 are workflow steps that the scheduler accepted but that conflict with the observed device structure or command schema. The cooling-setpoint entry records two executed rejections and one rejected workflow, followed by `SetpointRaiseLower` twice and a thermostat-mode write once; two explanations held at every failure (the value was below the heating setpoint, and below the room temperature), and neither was adopted as a precondition. A live check of the library on three development homes, run with the library's own loader (not by the agent), found that skills with executed evidence succeeded in 343 of 347 skill runs, the learned preconditions predicted all 4 failures, and skills seen only inside scheduled workflows succeeded in 92 of 112. In the same check, `OperationalState.Pause` returned a different effect than recorded in 10 of 13 runs, and heating-setpoint writes failed in 8 of 10.

**What the SimuHome agent uses from the library.** Three things: (i) the descriptions of the 18 elementary skills, as the text of the corresponding options; (ii) power prerequisites: the learned state precondition of cycle starts, and, as a heuristic reading of the learned composites, the actions that follow `OnOff.On` in a composite (fan speed, fan mode, thermostat mode, level, wash mode); (iii) the avoid entry for cooling-setpoint writes, which gates the `SetpointRaiseLower` repair. The agent does not use the composite programs, the capability preconditions, the effects, or the learned value scales at inference; its unit conversions are fixed rules (Table 3). The ablation *No learned contracts* removes (i)–(iii).

### B.3 Checklist and Normalization

The checklist is one schema-constrained call (Appendix H) with four worked examples that are not benchmark requests; it is repeated only if the output is not valid JSON. Before the call, number words in clock times and durations are converted to digits ("five oh three PM" $\rightarrow$ 5:03 PM, "nineteen minutes" $\rightarrow$ 19 minutes). After it, deterministic rules re-type, delete or overwrite checklist fields using the request text: (i) the phrases "at $c$, that is $N$ minutes from now", "at $c$, which is $N$ minutes before/after $X\ k$ finishes" and "$X\ k$ will finish around $c \ldots N$ minutes later" ($X\ k$ a numbered device) are parsed into the timing record, keeping both the relative and the stated time; when the request contains a single "$N$ minutes before/after $X$ finishes", its $N$ and relation overwrite those of the goals the model tied to an appliance's finish; (ii) a goal without timing inherits the timing of another goal in the same sentence unless the sentence says "now"; (iii) a question about a room's lighting, temperature, humidity or air whose device mention is generic ("lighting", "air") becomes a room measurement; (iv) in a request that contains a device command, every room-condition goal that does not name a grounded device is dropped, because in such requests these goals restate the situation or purpose that the commands address (in the seed-42 records, every dropped goal was of this kind), and executing them would let the effect model operate devices that the user did not name; the prompt separately asks the model not to turn reasons into goals. A room-condition goal that names a grounded device becomes a device command; (v) a goal on a device that another goal uses as its anchor becomes a time marker at that device's completion if its quote contains "wait", "finish" or "done" or its relation is "when it finishes"; (vi) for a finish-together goal, the running appliance is made the anchor and the other device the target, and the target's untimed goals take the finish-together timing; (vii) exact duplicates are merged. When the decision model's confidence in the direction of a room condition is below 0.3, the checklist's direction is used. All rules were written on the development sets (Appendix C).

### B.4 Typed Decisions

Both question sets use `jev-1.13.0` with Choice questions whose instructions refer to fields of the state by path (e.g. `goals[2].device`); a set with more than 60 questions is split into several requests. The grounding set asks, per goal, "Which device does `goals[i].device` (in `goals[i].room`) refer to?" with the rule "Both the kind of device and its number must match the mention. Choose none if no listed device matches", whose options are the devices of the named room (or of the home) labelled as, e.g., "dimmable light (dimmer light) number 1 in the Living Room", plus *none*; and, where needed, the room, the room measurement, the direction, and the device an anchored timing refers to (a Choice over the devices of the anchor's room, or of the home, plus none). The action set asks "Which single action on ⟨device⟩ accomplishes `goals[i]`?" over the device's interface primitives, each described as "`Cluster.Name`: what it does; options or current value", with a short hint on matching power, fan-percentage and mode goals; one Choice per enumerated argument of each candidate action; and, for a goal that asks about a device, "Which attribute of ⟨device⟩ answers `goals[i].property`?" over its readable attributes with their current values. Numbers never appear as options. In the ablation ***LLM decisions***, the backbone answers the same questions in one schema-constrained call per set, with the option keys as the only allowed values; since it returns a single answer, this ablation has no ranked alternatives for repair.

### B.5 Temporal Network

Time points are seconds from the request time. Besides the constraints of Section 3.4, a pause or stop comes at least a minute after the start of the same device; a device without an earlier action counts "$N$ minutes after the previous action" from the request time (a convention of the benchmark's requests, as is the per-device reading of Section 3.4; we identified both on development episodes); soft clock and completion-time statements must hold within 60 s ($|x - c| \leq 60\,\mathrm{s}$), and "finished by $c$" bounds $e_k \leq c + 60\,\mathrm{s}$. A constraint $c^{\min} \leq y - x \leq c^{\max}$ contributes edges $x \rightarrow y$ with weight $c^{\max}$ and $y \rightarrow x$ with weight $-c^{\min}$; the network is consistent if and only if the distance graph has no negative cycle (Floyd–Warshall). If it is inconsistent, a soft constraint is reported as conflicting if it alone is inconsistent with the hard constraints; if the hard constraints are themselves inconsistent (for example, a cycle that would have to start before now, or a pause before the start), those whose removal restores consistency are reported. Execution times are the earliest times of the hard network; actions due within 15 s run immediately. An appliance's completion is read from

Table 3: Description interface knowledge in the SimuHome agent (all written on development episodes; none reads benchmark labels).

| Component | Rule |
|---|---|
| Option text | One-line descriptions of commands and attributes, used when no learned description exists; robot-vacuum run/clean modes and resume described explicitly; a matching hint in the action question. |
| Option list | Informational clusters (basic information, descriptor, power measurement, ...) and configuration attributes are not offered; `Toggle` is not offered. |
| Units | Temperatures, setpoints and lift positions in hundredths; level clamped to 1–254; fan step direction from "up/down" words; dryness-level names 0–3 from the cluster documentation. |
| Arguments | A mode is set only if the checklist states one; a robot-vacuum start/stop without a stated mode uses a cleaning or idle mode. |
| Grounding | "light $N$" denotes the on/off light; room names matched by normalized string before asking the decision model. |
| Order and power | Power, then settings, then start/resume, then pause/stop/off; power before three media commands. |
| Repair | Power on and retry after a failed call; `SetpointRaiseLower` with $\mathrm{Amount} = (\text{target} - \text{current})/10$ for rejected setpoint writes. |
| Channels | `Channel.ChangeChannel` returns a server error in the simulator, so a channel change writes `Channel.CurrentChannel` with the matching channel-list entry. |
| Room conditions | The effect model of Appendix B.6. |
| Checklist | The normalization rules of Appendix B.3. |

`OperationalState.CountdownTime` together with the current time. To finish together with an appliance, the target's requested settings are applied, its cycle is started, its `CountdownTime` is read, and the cycle is stopped until its planned start; if the duration exceeds the appliance's remaining time by more than 30 s, the request is reported as impossible with both numbers (the probe calls are not undone). If the target has no operational-state cluster, it runs now; if its duration cannot be read, it runs now and is stopped when the appliance finishes.

### B.6 Room Conditions

The public control rules (`get_environment_control_rules`) are used to select, per measurement, the device types in the room that affect it. A description rule effect model mirrors the rules: an air conditioner lowers temperature when it is on, cooling, with a nonzero fan and a setpoint below the room temperature (the agent sets full fan and, if needed, a setpoint 3 °C below the room, within 16–26 °C); a heat pump raises it when heating with a setpoint above the room temperature (setpoint 3 °C above, at most 30 °C); humidifiers, dehumidifiers and air purifiers act when on with a nonzero fan (full fan); lights raise illuminance when on (dimmable lights at level 254), and "too bright" turns lights off. Deactivating a device that pushes the other way is also an option. A dimmable light at level 1 counts as at its limit for "decrease". Device types listed by the rules but not in the model count as unable to help. If no device can help, the agent reports the request as infeasible with the room, the devices it checked, and why each cannot help.

## C Benchmarks, Demonstrations and Development Protocol

Table 4: Demonstration statistics.

| | SimuHome | TimeArena |
|---|---|---|
| Learning records (successes + error-free segments) | 308 (50 + 258) | 169 (54 + 115) |
| Source attempts, successful / unsuccessful | 50 / 205 | 54 / 61 |
| Mean tool calls per record | 4.26 | n/a |
| Tools (SimuHome) or verbs (TimeArena) used | 14 | 44 |
| Learned skills or programs | 18 elementary + 16 composite + 13 informational or control | 27 programs, 188 actions |
| Contracts | 1 state precondition; 13 avoid entries | 168 dependency; 29 verb-level (16 verbs) |
| Held-out device types or tasks | dehumidifier | cooking10, household7, lab5 |

**Demonstrations.** The demonstrations are those used by the ExpeL and Workflow Memory baselines: GPT-5.1 runs of the official ReAct agent on SimuHome episodes generated with the benchmark's generator from seeds 10001 to 10300, disjoint from the evaluation seeds (1 to 125); no demonstration shares a configuration, a home layout or a request with an evaluation episode. A demonstration counts as successful when the benchmark's check scores it 1; the learner sees only the successful trajectories and the error-free segments of unsuccessful attempts. Requests or tool calls that read or operate a dehumidifier were excluded, so the dehumidifier is a held-out device type. On TimeArena the demonstrations are single-task episodes of the 27 tasks other than cooking10, household7 and lab5.

**Development protocol.** All development of the agent (checklist prompt, normalization rules, question wording, repair rules) used two development sets drawn from the unused generated SimuHome episodes, which share no configuration, home layout or request with the evaluation set and are not demonstration episodes (a few share a generation seed, and thus a home, with a demonstration when a cell had too few other episodes): a first set of 96 episodes (8 per cell) and a second set of 91 episodes drawn at random from the remaining pool (8 per cell, 3 for the scarce QT4-1 feasible cell), held out until the first had been used and then also used for development. The evaluation episodes were run only after the code was frozen, once per backbone and seed. During the evaluation runs we changed only the language-model client's request handling, twice: its own in-flight limit, which duplicated the harness's and could deadlock, was removed, and the choice between the two endpoints serving each backbone was decoupled from the sampling seed (it had sent every first call of a seed to the same endpoint). Both changes leave the agent's logic unchanged; episodes completed before a change were kept, and unfinished ones were rerun from the start. A transient authorization error of the decision model's API (HTTP 403) interrupted 145 episodes (144 of the ablation runs and one of the evaluation runs), which were rerun from the start with unchanged code. With Gemma 4 12B, success on the first development set rose from 79.2% to 96.9% during development; on the second, before it was used for development, the agent scored 86.8%. On TimeArena, development used 30 two- and three-task configurations whose task sets differ from every test configuration and exclude the three held-out tasks (individual tasks recur in test); cooking9, household6 and lab4 were also removed from the development learner, and further seen tasks were withheld in some runs, to test unseen-task handling. During TimeArena development the learner's dependencies were also compared with the simulator's task definitions as a diagnostic; neither the learner nor the agent reads them. We also found two interface problems during development that affect every method. On SimuHome, `Channel.ChangeChannel` fails with a server error, so the agent sets `Channel.CurrentChannel` to an entry of the channel list instead. On TimeArena, in all ten cooking three-task test configurations and three of the ten cooking pairs of TimeArena, two tasks share an ingredient name, and each such task's dish then accepts only its own copy of the shared ingredient, so its other ingredients cannot be added; this caps the completion rate, macro-averaged over scenarios, at 90% for two tasks and 66.7% for three (70% and 0% on cooking). Adding an ingredient to another task's dish would circumvent this, but violates the "(Only for Task $k$)" instruction, and the agent never does it.

## D Implementation and Baselines

All runs use a harness that reproduces the baselines' evaluation (the baseline runs were made separately, before ours): one fresh simulator per episode, the pinned official tool functions called through the same code path, the same runner and evaluators, and the same LLM judge panel (three `gpt-5-mini-2025-08-07` judges, temperature 1.0, medium reasoning effort, majority vote) for QT1 feasible and all infeasible episodes. Backbones are served with vLLM (bf16, thinking disabled) and sampled with temperature 0.7, top-$p$ 1.0, the run seed, and at most 1,536 output tokens per call; at most six requests are in flight per endpoint, as for the baselines. Seeds 42, 43 and 44 change the language model's sampling; decision-model answers are cached by their exact input, so identical requests receive identical answers. Our agent makes one checklist call (repeated only if the output is not valid JSON) and, when the reply consists only of room measurements, one call for the interpretive sentences; it sends at most two decision-model question sets. It is limited to 400 tool calls and 900 s per episode. Baseline settings are those of the original experiments. On SimuHome, ReAct and Reflexion (at most three trials) take at most 40 steps, and ExpeL and Workflow Memory are built from the same GPT-5.1 demonstrations with their official code and use the same ReAct

actor and limits. Reflexion receives only whether a trial failed as feedback, and we report its final trial. On TimeArena, the same limits count decisions instead of steps (at most 40 per episode, a wait counting as one), within the benchmark's time limits of 40, 80 and 120 minutes for one, two and three tasks. Self-plan is TimeArena's official SelfPlan agent; the SimuHome version mentioned in Section 4.1 is our port, which adds two planning calls before the ReAct loop. All baselines sample with temperature 0.7, top-$p$ 1.0 and at most 1,536 output tokens per call, with a timeout of 900 s per episode.

# E TIMEARENA DETAILS

The learner reads the 54 successful traces and 115 error-free segments of unsuccessful attempts. It also takes pre-state words from the action descriptions in the task instructions. Durations come from "it will take $N$ minutes"; an action is *blocking* if the environment keeps the agent occupied and *background* if the agent can act while it runs. Dependencies come from three sources: 42 refusals of the form "Because X is Y", which resolve to 20 distinct dependencies on the action that takes X out of state Y (matched by the learned pre-state, else the earliest candidate in the program); a rule that links two actions sharing an object that is not pooled (a pool is a set of duplicated objects without an "(Only for Task $k$)" marker, which any task may use) when the first finishes before the second starts in every successful trace and no trace overlaps or reverses them; and 12 pairs on different objects, ordered the same way in every successful trace, that the decision model judged dependent ($p \geq 0.5$; on demonstration pairs this threshold recalled 15 of 20 dependencies shown by refusals with no false positive among 106 independent pairs). The library has 27 programs, 188 actions, 168 dependency contracts, and 29 verb-level contracts over 16 of its 44 verbs. For an unseen task, one language-model call proposes the program from the task's description, templates and objects, given two retrieved programs of the same scenario; code ranks groundings for any template the proposal leaves unused (template completion; all 27 learned programs use every listed template), and the decision model picks one; code then replaces implausible containers, orders each object's actions from its initial state, and applies the verb-level contracts; the decision model classifies unseen verbs as background when $p \geq 0.30$ (a threshold set on the demonstration verbs). At a refusal, code finds the prerequisite action or creates one from the templates, and the decision model breaks remaining ties; a refusal that names only the container (a behaviour of the simulator) is treated as a possibly unmet prerequisite of the first object. When nothing is ready, the agent tries once an action whose unmet dependencies all lack refusal evidence (probing), else waits while a background action runs, else tries the least-blocked pending action; after three consecutive refusals it issues only actions not refused before (the harness ends an episode after five). If the plan is exhausted while an episode with an unseen task continues, the decision model proposes one more grounded action (at most six times).

# F ADDITIONAL RESULTS

## F.1 FAILURE ANALYSIS

Table 5: Failures of our agent on SimuHome, seed 42, by the first stage that went wrong (post-hoc analysis of the evaluation records; counts of failed episodes).

| | Gemma 4 12B (46) | | Qwen 3.5 9B (66) | |
|---|---|---|---|---|
| Failure source | F | IF | F | IF |
| Checklist (goal, value or timing not extracted) | 7 | 7 | 14 | 12 |
| Reply judged insufficient (behaviour correct) | 0 | 13 | 5 | 7 |
| Benchmark label contradicts the request | 4 | 3 | 2 | 4 |
| Room-condition effect model | 1 | 1 | 1 | 6 |
| Action or argument selection | 4 | 0 | 5 | 0 |
| Execution or interface | 3 | 0 | 4 | 0 |
| Temporal network | 3 | 0 | 4 | 0 |
| Grounding | 0 | 0 | 0 | 1 |
| Other | 0 | 0 | 0 | 1 |

Each failed episode of seed 42 was assigned to the first stage that went wrong, from its checklist, decisions, tool calls, reply and the benchmark's goals (Table 5). The checklist is the main source for both backbones, mostly omitted or merged goals and timing phrases the rules do not cover (for example, a stated time and a relative time in different sentences). The second source is the reply: in infeasible room-condition episodes the agent refused correctly but its template reason ("the heat pump is not heating, so it cannot lower the temperature") did not state that the room has no cooling device or the current temperature, which the judges required; with Qwen, the interpretive sentence that the language model appends to a measurement report sometimes contradicts the reported value ("the product ID is not available"). Grounding by the decision model accounts for one failure in 112. Several failures are due to the benchmark: gold labels that contradict the request ("the air is too clean" labelled as a pm10 decrease; a channel label that does not match the channel list), and room-condition checks against a counterfactual run in which an already-running purifier ends lower than after our change, because the simulator fixes a purifier's rate from the concentration at the time of the last change. Temporal failures come mostly from finish-together requests for robot vacuums, whose cycle duration is not exposed before the run starts, so the agent starts them at once.

### F.2 Per-cell Results with Standard Deviations

Table 6: SimuHome success of Ours (%), mean ± population standard deviation over seeds 42, 43, 44.

| Backbone | QT1 | | QT2 | | QT3 | | QT4-1 | | QT4-2 | | QT4-3 | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | F | IF | F | IF | F | IF | F | IF | F | IF | F | IF | Held-out | Overall |
| Gemma 4 12B | 92.7 ±0.9 | 96.7 ±0.9 | 90.7 ±0.9 | 78.0 ±0.0 | 98.0 ±0.0 | 100.0 ±0.0 | 97.3 ±0.9 | 100.0 ±0.0 | 92.0 ±0.0 | 95.3 ±0.9 | 89.3 ±0.9 | 83.3 ±0.9 | 98.3 ±0.0 | 92.8 ±0.3 |
| Qwen 3.5 9B | 76.0 ±1.6 | 96.7 ±0.9 | 91.3 ±0.9 | 79.3 ±0.9 | 95.3 ±0.9 | 100.0 ±0.0 | 90.7 ±1.9 | 100.0 ±0.0 | 90.7 ±0.9 | 95.3 ±4.1 | 85.3 ±3.4 | 72.7 ±2.5 | 96.6 ±0.0 | 89.4 ±0.3 |

### F.3 Cost

Table 7: Cost per SimuHome episode: language-model (backbone) calls and tokens (thousands), invalid tool calls, agent time (seconds, the agent phase including model and tool calls), tool calls, and typed-decision requests. Baselines with Gemma 4 12B; for Reflexion, calls and tokens include all trials, and time and tool calls are those of its final trial. The baselines make no decision-model requests.

| Method | LLM calls | Tokens (K) | Invalid calls | Agent time | Tool calls | Decisions |
|---|---|---|---|---|---|---|
| ReAct | 6.85 | 44.69 | 0.23 | 23.0 | 5.6 | 0 |
| Reflexion | 15.68 | 103.97 | 0.50 | 22.8 | 5.6 | 0 |
| ExpeL | 6.44 | 80.33 | 0.14 | 66.2 | 5.0 | 0 |
| Workflow Memory | 6.67 | 47.81 | 0.20 | 65.0 | 5.5 | 0 |
| Ours (Gemma 4 12B) | 1.06 | 3.35 | 0.01 | 10.3 | 17.2 | 1.63 |
| Ours (Qwen 3.5 9B) | 1.06 | 3.29 | 0.01 | 8.3 | 17.1 | 1.62 |

The agent phase takes 10.3 s per episode (median 8.5 s), against 23.0 s (median 21.3 s) for ReAct and 66.2 s (median 32.9 s) for ExpeL, despite more tool calls (17.2 against 5.6); the baseline times come from separate runs and depend on endpoint load (with Qwen: 8.3 s against 20.2 s and 22.6 s). Of our 10.3 s, 8.5 s are language-model time, nearly all in the checklist call. When the backbone answers the typed questions in place of the decision model (*LLM decisions*), each decision request takes 5.4 s on average (median 2.2 s) instead of 0.22 s, and the agent makes 2.68 instead of 1.06 language-model calls per episode.

### F.4 TIMEARENA BY SCENARIO AND SEEN TASKS

Table 8: TimeArena three-task episodes by scenario, Gemma 4 12B, with the benchmark's metrics (AS, CS, CR, CT). No agent completes a cooking three-task episode because of the shared-ingredient defect (Appendix C), so CT is undefined there (–); the Oracle, computed offline from the dependency graphs, is not affected by the defect.

| | Cooking | | | | Household activity | | | | Laboratory work | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Method | AS↑ | CS↑ | CR↑ | CT↓ | AS↑ | CS↑ | CR↑ | CT↓ | AS↑ | CS↑ | CR↑ | CT↓ |
| ReAct | 79.33 | 1.65 | 0.00 | – | 91.23 | 2.34 | 56.67 | 43.21 | 92.70 | 2.03 | 43.33 | 45.84 |
| Reflexion | 78.60 | 1.74 | 0.00 | – | 98.97 | 2.45 | 90.00 | 41.11 | 99.73 | 2.22 | 93.33 | 44.93 |
| Self-plan | 38.10 | 1.44 | 0.00 | – | 65.27 | 2.23 | 23.33 | 42.56 | 76.07 | 1.95 | 36.67 | 43.14 |
| ExpeL | 60.17 | 1.63 | 0.00 | – | 91.17 | 2.49 | 70.00 | 40.24 | 90.97 | 2.15 | 70.00 | 44.56 |
| Workflow Memory | 69.40 | 1.56 | 0.00 | – | 95.70 | 2.19 | 60.00 | 45.34 | 97.27 | 2.14 | 73.33 | 44.53 |
| Ours | **89.10** | **1.82** | 0.00 | – | **100.00** | **2.65** | **100.00** | **37.77** | **100.00** | **2.88** | **100.00** | **34.70** |
| Oracle | 100.00 | 1.90 | 100.00 | 52.50 | 100.00 | 2.82 | 100.00 | 35.40 | 100.00 | 2.88 | 100.00 | 34.70 |

Table 9: TimeArena single-task episodes of the 27 tasks covered by the demonstrations, Gemma 4 12B. Bold: best among agents.

| Method | AS↑ | CS↑ | CR↑ | CT↓ |
|---|---|---|---|---|
| ReAct | 98.05 | 5.10 | 95.06 | 19.77 |
| Reflexion | **100.00** | 5.15 | **100.00** | 19.43 |
| Self-plan | 77.09 | 5.05 | 58.02 | 18.27 |
| ExpeL | 97.12 | 5.15 | 92.59 | 19.33 |
| Workflow Memory | 96.44 | 5.09 | 91.36 | 19.65 |
| Ours | **100.00** | **6.18** | **100.00** | **16.19** |

## G LEARNED SKILL LIBRARY (EXAMPLES)

```
cmd:OperationalState.Start            (elementary; 32 occurrences: 13 executed successes,
                                        2 executed failures, 17 scheduled)
  parameters : device : has_command(OperationalState, Start)
  program    : execute_command(device, ep(OperationalState), OperationalState, Start, {})
  pre        : OnOff.OnOff = true      established by cmd:OnOff.On
               (failure followed by recovery in records 42, 46; holds in 12/12 executed
               successes with known state)
  effect     : OperationalState.OperationalState = 1   (call response, support 13)

comp:OnOff.On+FanControl.PercentSetting   (composite; accepted by description length, -98.5 bits)
  parameters : device : has_command(OnOff, On) & writable(FanControl.PercentSetting)
               PercentSetting : int, range [0, 100] by naming rule   (observed 30..100)
  program    : execute_command(device, ep(OnOff), OnOff, On, {});
               write_attribute(device, ep(FanControl), FanControl, PercentSetting, $PercentSetting)

write:Thermostat.SystemMode           (elementary; 22 occurrences, 18 successful:
                                        14 executed, 4 accepted in workflows)
  parameters : SystemMode : enum {OFF=0, AUTO=1, COOL=3, HEAT=4}; integer codes and names accepted
  effect     : Thermostat.SystemMode = $SystemMode   (call response, support 14)

avoid write:Thermostat.OccupiedCoolingSetpoint   (2 executed rejections, 1 rejected workflow)
  alternatives observed after failure: cmd:Thermostat.SetpointRaiseLower (2), write:Thermostat.SystemMode (1)
  explanations holding at every failure (not promoted): value < OccupiedHeatingSetpoint,
                                                        value < LocalTemperature
```

## H PROMPTS

**Goal checklist (system prompt).** The four worked examples and the JSON schema are in the supplementary material.

```
You turn a smart-home request into a checklist of atomic goals. Output JSON only.

One goal = ONE thing to report or ONE setting/command on ONE device (or one room condition to change).
Split compound requests: "turn on the humidifier and set its fan to 30 percent" is two goals
(power on; fan 30 percent). "put the AC on cooling with the fan at 70%" is three goals if power-on is
```

```
asked, otherwise two (cooling mode; fan 70 percent). Repeat the room and device mention in every goal
that concerns them.

Fields (copy words from the request; do not invent ids):
- quote: the exact words of the request this goal comes from.
- kind:
  report_room_state   = tell the user a room measurement (temperature, humidity, brightness/illuminance, air
      quality/dust/PM10).
  report_device_info  = tell the user an attribute/setting/status/capability of a named device.
  control_device      = change a named device (power, mode, level, fan, position, start/pause/stop, ...).
  adjust_room_condition = the user complains about or wants a change in a room condition without naming a
                        device command (too hot, too dim, stuffy, too humid, too bright ...).
- room: the room as written ("utility room", "living room"), or null if none is stated for this goal.
- device: the device as written INCLUDING its number ("dimmer light 1", "AC 1", "washer 2", "blinds 1"), or
      null.
- property: what is read or set ("illuminance", "humidity", "fan speed", "product ID", "spin speeds", "level",
  "mode", "operational state", "position"), or null.
- action: the operation as written ("turn on", "turn off", "set", "start", "pause", "stop", "resume",
  "change mode", "close", "open", "raise", "lower"), or null for reports.
- value: the target number for this goal exactly as stated (30 for "30 percent", 65.0 for "65% closed",
  40 for "level 40", 24 for "24 degrees"). Words with a clear number: "fully closed" = 100, "fully open" = 0,
  "max"/"full" fan = 100. Otherwise null.
- value_text: non-numeric target words ("Heavy", "cooling", "Normal dryness", "Max", "strong breeze", "low"),
      or null.
- direction: for adjust_room_condition, the change the user WANTS: "decrease" when it is too hot/bright/humid/
      dusty
  (they want less), "increase" when it is too cold/dim/dry; null otherwise.
- timing: when the goal should happen.
  type "now": immediately (also when no time is given).
  type "in_minutes": N minutes from now -> minutes = N.
  type "minutes_after_previous": N minutes after the previous timed action in the request -> minutes = N.
  type "at_clock_time": only a clock time is given -> clock_time.
  type "relative_to_device_finish": tied to when another device (the anchor) finishes -> anchor_device,
     anchor_room, anchor_relation ("after" = N minutes after it finishes, "before" = N minutes before it
      finishes,
     "when_finishes" = exactly when it finishes (e.g. pause when it finishes), "finish_together" = time this
      goal so
     that this device finishes at the same time as the anchor, "finish_by" = must be finished by a clock time)
      ,
     minutes = N or null.
  clock_time: ALWAYS copy any clock time stated for this goal (e.g. "9:07 AM", "1:59 PM"), even when minutes
      or
     an anchor are also given ("At 9:07 AM, that is 16 minutes from now" -> type in_minutes, minutes 16,
     clock_time "9:07 AM"). null if no clock time is stated.
  anchor_finish_clock: if the user states WHEN the anchor finishes ("the dishwasher will finish at about 3 10
      PM"),
     copy that clock time here; otherwise null.
  For goals without timing use {"type":"now","minutes":null,"clock_time":null,"anchor_device":null,
  "anchor_room":null,"anchor_relation":null,"anchor_finish_clock":null}.
Statements that only give timing or context ("wait for the dishwasher to finish", "the washer finishes at 2 PM
      ",
"I will be back in an hour") are NOT goals: put that information into the timing of the goals it qualifies.
Reasons and purposes ("so the air clears", "I need the room ready before I shower", "to help items firm up")
explain WHY; they are NOT goals unless they are the only request. Do not add goals the user did not ask for.
      Report questions ("is it dim?") are report goals, not control goals.
"Set X to Running state" means start X (action "start").
```

**Interpretive sentence.** "You help a smart-home assistant. Given the user's request and measured values, write ONE or TWO short sentences that answer any descriptive part of the question (e.g. whether a room is bright or dim, warm or cool, humid or dry, dusty or clean) using the numbers. Talk only about the listed room measurements; do not mention devices, actions, or anything not in the values. Output only the sentences."