PersonalHomeBench: Evaluating Agents in Personalized Smart Homes

Anonymous ACL submission

Abstract

Agentic AI systems are rapidly advancing toward real-world applications, yet their readiness in complex and personalized environments remains insufficiently characterized. To address this gap, we introduce **PersonalHomeBench**, a benchmark for evaluating foundation models as agentic assistants in personalized smart home environments. The benchmark is constructed through an iterative process that progressively builds rich household states, which are then used to generate personalized, context-dependent tasks. To support realistic agent–environment interaction, we provide **PersonalHomeTools**, a comprehensive toolbox enabling household information retrieval, appliance control, and situational understanding. PERSONALHOMEBENCH evaluates both reactive and proactive agentic abilities under unimodal and multimodal observations. Thorough experimentation reveals a systematic performance reduction as task complexity increases, with pronounced failures in counterfactual reasoning and under partial observability, where effective tool-based information gathering is required. These results position PERSONALHOMEBENCH as a rigorous evaluation platform for analyzing the robustness and limitations of personalized agentic reasoning and planning.

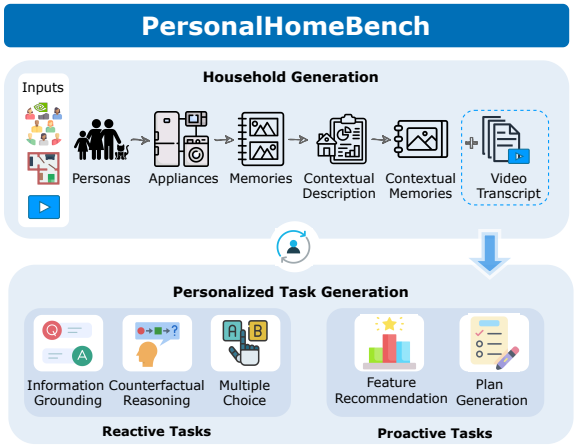


Figure 1: **Overview of the PERSONALHOMEBENCH data generation pipeline.** Detailed household environments are assembled from personas, devices, memories, and contextual descriptions, with optional video grounding. Five categories of personalized tasks are then generated from these environments, spanning reactive question answering and proactive assistance

1 Introduction

Agentic AI systems aim to move beyond passive prediction toward autonomous assistance in everyday environments, where robust performance depends on contextual reasoning, uncertainty handling, and personalization. Although a growing body of benchmarks have been proposed to evaluate agentic systems, most existing evaluations focus on generic tasks in simplified or simulated settings (Chang et al., 2024a; Bonatti et al., 2025; Xi et al., 2025). As a result, they offer limited insight into model behavior under evolving individualized preferences, long term context, and partial observability, yielding an incomplete assessment of readiness for personalized real world deployment.

Smart home environments provide a natural and challenging testbed for exposing these limitations. Effective operation in this domain requires reasoning over heterogeneous devices, maintaining persistent and temporally grounded context, and adapting to evolving user preferences while accounting for safety, efficiency, and operational constraints. Prior smart home benchmarks have largely relied on simplified abstractions, such as text only environments with limited device coverage and primitive actions, or narrowly scoped tasks focused on isolated device control (Li et al., 2025; Bartkowiak and Podstawski, 2025). Viewing smart homes as a realistic setting for personalized agentic intelligence, we introduce **PersonalHomeBench**, a framework for evaluat-

ing foundation models in household environments that demand contextual reasoning, personalization, and proactive decision making beyond reactive task execution.

PersonalHomeBench adopts a staged environment construction process that reflects real household evolution, sequentially instantiating resident personas, assigning device ecosystems, and accumulating contextual signals over time to synthesize coherent household worlds, with optional multimodal grounding. As illustrated in Fig. 1, five categories of personalized tasks are derived from these environments to support unified evaluation of reactive reasoning and proactive planning, using a procedure inspired by (Singh et al., 2024). These include three reactive question answering tasks, INFORMATION GROUNDING, COUNTERFACTUAL REASONING, and MULTIPLE CHOICE, and two proactive tasks, FEATURE RECOMMENDATION and PLAN GENERATION, which target appliance- and routine-level orchestration for timely, preference-aware home assistance.

To support faithful evaluation of agentic behavior, we introduce **PersonalHomeTools**, a smart home toolbox that provides a grounded interaction layer between agents and realistic household environments. It exposes structured household information retrieval, persistent memory and event access, environmental perception, and realistic appliance control through domain-specific APIs. Together, these capabilities enable systematic study of agents that unify perception, memory, and tool use for personalized decision making.

We evaluate a wide spectrum of foundation models, spanning compact four billion parameter models suitable for privacy-preserving edge deployment to cloud-based large-scale models. Beyond task-specific metrics, we employ a Role-Playing LLM (Wang et al., 2024) as a Judge to estimate projected satisfaction for individual household members. Across model scales, performance degrades sharply on tasks involving tool use, long-horizon planning, and advanced reasoning, often accompanied by increased hallucination. These findings reveal fundamental limitations of current systems and highlight the difficulty of achieving robust, personalized, and proactive behavior in realistic smart home settings, establishing PERSONALHOMEBENCH as a practical resource for diagnosing failures and guiding the development of reliable personalized agents.

Our contributions are threefold:

- We introduce *PersonalHomeBench*[1], a benchmark for studying personalized and proactive agent behavior in contextually grounded smart home environments.
- We present *PersonalHomeTools*, a comprehensive smart home toolkit for realistic agent evaluation.
- We conduct systematic evaluation across model scales and reasoning levels, and leverage a Role Playing LLM-as-a-Judge for assessing personalized satisfaction.

2 Related Works

Table 1: Comparison of benchmarks along five dimensions: HOME TYPE, personalization (PERS.), proactiveness (PROACT.), agentic interaction with tools (AGENTIC), and multimodal support (MM). SIM = simulated homes, REAL = real-home data, and NONE = non–home-based settings. * marks settings where multimodal content is represented as text for inference.

METHOD	HOME-TYPE	PERS.	PROACT.	AGENTIC	MM
EMBODIED BENCHMARKS	SIM	✗	✗	✓	✓
TOYOTA SMARTHOME	REAL	✗	✗	✗	✓
PERSONALTRAVELPLANNER	NONE	✓	✗	✓	✗
HOMEBENCH	SIM	✗	✗	✓	✗
EDGEWISEPERSONA	SIM	✓	✓	✓	✗
CONTEXTAGENT	NONE	✓	✓	✓	✓*
PERSONALHOMEBENCH (OURS)	SIM+REAL	✓	✓	✓	✓

As summarized in Table 1, embodied household benchmarks (Puig et al., 2018; Shridhar et al., 2019; Li et al., 2024; Chang et al., 2024b; Yang et al., 2025c) focus on navigation and atomic action execution in simulated 3D environments, but do not model real smart-device ecosystems or personalization. Toyota Smarthome (Das et al., 2019) offers real-world home video data, but is limited to narrow demographics and fixed environments, and focuses on activity recognition rather than personalized, proactive, or interactive agentic assistance.

Recent benchmarks incorporate aspects of tool usage and personalization. PersonalTravelPlanner (Singh et al., 2024) studies personalized planning with simple tools outside the home domain. HomeBench (Li et al., 2025) evaluates reactive smart-home command validation with limited appliance coverage and restricted operations, while EdgeWisePersona (Bartkowiak and Podstawski, 2025) focuses on profile reconstruction and lightweight routine inference over a small device set, with only minimal personalization via short persona descriptions. ContextAgentBench (Yang et al., 2025b) introduces proactive agents

[1]Dataset and code will be released

with contextual information and tools, but relies on ego-centric traces, generic APIs, and language-only inference with limited personalization.

PERSONALHOMEBENCH focuses on agent behavior in realistic home settings where personalization, long-term context, perception, and smart-device interaction jointly influence decisions. By integrating with PERSONAHOMETOOLS, it evaluates whether agents can retrieve relevant household information, reason over persistent state, and coordinate appliance actions to provide proactive, personalized assistance. As summarized in Table 2, no existing benchmark simultaneously covers real home data, personalization, proactiveness, domain-specific tools, and multimodality, making *PersonalHomeBench* a unified setting for studying personalized agentic behavior in smart home domain. Additional discussion of related benchmarks is provided in Appendix B.

3 PersonalHomeBench

An overview of the PERSONALHOMEBENCH generation pipeline is shown in Figure 1. The individual components are detailed in the subsequent sections.

3.1 Household Generation

As illustrated in the *Household Generation* module of Fig. 1, generation begins from a minimal scaffold specifying household type, layout, and coarse demographics, and is seeded with personas from Nemotron Personas (Meyer and Corneil, 2025) to instantiate realistic occupants. These personas, together with household metadata, are used by a data generation model to produce complete occupant profiles that conform to a predefined schema (Appendix F). Household context is progressively enriched with appliances, historical memories, and updated occupant and appliance states, followed by construction of a compact relevant-memory cache for personalized task generation.

For the multimodal setting, we further ground household context using exocentric videos captured by cameras placed across rooms (Appendix C.2). We produce structured video transcripts containing entity references, object mentions, timestamped events, and aligned audio captions (cf. F for schema). To preserve fidelity to visual evidence while supporting reasoning beyond direct observation, we maintain separate representations for video-grounded and inferred information.

All generated data undergo manual review to ensure realism, internal consistency, and schema compliance, yielding high-quality household environments used for the personalized task generation.

3.2 Personalized Task Generation

We derive five categories of personalized tasks that probe complementary aspects of reasoning and planning in smart home settings. Task generation assumes full observability, providing the model with the complete household context.

Reactive tasks are framed as closed-form question answering and include INFORMATION GROUNDING (IG), COUNTERFACTUAL REASONING (CF), and MULTIPLE CHOICE (MC). These tasks assess the ability of an agent to extract and integrate household information across varying levels of difficulty, ranging from direct factual lookup to multi-hop reasoning over numerous entities and attributes. To assess factual grounding, we employ IG for open-ended generation and MC for constrained selection.

CF extends IG by introducing paired counterfactual variants, one that changes the ground-truth answer (*CHG*) and one that leaves it unaltered (*ORG*), enabling targeted analysis of robustness to hypothetical perturbations. For example, changing temporal context should update a resident's age, while altering unrelated attributes like attire should not.

Proactive tasks evaluate open-ended, subjective decision making for personalized assistance, testing whether agents can anticipate user needs and initiate appropriate actions without explicit queries. We define two such tasks, FEATURE RECOMMENDATION (FR) and PLAN GENERATION (PG), for appliance and routine orchestration.

For FR, the agent ranks appliance features by contextual relevance using API-level inventories, reflecting realistic feature discovery and configuration. In PG setting, agents generate multi-step plans that coordinate heterogeneous devices toward a shared goal, such as initiating first-aid assistance, notifying caregivers, and shutting down hazardous appliances in safety-critical situations (Fig. 2).

3.3 Dataset Statistics

Household Data. *PersonalHomeBench* includes 1,100 households generated using the procedure in Sec. 3.1, comprising over 2,000 unique household member profiles. The households exhibit broad demographic and structural diversity, with balanced age and gender distributions (Fig. 3a). Most house-

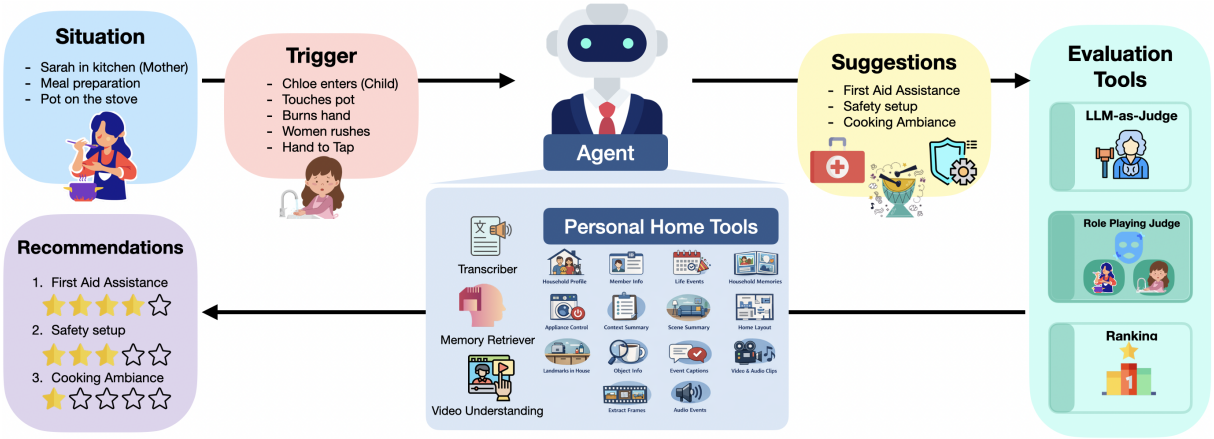


Figure 2: An agent interprets real-world household situations and triggers, using PersonalHomeTools to generate proactive plans which are evaluated by the Role Playing Judge to produce personalized recommendations.

holds do not own pets, while pet ownership is most popular among families (Fig. 3b). Furthermore, we ground 100 households in real-home videos spanning *Health*, *Safety*, and *Daily Care* scenarios, with the video length distribution shown in Fig. 3c. Member profiles contain rich persona attributes, including hobbies, lifestyles, and preferences, supporting fine-grained personalization; frequently occurring attributes are summarized in Fig. 3d. Additional analyses appear in Appendix C.

Personalized Task Data. *PersonalHomeBench* contains a total of **9,168** task instances, comprising 6,000 text-only examples and 3,168 multimodal examples. The text-only split contains 1,000 instances each for IG, MC, FR, and PG, and 2,000 instances for CF. For multimodal grounding, the dataset contains 742 instances each of IG and MC, 1,484 CF instances, and 100 instances each of FR and PG. Reactive tasks (IG, CF, MC) are further annotated with three difficulty levels (easy, medium, hard). Compared to existing benchmarks on personalized and proactive agentic systems (Xie et al., 2024a; Singh et al., 2024; Yang et al., 2025b; Bartkowiak and Podstawski, 2025), *PersonalHomeBench* offers broader coverage in scale, task diversity, and supported input modalities, enabling a more comprehensive evaluation of agentic behavior in realistic settings.

4 PersonalHomeTools

As depicted in Fig. 2, *PersonalHomeTools* serves as a unified interaction layer that enables agents to perceive, reason about, and act within household environments. The toolbox supports household information retrieval, appliance control, and contextual understanding.

Household information retrieval provides access to relatively static metadata about occupants, pets, appliances, and other household entities through lightweight tools with deterministic structured outputs.

Appliance control tools allow agents to modify the state of smart devices within the household. We implement feature-level interfaces for more than 40 smart appliances using public documentation from major smart home providers[2]. Interactions may be deterministic, such as power control, or mediated by specialized agents such as personalized recommendation engines.

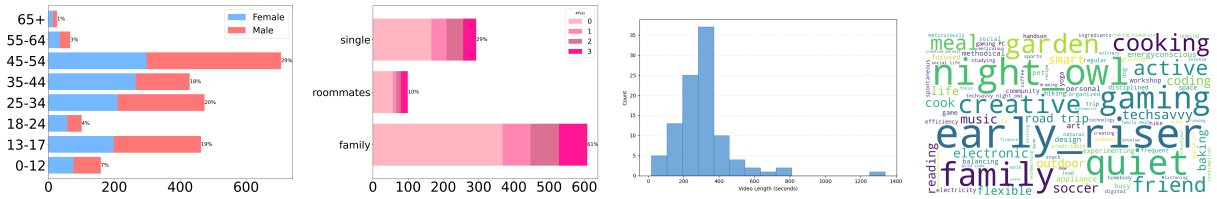
To support deep **contextual understanding**, the framework includes three specialized agents: a *Transcriber*, a *Memory Retriever*, and a *Video Understanding Agent*. The Transcriber incrementally maintains household state by extracting memories, events, and contextual attributes from observations. The Memory Retriever surfaces relevant household and occupant-specific memories, while the *Video Understanding Agent* retrieves clips, frames, and audio cues for multimodal grounding.

Together, these components provide a coherent interface for contextual reasoning, persistent memory retrieval, and personalized action execution. A complete list of supported appliances, features, and tools is provided in Appendix J.

5 Experimental Setup

This section presents the evaluation protocol for assessing models on *PersonalHomeBench*, encompassing the experimental settings, and metrics.

[2]LG Electronics and Samsung.



(a) Demographics (b) Pets (c) Video Lengths (d) Attributes of Personas

Figure 3: Statistics of generated household and persona diversity in PersonalHomeBench. (a) Age and gender. (b) Pets per household type. (c) Video lengths (sec). (d) Word cloud of persona attributes.

5.1 Models and Hyperparameters

Models. We benchmark eleven models spanning a wide range of size tiers, architectures, and reasoning configurations. The evaluated models are grouped into three categories:
(1) *Large proprietary models*: GPT-4 (Achiam et al., 2023), GPT-4o (Hurst et al., 2024), and Gemini 2.5 Pro (Comanici et al., 2025), (2) *Medium-size open models*: Qwen3-30B-Instruct, Qwen3-30B-Thinking (Yang et al., 2025a), Nemotron3-Nano-30B-A3B (Blakeman et al., 2025), and GPT-OSS-20B (Agarwal et al., 2025). (3) *Small-size open models*: Qwen3-4B and Qwen3-4B-Thinking (Yang et al., 2025a) (c.f. Appendix E).

We evaluate Qwen3 models in thinking mode and GPT-OSS-20B at low, medium, and high reasoning levels, denoted by a single-letter tag (e.g., Qwen3-4B [T], GPT-OSS-20B [L/M/H]).

We use Gemini 2.5 Pro as the primary model for dataset generation and as the evaluation backbone for all LLM-as-a-Judge protocols. To reduce potential evaluation bias, we report the performance of Gemini 2.5 Pro separately from that of the other evaluated models.

Hyperparameters. We use temperature 0.6 for dataset generation to encourage diversity, and temperature 0.0 with top-$p = 1.0$ during evaluation for deterministic and reproducible outputs. Each task is allotted up to 15 turns to finish.

In addition to the agentic inference, we evaluate models in a *Sole-Reasoning* setting, where the full household context is directly provided to the model without requiring any tool invocation.

5.2 Evaluation Metrics

Reactive Tasks. For these deterministic question answering tasks, we report accuracy (ACC) (Appendix Eq. 3 & 6). For IG and CF, correctness is first assessed using string-level exact match. When an exact match fails, we employ a semantic equivalence judge (Xie et al., 2024a). This hybrid approach is necessary because valid responses may vary in surface form, ranging from short phrases to full sentences. For CF, we report accuracy for changed-answer (*CHG*) and unchanged-answer settings (*ORG*). For MC, accuracy is computed via exact match against the ground-truth option.

Proactive Tasks. The Feature Recommendation task is formulated as a ranking problem and evaluated using MAP@1 (Zhu, 2004). For Plan Generation, we report both RPJ score to assess preference-aligned plan quality and a *validity* metric, defined as the proportion of sub-program steps that are executable with correct tools and well-formed arguments (Appendix Eq. 11).

Role-Playing Judge. Evaluating PG requires capturing subjective satisfaction under personalized household preferences. Following the established practice of using LLMs as judges for plan quality evaluation (Xie et al., 2024a; Singh et al., 2024), we introduce RPJ, a Role-Playing LLM-as-a-Judge framework in which the judge evaluates plans from the perspectives of individual household members and the household collectively.

RPJ operates in two modes. In ABS mode, the judge assigns a scalar score and textual feedback to each sub-program within a plan, enabling fine-grained analysis of plans that mix strong and weak components; we report average RPJ_{ABS} for multimodal PG. In REL mode, the judge ranks complete plans across models according to household preferences, framing evaluation as a preference comparison that aligns naturally with RLHF-style objectives (Comanici et al., 2025). RPJ_{REL} is also more efficient at scale: evaluating N models requires $N\times$ calls under RPJ_{ABS} but only a single pass under RPJ_{REL}. For Table 2, RPJ_{REL} ranks all 11 candidate plans per instance and we report win rate, the fraction of instances a model is ranked first. RPJ algorithm and metric definitions are provided in Appendix D.

Table 2: Model performance sorted by size. Proprietary models are shaded. Bold and underline mark the best and second-best non Gemini 2.5 Pro results. Confidence intervals are reported in Appendix D.5.

Model	Input Modality	Information Grounding (ACC)		Counterfactual Reasoning (ACC)		Multiple Choice (ACC)		Feature Reco. (MAP@1)		Plan Gen. (RPJ_{REL})		Plan Gen. (Validity)	
		No Tools	With Tools	No Tools	With Tools	No Tools	With Tools	No Tools	With Tools	No Tools	With Tools	No Tools	With Tools
Text-Based Data				CHG | ORG	CHG | ORG								
Qwen3-4B	Text	75.00	56.40	67.20 | 67.80	49.30 | 41.60	86.70	65.10	0.26	0.22	8.10	2.00	0.66	0.00
Qwen3-4B [T]	Text	75.70	32.60	70.30 | 71.70	42.10 | 30.60	90.30	57.00	0.26	0.23	6.90	5.50	0.77	0.01
GPT-OSS-20B [L]	Text	80.00	67.80	71.20 | 74.40	63.70 | 62.50	88.70	**79.60**	0.24	0.21	9.70	7.40	0.75	0.34
GPT-OSS-20B [M]	Text	79.94	68.90	70.66 | 74.95	**64.10 | 63.20**	88.70	77.20	0.24	0.22	10.00	5.50	0.73	**0.42**
GPT-OSS-20B [H]	Text	**81.20**	69.60	71.90 | 76.10	63.70 | 62.80	89.20	75.20	0.25	0.23	8.50	4.40	0.70	0.41
Qwen3-30B	Text	73.50	64.14	59.50 | 67.60	49.65 | 55.04	83.20	61.70	0.26	0.23	5.80	1.90	0.64	0.00
Qwen3-30B [T]	Text	78.82	67.50	**72.33** | 76.72	57.40 | 60.10	**91.60**	77.60	0.25	0.24	6.90	4.80	0.66	0.01
Nemotron3-Nano-30B-A3B	Text	75.62	64.90	69.63 | 72.03	58.80 | 57.10	86.20	73.80	0.26	0.24	9.40	5.70	0.82	0.01
GPT-4o	Text	80.00	71.03	66.90 | 75.70	58.74 | 63.14	86.30	75.00	**0.28**	0.23	6.10	5.10	0.84	0.02
GPT-4	Text	77.90	**73.80**	69.90 | **77.00**	64.00 | 61.80	86.30	79.20	0.27	**0.25**	**19.30**	**22.30**	**0.87**	0.02
Gemini 2.5 Pro	Text	86.60	77.50	78.80 | 82.60	72.80 | 73.00	93.80	85.80	0.30	0.29	12.00	35.40	0.79	0.03
Multimodal Data										RPJ_{ABS}			
Gemini 2.5 Pro	Text	67.74	64.22	66.65 | 64.65	56.11 | 61.26	85.18	80.59	0.21	0.20	3.66	3.13	0.75	0.51
Gemini 2.5 Pro	Multimodal	77.49	69.75	60.51 | 72.51	55.56 | 64.79	88.95	84.50	0.20	0.16	2.85	3.25	0.70	0.40

6 Results and Discussion

We evaluate model performance on PERSONALHOMEBENCH across reactive and proactive tasks under both *Sole Reasoning* and *Agentic* settings. We report findings along these axes: reasoning and counterfactual robustness, agentic planning and tool use, multimodal grounding, and the effect of personalization. We close with validation of our RPJ evaluation framework via human studies and self-reflection experiments.

6.1 Sole-Reasoning and Agentic Planning

Sole Reasoning setting isolates reasoning from tool-use and planning failures, providing an approximate upper bound on model capability. Even here, performance is strong but not saturated: most large models exceed 85% accuracy on MC and achieve plan validity scores between 0.64 and 0.87, indicating that reactive grounding and proactive planning still remain partially unsolved. Extended reasoning consistently improve reactive performance under *Sole Reasoning*, but these gains largely vanish or diminish in agentic settings (Sec. 6.3).

Counterfactual robustness. Counterfactual conditions expose a sharper failure mode. Under CHG, where semantically meaningful state changes invalidate prior conclusions, models incur accuracy drops of 8–14 percentage points relative to IG baselines (e.g., Qwen3-30B: 73.50% → 59.50%; GPT-4o: 80.00% → 66.90%), as models must recompute inferences rather than retrieve cached ones. The ORG condition produces milder drops of 4–6% points, suggesting models can often discount irrelevant perturbations but struggle to propagate valid changes. GPT-4 series are partial exceptions, exhibiting greater robustness to both settings.

Agentic Performance. Table 2 reveals the benchmark's central finding: transitioning from *Sole Reasoning* to the *Agentic* setting produces substantial performance drops across all task types, consistent with observations in prior agentic AI work (Xie et al., 2024a). The dominant failure pattern is *premature shortcutting*: models issue one or two context retrieval calls and generate an answer without acquiring sufficient information (Figure 4b). This gap is most acute in PG, where a semantically plausible plan can be produced after minimal retrieval, incentivizing models to bypass appliance discovery and API familiarization entirely. The result is plans that appear coherent but are non-executable, reflected in near-zero plan validity scores for nearly all models. The GPT-OSS family is a notable exception, maintaining comparatively stable validity (0.34–0.42) across both settings as it often gathers API information (Figure 4b).

6.2 Personalization

A core claim of PERSONALHOMEBENCH is that conditioning agent behavior on structured user models, comprising persona, long-term memories, and household state, introduces reasoning demands that generic benchmarks do not capture (Li et al., 2025). Table 3 quantifies this effect against an anonymous-identifier (generic) baseline.

Under *Sole Reasoning*, personalization consistently *degrades* performance across IG and CF for all models, with drops as large as 18 percentage points (e.g., GPT-OSS-20B [H] on IG), confirm-

ing that reasoning over individualized household details introduces substantial complexity beyond environment-driven task understanding. When PERSONALHOMETOOLS is available, however, these drops are dramatically reduced and in several cases reversed into modest gains. Structured, tool-mediated access to household information allows models to retrieve user-specific details on demand rather than relying on brittle inference over dense context, effectively transforming personalization from an additional cognitive burden into a tractable retrieval-and-reasoning problem. Additional results appear in Appendix Tables 8 and 9.

Table 3: **Effect of personalization**, measured as accuracy difference ($\Delta = Personalized – Generic$)

	IG		CF	
Model	No Tools	With Tools	No Tools	With Tools
GPT-OSS-20B [H]	↓18.34	↑3.83	↓22.54	↑1.42
GPT-4	↓21.88	↓0.53	↓26.33	↓0.64
Gemini 2.5 Pro	↓13.20	↑0.60	↓19.10	↑1.10

6.3 Tool usage error analysis

Figure 4a provides a detailed breakdown of failure modes across tasks and model scales. `EXECUTION_ERROR` is the dominant failure mode, especially for smaller and mid-sized models, indicating that while generated plans are often semantically plausible, they frequently fail under runtime constraints due to invalid member IDs, out-of-scope appliance queries, or skipped prerequisite steps such as retrieving the valid appliance list before issuing a status query.

Reactive tasks exhibit the highest concentration of tool errors because they are fundamentally deterministic: correct answers depend on complete and well-formed retrieval chains. In contrast, proactive tasks appear to produce fewer observable errors, but this is partly misleading since their open-ended nature allows models to bypass grounding and generate semantically coherent yet non-executable plans (near-zero plan validity scores in Table 2).

Model scaling reduces argument and tool-selection errors but does not eliminate execution failures. Reasoning-mode scaling introduces an additional failure pathway in the agentic setting: `UPSTREAM_FORMAT_ERROR` increases sharply for thinking models (T/L/M/H) as intermediate thought traces leak into schema-constrained JSON outputs, producing malformed tool calls that break downstream execution. As a result, the gains from explicit reasoning observed under *Sole Reasoning* often disappear or reverse in agentic settings, highlighting a fundamental tension between extended deliberation and structured tool-use compliance.

Additional discussions and results for the remaining tasks are provided in Appendix D.2 and D.4.

6.4 Multimodal Grounding

We evaluate multimodality along two axes: a text-only setting, where models generate and use video transcripts, and a full multimodal setting, where models infer directly over raw video.

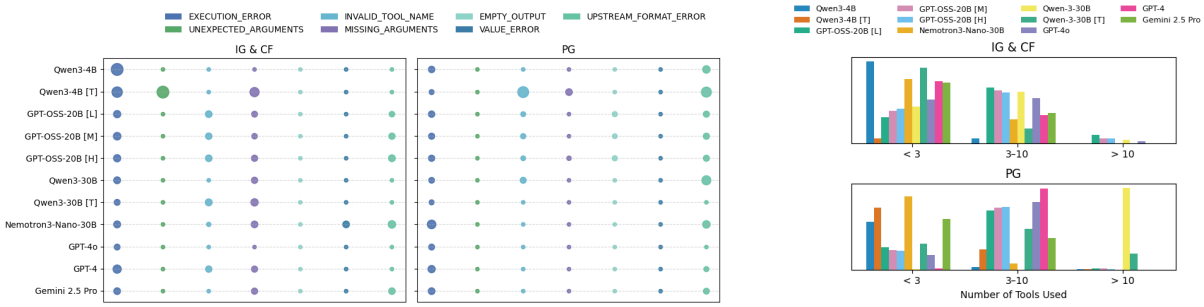
Multimodal inputs consistently benefit reactive tasks, particularly CF-ORG, where video footage provides a clear attentional anchor for targeted visual grounding; however, CF-CHG degrades, as textual perturbations (e.g., changing a shirt color) cannot be reflected in unedited video footage, introducing a cross-modal inconsistency that undermines grounding. Similarly, proactive tasks, show no such benefit and in several cases degrade (11% validity drop for PG). Without a guiding question to direct visual attention, models must accurately diffuse perceptual cues into personalized, forward-looking plans, a substantially harder problem. This manifests as contextually plausible but impersonally generic outputs (e.g., *Alert mom about the kid* rather than *Alert Sarah about Chloe*), undermining the household-specific grounding that distinguishes proactive assistance from generic advice. The larger, more heterogeneous multimodal tool stack further compounds this, increasing argument and hallucination error rates and limiting the net gains from perception alone.

6.5 Validation of RPJ Framework

We validate our Role-Playing LLM-as-a-Judge (RPJ) framework using both human evaluation and self-reflection.

Human Evaluation. Four human annotators benchmarked model-generated plan quality against overall and persona-level RPJ scores. We find an inter-annotator agreement rate exceeding 65%, with zero instances of absolute disagreement with RPJ (Appendix Fig. 12), indicating robust alignment with human judgment.

Self-Reflection Loop. To validate that RPJ provides an actionable optimization signal, we deploy it as a critic analogous to Reflexion (Shinn et al., 2023), where models iteratively refine their plans based on RPJ feedback. As shown in Table 4,



(a) (b)

Figure 4: Tool error type distribution (a) and tool call count distribution (b) across models for IG, CF and PG tasks. In (a), bubble size reflects error frequency across eight error types per model. In (b), tool call counts are grouped into fewer than 3, 3–10, and more than 10 calls, illustrating how actively each model retrieves context.

this technique improves personalization scores and plan validity across uni- and multi-modal settings.

Table 4: Reflection driven Agentic Plan Generation results using GEMINI 2.5 PRO.

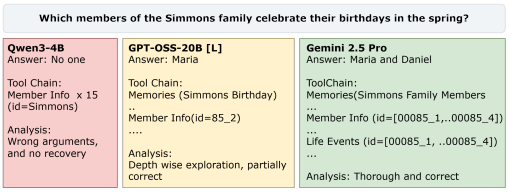
Setting	Unimodal		Multimodal	
Gemini	RPJ_{ABS}	Validity	RPJ_{ABS}	Validity
Base	3.380	0.030	3.250	0.405
w/ Reflection	3.520	0.428	4.240	0.496

Furthermore, cross-evaluator validation using GPT-4 as an independent judge yields highly stable model rankings. Mild intra-family bias is observed, where models assign marginally higher scores to outputs from the same model family, but this effect does not alter overall conclusions, confirming robustness to the choice of judge model.

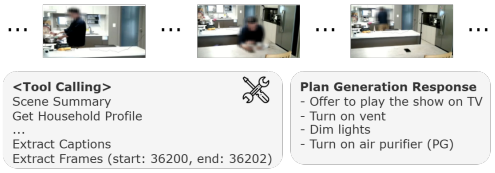
Together, these results establish RPJ as a reliable framework for evaluating and improving personalized planning quality. We provide qualitative examples of RPJ in Appendix I.2 subsection.

6.6 Qualitative Analysis.

Figure 5 illustrates representative success and failure cases across reactive and proactive settings. In the reactive setting (Figure 5a), weaker models issue redundant or misdirected tool calls and fail to recover from early errors, while stronger models issue targeted queries and assemble coherent evidence before answering. Figure 5b shows an example of video frames alongside the corresponding tool calls and generated plan with most models beginning with scene summary retrieval. Additional qualitative examples in Appendix I further corroborate the quantitative findings.



(a) Reactive Task (IG) Example



(b) Proactive Task (PG) Example

Figure 5: Qualitative examples of reactive question answering (a) and multimodal proactive planning (b).

Conclusion

We presented PERSONALHOMEBENCH, a benchmark for evaluating personalized agentic behavior in realistic smart home environments, alongside PERSONALHOMETOOLS, a domain-specific toolset for grounded interaction with household context, memory, perception, and device control. Extensive evaluation reveals that strong performance under full observability does not transfer to reliable agentic behavior: counterfactual reasoning, grounded tool use, and personalized plan generation remain consistently challenging. Gains from model scaling, explicit reasoning modes, and multimodal inputs are real but limited and task-dependent, underscoring that these axes alone are insufficient to close the gap. We hope PERSONALHOMEBENCH provides a systematic testbed for studying the interplay between reasoning, tool use, and personalization, and supports future progress toward robust personalized agentic systems.

Limitations and Future Work

Multimodal evaluation is currently restricted to GEMINI 2.5 PRO, as most available models either lack native video input support or exhibit insufficient reasoning and planning capabilities for long-horizon agentic evaluation (Singh et al., 2025; Tang et al., 2025; Xu et al., 2025). As multimodal models mature, future iterations of

PERSONALHOMEBENCH will incorporate a broader set of video-capable agents for more comprehensive comparison. Additionally, pairing rich textual context with a dedicated multimodal review stage represents a promising direction for recovering the benefits of both modalities in proactive tasks.

PERSONALHOMEBENCH also focuses on non-embodied smart home assistance; extending the benchmark to hybrid embodied–device settings is an important open direction. Finally, while RPJ provides scalable evaluation of subjective plan quality with strong human alignment, it remains an LLM-based proxy. Future work will expand direct human evaluation, explore preference learning from human feedback, and investigate training-time integration of RPJ-style supervision.

Plan Generation Design. Our current setting generates plans in a single pass and evaluates them step by step across each sub-program. This is appropriate given the task design: proactive plans are intended to be self-contained, granular, and executed upon user approval, making one-shot generation a natural fit.

A multi-turn setting, where each subsequent step is generated conditioned on the evaluation outcome of the previous one, can be considered as a natural extension. However, we found this setting poorly suited to proactive tasks in its current form. Due to the open-ended nature of proactive assistance, models exhibit persistent topic drift, continuing to generate steps until the maximum number of turns is exhausted. For example, given a trigger such as a garage door left open, a model may correctly alert the user and close the door, but then proceed to initiate a robot vacuum, order groceries, prepare a dinner ambiance plan, and extend into next-morning routines, with no natural stopping criterion. Introducing principled termination conditions and scope guardrails to contain multi-turn proactive plan generation is an important direction for future work.

References

Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, and 1 others. 2023. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*.

Sandhini Agarwal, Lama Ahmad, Jason Ai, Sam Altman, Andy Applebaum, Edwin Arbus, Rahul K Arora, Yu Bai, Bowen Baker, Haiming Bao, and 1 others. 2025. gpt-oss-120b & gpt-oss-20b model card. *arXiv preprint arXiv:2508.10925*.

Patryk Bartkowiak and Michal Podstawski. 2025. Edgewisepersona: A dataset for on-device user profiling from natural language interactions. *arXiv preprint arXiv:2505.11417*.

Aaron Blakeman, Aaron Grattafiori, Aarti Basant, Abhibha Gupta, Abhinav Khattar, Adi Renduchintala, Aditya Vavre, Akanksha Shukla, Akhiad Bercovich, Aleksander Ficek, and 1 others. 2025. Nemotron 3 nano: Open, efficient mixture-of-experts hybrid mamba-transformer model for agentic reasoning. *arXiv preprint arXiv:2512.20848*.

Rogerio Bonatti, Dan Zhao, Francesco Bonacci, Dillon Dupont, Sara Abdali, Yinheng Li, Yadong Lu, Justin Wagle, Kazuhito Koishida, Arthur Bucker, Lawrence Keunho Jang, and Zheng Hui. 2025. Windows agent arena: Evaluating multi-modal OS agents at scale. *Proceedings of the 42nd International Conference on Machine Learning*, 267:4874–4910.

Ma Chang, Junlei Zhang, Zhihao Zhu, Cheng Yang, Yujiu Yang, Yaohui Jin, Zhenzhong Lan, Lingpeng Kong, and Junxian He. 2024a. Agentboard: An analytical evaluation board of multi-turn llm agents. *Advances in neural information processing systems*, 37:74325–74362.

Matthew Chang, Gunjan Chhablani, Alexander Clegg, Mikael Dallaire Cote, Ruta Desai, Michal Hlavac, Vladimir Karashchuk, Jacob Krantz, Roozbeh Mottaghi, Priyam Parashar, and 1 others. 2024b. Partnr: A benchmark for planning and reasoning in embodied multi-agent tasks. *arXiv preprint arXiv:2411.00081*.

Gheorghe Comanici, Eric Bieber, Mike Schaekermann, Ice Pasupat, Noveen Sachdeva, Inderjit Dhillon, Marcel Blistein, Ori Ram, Dan Zhang, Evan Rosen, and 1 others. 2025. Gemini 2.5: Pushing the frontier with advanced reasoning, multimodality, long context, and next generation agentic capabilities. *arXiv preprint arXiv:2507.06261*.

Srijan Das, Rui Dai, Michal Koperski, Luca Minciullo, Lorenzo Garattoni, Francois Bremond, and Gianpiero Francesca. 2019. Toyota smarthome: Real-world activities of daily living. In *The IEEE International Conference on Computer Vision (ICCV)*.

Aaron Hurst, Adam Lerer, Adam P Goucher, Adam Perelman, Aditya Ramesh, Aidan Clark, AJ Ostrow, Akila Welihinda, Alan Hayes, Alec Radford, and 1

others. 2024. Gpt-4o system card. *arXiv preprint arXiv:2410.21276*.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph Gonzalez, Hao Zhang, and Ion Stoica. 2023. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the 29th symposium on operating systems principles*, pages 611–626.

Chengshu Li, Ruohan Zhang, Josiah Wong, Cem Gokmen, Sanjana Srivastava, Roberto Martín-Martín, Chen Wang, Gabrael Levine, Wensi Ai, Benjamin Martinez, Hang Yin, Michael Lingelbach, Minjune Hwang, Ayano Hiranaka, Sujay Garlanka, Arman Aydin, Sharon Lee, Jiankai Sun, Mona Anvari, and 16 others. 2024. Behavior-1k: A human-centered, embodied ai benchmark with 1,000 everyday activities and realistic simulation. *arXiv preprint arXiv:2403.09227*.

Silin Li, Yuhang Guo, Jiashu Yao, Zeming Liu, and Haifeng Wang. 2025. Homebench: Evaluating llms in smart homes with valid and invalid instructions across single and multiple devices. *arXiv preprint arXiv:2505.19628*.

Ya-Ting Lu, Shenzhi Yang, Cheng Qian, Gui-Fang Chen, Qinyu Luo, Yesai Wu, Huadong Wang, Xin Cong, Zhong Zhang, Yankai Lin, Weiwen Liu, Yasheng Wang, Zhiyuan Liu, Fangming Liu, and Maosong Sun. 2024. Proactive agent: Shifting llm agents from reactive responses to active assistance. *ArXiv*, abs/2410.12361.

Adyasha Maharana, Dong-Ho Lee, Sergey Tulyakov, Mohit Bansal, Francesco Barbieri, and Yuwei Fang. 2024. Evaluating very long-term conversational memory of LLM agents. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 13851–13870, Bangkok, Thailand. Association for Computational Linguistics.

Yev Meyer and Dane Corneil. 2025. Nemotron-Personas-USA: Synthetic personas aligned to real-world distributions.

Xavier Puig, Kevin Kyunghwan Ra, Marko Boben, Jiaman Li, Tingwu Wang, Sanja Fidler, and Antonio Torralba. 2018. Virtualhome: Simulating household activities via programs. *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 8494–8502.

Yujia Qin, Shihao Liang, Yining Ye, Kunlun Zhu, Lan Yan, Yaxi Lu, Yankai Lin, Xin Cong, Xiangru Tang, Bill Qian, Sihan Zhao, Lauren Hong, Runchu Tian, Ruobing Xie, Jie Zhou, Mark Gerstein, dahai li, Zhiyuan Liu, and Maosong Sun. 2024. ToolLLM: Facilitating large language models to master 16000+ real-world APIs. In *The Twelfth International Conference on Learning Representations*.

Alireza Salemi, Sheshera Mysore, Michael Bendersky, and Hamed Zamani. 2024. LaMP: When large language models meet personalization. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 7370–7392, Bangkok, Thailand. Association for Computational Linguistics.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. 2023. Reflexion: Language agents with verbal reinforcement learning. *Advances in Neural Information Processing Systems*, 36:8634–8652.

Mohit Shridhar, Jesse Thomason, Daniel Gordon, Yonatan Bisk, Winson Han, Roozbeh Mottaghi, Luke Zettlemoyer, and Dieter Fox. 2019. Alfred: A benchmark for interpreting grounded instructions for everyday tasks. *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 10737–10746.

Aaditya Singh, Adam Fry, Adam Perelman, Adam Tart, Adi Ganesh, Ahmed El-Kishky, Aidan McLaughlin, Aiden Low, AJ Ostrow, Akhila Ananthram, and 1 others. 2025. Openai gpt-5 system card. *arXiv preprint arXiv:2601.03267*.

Harmanpreet Singh, Nikhil Verma, Yixiao Wang, Manasa Bharadwaj, Homa Fashandi, Kevin Ferreira, and Chul Lee. 2024. Personal large language model agents: A case study on tailored travel planning. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing: Industry Track*, pages 486–514, Miami, Florida, US. Association for Computational Linguistics.

Changli Tang, Yixuan Li, Yudong Yang, Jimin Zhuang, Guangzhi Sun, Wei Li, Zejun Ma, and Chao Zhang. 2025. video-salmonn 2: Captioning-enhanced audio-visual large language models. *arXiv preprint arXiv:2506.15220*.

Noah Wang, Zy Peng, Haoran Que, Jiaheng Liu, Wangchunshu Zhou, Yuhan Wu, Hongcheng Guo, Ruitong Gan, Zehao Ni, Jian Yang, and 1 others. 2024. Rolellm: Benchmarking, eliciting, and enhancing role-playing abilities of large language models. In *Findings of the Association for Computational Linguistics: ACL 2024*, pages 14743–14777.

Zhiheng Xi, Yiwen Ding, Wenxiang Chen, Boyang Hong, Honglin Guo, Junzhe Wang, Xin Guo, Dingwen Yang, Chenyang Liao, Wei He, and 1 others. 2025. Agentgym: Evaluating and training large language model-based agents across diverse environments. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 27914–27961.

Jian Xie, Kai Zhang, Jiangjie Chen, Tinghui Zhu, Renze Lou, Yuandong Tian, Yanghua Xiao, and Yu Su. 2024a. Travelplanner: A benchmark for real-world planning with language agents. *arXiv preprint arXiv:2402.01622*.

Tianbao Xie, Danyang Zhang, Jixuan Chen, Xiaochuan Li, Siheng Zhao, Ruisheng Cao, Toh Jing Hua, Zhoujun Cheng, Dongchan Shin, Fangyu Lei, Yitao Liu, Yiheng Xu, Shuyan Zhou, Silvio Savarese, Caiming Xiong, Victor Zhong, and Tao Yu. 2024b. Osworld: Benchmarking multimodal agents for open-ended tasks in real computer environments. In *Advances in Neural Information Processing Systems*, volume 37, pages 52040–52094. Curran Associates, Inc.

Jin Xu, Zhifang Guo, Jinzheng He, Hangrui Hu, Ting He, Shuai Bai, Keqin Chen, Jialin Wang, Yang Fan, Kai Dang, and 1 others. 2025. Qwen2. 5-omni technical report. *arXiv preprint arXiv:2503.20215*.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, and 1 others. 2025a. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*.

Bufang Yang, Lilin Xu, Liekang Zeng, Kaiwei Liu, Siyang Jiang, Wenrui Lu, Hongkai Chen, Xiaofan Jiang, Guoliang Xing, and Zhenyu Yan. 2025b. Contextagent: Context-aware proactive LLM agents with open-world sensory perceptions. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*.

Rui Yang, Hanyang Chen, Junyu Zhang, Mark Zhao, Cheng Qian, Kangrui Wang, Qineng Wang, Teja Venkat Koripella, Marziyeh Movahedi, Manling Li, and 1 others. 2025c. Embodiedbench: Comprehensive benchmarking multi-modal large language models for vision-driven embodied agents. *arXiv preprint arXiv:2502.09560*.

Shunyu Yao, Howard Chen, John Yang, and Karthik Narasimhan. 2022. Webshop: Towards scalable real-world web interaction with grounded language agents. In *Advances in Neural Information Processing Systems*, volume 35, pages 20744–20757. Curran Associates, Inc.

Xuan Zhang, Yang Deng, Zifeng Ren, See-Kiong Ng, and Tat-Seng Chua. 2024. Ask-before-plan: Proactive language agents for real-world planning. *ArXiv*, abs/2406.12639.

Sebastian Zhao, Alan Zhu, Hussein Mozannar, David A. Sontag, Ameet Talwalkar, and Valerie Chen. 2025. Codinggenie: A proactive llm-powered programming assistant. *Proceedings of the 33rd ACM International Conference on the Foundations of Software Engineering*.

Mu Zhu. 2004. Recall, precision and average precision. *Department of Statistics and Actuarial Science, University of Waterloo, Waterloo*, 2(30):6.

A Outline

The appendix provides supplementary material that supports and extends the main paper, including additional analyses, experimental details, and resources omitted from the main text due to space constraints. Specifically, we include:

- Sec. B: Additional related work and extended comparisons.
- Sec. C: Further statistics and analysis of households in PersonalHomeBench.
- Sec. D: Extended experimental results and auxiliary evaluation metrics.
- Sec. E: Model descriptions and hyperparameter settings.
- Sec. F: Data schemas used in the benchmark.
- Sec. G: Prompts used for simulated and semi-simulated data generation.
- Sec. H: Instructions provided to human annotators for RPJ evaluation and for video collection.
- Sec. I: Additional qualitative examples.
- Sec. J: Detailed documentation of all tools in PersonalHomeTools.

B Additional Related Works

Links to the main draft sections

Introduction (1), Related Works (2), PERSONALHOMEBENCH (3), PERSONALHOMETOOLS (4), Experimental Setup (5), and Results (6).

Reactive Agentic Benchmarks. A large body of work has established benchmarks for evaluating agentic behavior in web-based interaction (Yao et al., 2022), UI navigation (Xie et al., 2024b), game-like environments, and simulated worlds (Chang et al., 2024a). These benchmarks have standardized evaluation of multi-step instruction following and reasoning, but typically emphasize reactive task completion in environments with limited long-term context and minimal personalization. More recent tool-centric benchmarks evaluate whether foundation models can select and invoke general-purpose APIs, providing insight into basic tool usage (Qin et al., 2024). However, the tools are largely domain-agnostic, and tasks rarely require sustained user modeling, household continuity, or proactive assistance.

Embodied Home Agentic Benchmarks. Embodied household benchmarks such as VirtualHome (Puig et al., 2018), ALFRED (Shridhar et al., 2019), BEHAVIOR (Li et al., 2024), and PARTNR (Chang et al., 2024b) evaluate agents in simulated home environments with navigation and atomic action execution. In contrast, PERSONALHOMEBENCH focuses on non-embodied, real-world home environments, using long-horizon videos captured in diverse households to reflect natural daily routines, health, and safety scenarios.

Proactive Agentic Benchmarks. Current proactive LLM agents cover specialized frameworks for clarification (Zhang et al., 2024), multi-agent prediction (Lu et al., 2024), and UI-monitored task assistance (Lu et al., 2024; Zhao et al., 2025). ContextAgentBench (Yang et al., 2025b) takes an important step toward evaluating proactive agents with contextual information and tool usage, but focuses primarily on ego-centric daily traces, relies on generic tools, and operates in language-only inference settings. In contrast, our benchmark targets shared household environments, uses domain-specific smart home tools, and evaluates both reactive and proactive behaviors grounded in persistent, personalized context with optional multimodal inputs.

Personalization in Interactive Systems. Personalization has been studied through persona consistency benchmarks, long-term memory datasets, and user-specific question answering tasks. These benchmarks primarily assess whether models can recall or remain consistent with user attributes in text-based interactions (Salemi et al., 2024; Maharana et al., 2024). Parallel work explores personalization for narrow

planning problems, such as personalized travel assistants with limited tool support (Singh et al., 2024). While valuable, these settings treat personalization largely as a property of dialogue or retrieval. PERSONALHOMEBENCH treats personalization as a central driver of interactive decision making, assessing how user preferences influence reasoning, tool usage, and long-horizon planning in dynamic environments, and how this information is leveraged when evaluating generated plans.

Home-Centric Benchmarks.

Unlike datasets such as Toyota Smarthome (Das et al., 2019), which are staged, demographically narrow, and not designed around smart-device interaction, our data spans diverse home layouts, age groups, relationships, and unprompted activities. Videos range from seconds to tens of minutes and may cover multiple activities (e.g., cooking, leisure, sleep), enabling evaluation of persistent context tracking and proactive assistance over extended horizons.

Recent language-model-based home assistant benchmarks, such as HomeBench (Li et al., 2025) and EdgeWisePersona (Bartkowiak and Podstawski, 2025), move closer to smart home scenarios but remain limited. HomeBench focuses on reactive validation and execution of smart home commands without modeling personalization, while EdgeWisePersona emphasizes profile reconstruction and lightweight proactive routine inference with a small set of devices and simple state transitions.

C Additional Household Statistics

Links to the main draft sections

Introduction (1), Related Works (2), PERSONALHOMEBENCH (3), PERSONALHOMETOOLS (4), Experimental Setup (5), and Results (6).

C.1 Text-based Data Statistics

Household Size. The households capture a structured variety of domestic environments, categorized into: single occupant, family and non-related roommates. The range of household scales from single-occupancy to large-family residences (Fig. 6a).

Appliances Owned. The dataset features nearly 40 distinct appliance types, including five core appliances present in all households: refrigerator, thermostat, microwave, speaker and TV. Other top most prevalent appliances are listed in (Fig. 6c).

Pets Distribution. Across 409 households, a total of 823 animals are represented in this dataset. The pet population is categorized into nine animal types: cat, dog, hamster, lizard, guinea pig, rabbit, bird, fish and turtle - spanning a wide range of common domestic mammals, reptiles and aquatic companions.

C.2 Multimodal Data Statistics

Table 5 presents the distribution of the 100 scenarios constructed for this study across three core domains: safety care, health care, and daily care. Health care scenarios constitute the largest portion, reflecting the prevalence of situations involving well-being, medication, and environmental health in household settings. Daily care scenarios capture routine activities such as returning home and pet care, while safety care scenarios focus on risk-sensitive situations and accident prevention. Scenarios are categorized based on consistent semantic cues in their titles, enabling clear domain delineation while preserving coverage of diverse, realistic household situations spanning safety, health, and everyday routines.

Table 6 summarizes the participant composition of the real home data used in our benchmark across health care, safety care, and daily care scenarios. The dataset captures substantial diversity in household roles, including adults, caregivers, children, and pets, spanning a wide range of age groups from young children to older adults in their 70s. Safety care scenarios are the most prevalent, reflecting common real-world situations that involve risk monitoring and intervention, particularly in households with both caregivers and children. Daily care scenarios further emphasize routine, multi-occupant household dynamics, while health care scenarios focus on age-sensitive contexts involving adults across multiple life stages. Overall, this distribution highlights the benchmark's emphasis on realistic household diversity and interaction patterns, providing a representative foundation for evaluating personalized and proactive

assistance in everyday home environments. All the data is ego-centric views, and collected through safe placement of cameras in subject's homes.

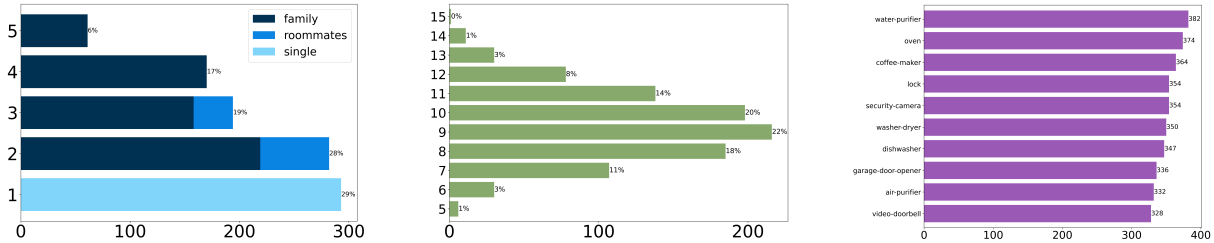
Instructions for Video Data Collection Participants. Participants were provided with scenario-specific guidance corresponding to the domains listed in Table 5 after household setup was completed, including camera placement, microphone activation, and sensor checks. For each scenario, participants received high-level instructions describing how the scene should unfold and the types of actions to perform, while allowing natural variation in execution. For scenarios involving safety or hazardous situations, participants were instructed to simulate the actions rather than perform them physically, and on-site staff ensured that all safety protocols were followed. Scenes with multiple participants were coordinated by coaching all subjects involved to maintain consistency in the intended narrative. To ensure usable recordings, scenes were re-shot when necessary to improve clarity or coverage. The source data were collected under a service contract with an external vendor and processed in compliance with applicable personal data protection regulations following a personal information impact assessment. Participants were compensated for their participation. Where applicable, their information is redacted to ensure PII protection as per their agreement. The frames used in our qualitative example have been blurred because of the same reason.

Table 5: Distribution of scenario domains and classification criteria used in the benchmark.

Domain	Video Count	Classification Basis
Safety Care	30	Scenario titles reference safety, risk, hazard, or accident prevention
Health Care	38	Scenario titles reference health, medication, ventilation, or well-being
Daily Care / Routine	32	Scenario titles reference daily routines, returning home, pet care, or going out
Total	**100**	—

Table 6: Participant demographics across scenarios in the collected home data.

Scenario	Participant Type	Age Group	Gender	Participant Count
Health Care	Adult	30s	Male	2
Health Care	Adult	60–70s	Male	3
Health Care	Adult	60–70s	Female	3
Health Care (Total)				**8**
Safety Care	Adult (Caregiver)	30s	Female	5
Safety Care	Adult (Caregiver)	40–50s	Female	3
Safety Care	Adult (Caregiver)	40–50s	Male	2
Safety Care	Adult	30s	Male	1
Safety Care	Child	Under 10	Male	3
Safety Care	Child	Under 10	Female	5
Safety Care	Child	5 and over	Female	1
Safety Care	Child	10 and over	Female	2
Safety Care (Total)				**22**
Daily Care	Adult	60s	Male	1
Daily Care	Adult	70s	Male	1
Daily Care	Adult	70s	Female	1
Daily Care	Adult	30s	Male	1
Daily Care	Adult (Caregiver)	30s	Female	2
Daily Care	Adult (Caregiver)	40–50s	Female	2
Daily Care	Child	Under 10	Female	3
Daily Care	Child	Under 10	Male	1
Daily Care	Child	5 and over	Female	1
Daily Care	Child	10 and over	Female	2
Daily Care (Total)				**15**
Pets	Animal	–	Cat	1



(a) Household Structure (b) #Appliances Owned (c) Common Appliances Owned

Figure 6: Additional statistics of generated household in PersonalHomeBench. (a) Distribution of household size and composition. (b) Distribution of appliances owned in the households. (c) Common appliances owned by the households.

D Additional Results and Metric Definitions

Links to the main draft sections

Introduction (1), Related Works (2), PERSONALHOMEBENCH (3), PERSONALHOMETOOLS (4), Experimental Setup (5), and Results (6).

D.1 RPJ Algorithm

Algorithm 1 Role-Playing Judge (RPJ)

Require: Household personas $\mathcal{H}$, situation description $\mathcal{S}$, models $\{\mathcal{M}_i\}_{i=1}^{N}$ with generated plans $\{\mathcal{P}_i\}_{i=1}^{N}$, evaluation mode $mode \in \{\text{ABS}, \text{REL}\}$, judge model $\mathcal{M}_\theta$

Ensure: Evaluation results R containing overall and per-persona outcomes

Initialize judge model $\mathcal{M}_\theta$ with RPJ prompt conditioned on $\mathcal{H}$, $\mathcal{S}$, and $mode$
if $mode$ *is* ABS **then**
  Scoring: All scores are integers in $\{1, \ldots, 5\}$, where 5 denotes the highest quality.
  for each plan $\mathcal{P}_i$ **do**
    Query $\mathcal{M}_\theta$ to assign an overall $score_o$ to $\mathcal{P}_i$
    for each household persona $h \in \mathcal{H}$ **do**
      Query $\mathcal{M}_\theta$ to assign a per-persona $score_p$ to $\mathcal{P}_i$
    end for
    Store overall and per-persona scores in R
  end for
else
  Query $\mathcal{M}_\theta$ to rank all models $\{\mathcal{M}_i\}_{i=1}^{N}$ by overall usefulness of their plans
  for each household persona $h \in \mathcal{H}$ **do**
    Query $\mathcal{M}_\theta$ to rank all models $\{\mathcal{M}_i\}_{i=1}^{N}$ from the perspective of h
  end for
  Store overall and per-persona rankings in R
end if
Return R

D.2 Tool Usage Failure Analysis

To clarify failure modes, we categorize errors into three types:

- **Reasoning Errors.** Failures where the model does not arrive at the correct answer despite sufficient context. Our *Sole-Reasoning* setting directly captures this class. (See Section 6 for the main results.)

- **Reasoning & Planning Errors.** Failures to construct effective multi-step plans, leading to incorrect outcomes. This corresponds to the *agentic setting* in our benchmark. (See Section 6 for the main results.)

- **PG Plan Errors.** We evaluate PG program validity holistically, capturing parameter mismatches, signature violations, type errors, and execution failures. Beyond static analysis, each program is executed step-by-step with environment resets, enabling detection of dynamic interaction errors. (See Section 6 for the main results.)

- **Tool Usage Errors.** The errors arise in the agentic setting, where models actively gather information and interact with tools to solve tasks. We analyze these errors and their corresponding patterns below.

Figure 7 presents a bubble plot visualization of error distributions across models (y-axis) and error types (x-axis), with bubble size proportional to the average frequency of each error. We analyze trends across four task settings: **IG, CF**, **MC**, **FR**, and **PG**, revealing consistent structural patterns in model failure modes as well as task-specific sensitivities.

Dominance of Execution Errors. Across all tasks, `EXECUTION_ERROR` emerges as the most prominent failure mode, particularly for smaller and mid-sized models (e.g., Qwen3-4B, Qwen3-30B, Nemotron3-Nano-30B). The consistently large bubbles indicate that models frequently fail at the stage of tool execution rather than earlier reasoning or planning. This suggests that even when models produce syntactically valid plans, they struggle to satisfy runtime constraints. Notably, even stronger models (e.g., GPT-4o, Gemini 2.5 Pro) do not eliminate this failure mode entirely, indicating that execution robustness remains an open challenge.

These include scenarios such as invoking tools with non-existent member IDs, querying appliances that are not associated with a given household, or bypassing necessary intermediate steps (e.g., directly calling an appliance status API without first retrieving the valid appliance list via a preceding tool). Such errors highlight failures in state tracking and procedural grounding rather than surface-level formatting or schema adherence.

Argument-Level Errors: Planning vs. Grounding Gap. Errors such as `MISSING_ARGUMENTS` and `UNEXPECTED_ARGUMENTS` form the second dominant cluster. These are especially pronounced in **IG, CF** and **MC** settings, where multiple tool calls are required to gather sufficient information. The prevalence of these errors indicates a systematic gap between high-level planning and low-level parameter grounding:

- `MISSING_ARGUMENTS` suggests incomplete decomposition or failure to propagate required variables across steps.

- `UNEXPECTED_ARGUMENTS` indicates over-generation or hallucinated parameters not supported by the tool schema.

These two error types often co-occur across models, reinforcing the hypothesis that argument specification errors arise from inconsistencies in intermediate representations rather than isolated local failures.

Tool Selection vs. Tool Usage. `INVALID_TOOL_NAME` errors are relatively infrequent compared to argument-related issues, with generally smaller bubbles across all tasks. This suggests that models are largely capable of selecting the correct tool from the available set. However, the contrast between low tool-selection errors and high execution/argument errors highlights a key bottleneck: *Models often know which tool to use, but not how to use it correctly.*

This distinction implies that improvements in tool grounding and schema alignment may yield larger gains than improvements in tool retrieval or selection.

Task-level error trends: Reactive vs. Proactive settings. Grouping tasks by their information retrieval demand reveals a structurally important asymmetry in how errors surface. In *reactive question-answering* tasks (IG, CF, MC), models must invoke the correct sequence of tools and ground their answers in retrieved environmental state, there is no plausible path to a correct answer without tool compliance. This obligation

is reflected in the bubble plots: IG, CF and MC consistently show the largest and most concentrated error bubbles, dominated primarily by `EXECUTION_ERROR` and `MISSING_ARGUMENTS`, precisely because any deviation from a valid, complete tool call is immediately penalized.

In contrast, *proactive recommendation* tasks (FR, PG) impose no such hard grounding constraint, a model can produce a syntactically coherent plan or feature recommendation using arbitrarily little retrieved context, and such outputs are not trivially incorrect in the way a wrong lookup answer is. This leniency is reflected in markedly smaller and more diffuse bubbles across FR and PG, but it is a *misleading* signal of competence: the reduction in observable tool errors does not imply better performance, but rather that models frequently bypass grounding altogether, generating narrowly scoped or impractical recommendations that satisfy surface-level fluency while ignoring environmental constraints that a thorough tool-use strategy would have surfaced. The result is an inverse relationship between error visibility and failure severity across the two task families, *reactive tasks expose failures loudly through execution breakdowns, while proactive tasks conceal them quietly through under-grounded generation*.

Formatting errors. `UPSTREAM_FORMAT_ERROR` and `EMPTY_OUTPUT` remain comparatively infrequent across all settings. However, we observe a non-trivial dependence on model scale and behavior. Smaller models are disproportionately affected by `MISSING_ARGUMENTS`, `UNEXPECTED_ARGUMENTS`, and `EXECUTION_ERROR`, often exhausting their interaction budget attempting to construct syntactically valid and semantically consistent tool calls. In contrast, mid-sized models (20–30B) exhibit a distinct failure mode: they more frequently produce malformed outputs that violate strict schema constraints, such as generating additional reasoning text outside the expected JSON structure for tool calls.

These formatting violations are further amplified in multi-turn settings, where failed outputs are fed back into subsequent context. This can lead models to incorrectly assume task completion, prematurely switch tools, or propagate earlier mistakes through compounding errors. Notably, such issues are far less prevalent in larger, high-reasoning models, which demonstrate stronger adherence to output schemas and more stable multi-step planning.

We hypothesize that this behavior in mid-sized models stems from a combination of partial reasoning capability and overconfidence: unlike smaller models that fail early and explicitly, these models attempt more complex reasoning but lack robustness in structured generation. While it is possible to mitigate a subset of these errors through carefully engineered parsers or post-processing pipelines, our evaluation intentionally reflects *out-of-the-box* performance, without specialized parsing or correction mechanisms, in order to more faithfully capture real-world deployment constraints.

Model Scaling Trends. A clear trend emerges with increasing model scale: larger models tend to reduce the frequency of argument-level and tool-selection errors, but still exhibit non-trivial execution failures. This suggests that while scaling improves reasoning and planning fidelity, it does not fully resolve issues in tool interaction and execution robustness. Consequently, future improvements may require better training signals or architectures specifically targeting tool-use reliability rather than relying solely on scale. Overall, models struggle in **multi-hop tool usage**, especially when tools are abstracted for prompt efficiency (e.g., API-based retrieval instead of inline context). Parameter-related failures remain dominant, while increasing the number of tools often leads to inefficient exploration, redundant calls, and higher error rates. Conversely, fewer but more expressive tools reduce both cognitive load and misuse.

We notice that Gemini 2.5 Pro follows similar tool usage error trends for multimodal setting as well.

Implications for tool design:

- Prefer **simpler interfaces** with fewer arguments and richer outputs to reduce call complexity and parameter errors.
- Avoid **ambiguous or similar function names** (e.g., `get_event_captions` vs. `get_captions`).
- Include **explicit error signals** in outputs to support implicit self-correction, which larger models utilize effectively.

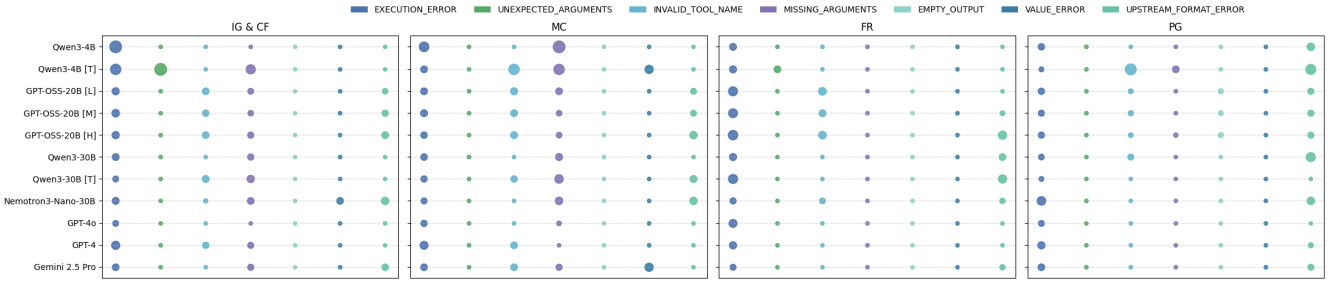


Figure 7: **Error distribution across models, tasks, and failure types.** Bubble plots show the average frequency of error types across four task settings (IG, CF, MC, FR, PG). Rows correspond to models and columns to error categories, with bubble size proportional to error frequency.

D.3 Definitions

For IG and CF, correctness is first assessed using string-level exact match (Eq. 1). If an exact match fails, we employ GEMINI 2.5 PRO as a semantic equivalence judge (Eq. 2) to determine whether the predicted answer is meaningfully consistent with the ground truth. This hybrid decision rule is formalized in Eq. 3.

For CF, performance is reported as the macro-average over counterfactual instances where the answer remains unchanged and those where the answer changes, as defined in Eq. 4.

For MC, accuracy is computed via exact match between the selected option (or its textual form) and the ground truth choice, as shown in Eq. 6. Since all valid answers are explicitly enumerated, semantic judging is not applied for this task.

For FR, we formulate the task as a ranking problem and report Mean Average Precision at rank k (MAP@k), which measures how highly relevant features are ranked within the recommendation list. MAP@k computation is given in Eq. 9.

We additionally report *Average Relevant Rank Sum (RRS)*, a ranking error metric that quantifies how far relevant items are from their ideal positions. For each instance, the absolute deviation between the predicted rank of each relevant item and its ideal rank (1–3) is summed, and the final score is obtained by averaging across instances (Eq. 12–13). Lower values indicate better ranking quality, with zero corresponding to perfect ordering. This metric provides complementary insight beyond top-k accuracy by capturing the magnitude of mis-ranking among relevant options.

For PG, we use two complementary metrics. First, subjective plan quality is assessed using the Role Playing Judge (RPJ) framework. Second, we measure *plan validity*, defined as the proportion of sub-program steps that are directly executable with correct tool or feature names and well-formed arguments, as formalized in Eq. 11.

Together, these metrics capture correctness for deterministic reactive tasks, ranking quality for proactive recommendation, and both perceived usefulness and practical executability for plan generation.

Table 7: Summary of evaluation metrics used for each task and their corresponding equation definitions.

Task	Metric	Equation
Information Grounding (IG)	Hybrid Exact/Semantic Accuracy	Eq. 3
Counterfactual Reasoning (CF)	Hybrid Exact/Semantic Accuracy	Eq. 3
Counterfactual Reasoning (CF)	Macro-Average Accuracy	Eq. 4
Multiple Choice (MC)	Exact-Match Accuracy	Eq. 6
Feature Recommendation (FR)	MAP@k	Eq. 9
Feature Recommendation (FR)	Average Relevant Rank Sum	Eq. 13
Plan Generation (PG)	RPJ Score	Algorithm 1
Plan Generation (PG)	Plan Validity	Eq. 11

$$\mathrm{EM}(\hat{y}_i, y_i) \;=\; \mathbb{I}[\mathrm{norm}(\hat{y}_i) = \mathrm{norm}(y_i)]\,, \tag{1}$$

$$\mathrm{J}(\hat{y}_i, y_i) \;=\; \mathrm{J}_{\textsc{Gemini 2.5 Pro}}(\hat{y}_i, y_i) \in \{0, 1\}, \tag{2}$$

$$c_i^{(\mathrm{IG/CF})} \;=\; \max\big(\mathrm{EM}(\hat{y}_i, y_i), \mathrm{J}(\hat{y}_i, y_i)\big), \qquad \mathrm{ACC}_{\mathrm{IG/CF}} \;=\; \frac{1}{N}\sum_{i=1}^{N} c_i^{(\mathrm{IG/CF})}. \tag{3}$$

$$\mathrm{ACC}_{\mathrm{CF}}^{\mathrm{MACRO}} \;=\; \frac{1}{2}\Big(\mathrm{ACC}_{\mathrm{CF}_{\mathrm{CHANGED}}} \;+\; \mathrm{ACC}_{\mathrm{CF}_{\mathrm{UNCHANGED}}}\Big), \tag{4}$$

$$\mathrm{ACC}_{\mathrm{CF}_s} \;=\; \frac{1}{|S_s|}\sum_{i \in S_s} c_i^{(\mathrm{CF})}, \qquad s \in \{\textsc{changed}, \textsc{unchanged}\}. \tag{5}$$

$$c_i^{(\mathrm{MC})} \;=\; \mathbb{I}[\mathrm{norm}(\hat{z}_i) = \mathrm{norm}(z_i)]\,, \qquad \mathrm{ACC}_{\mathrm{MC}} \;=\; \frac{1}{N}\sum_{i=1}^{N} c_i^{(\mathrm{MC})}. \tag{6}$$

$$P_i(r) \;=\; \frac{1}{r}\sum_{t=1}^{r} \mathrm{rel}_i(t), \tag{7}$$

$$\mathrm{AP@}k_i \;=\; \frac{1}{\min(k, R_i)}\sum_{r=1}^{k} P_i(r)\,\mathrm{rel}_i(r), \tag{8}$$

$$\mathrm{MAP@}k \;=\; \frac{1}{N}\sum_{i=1}^{N} \mathrm{AP@}k_i, \tag{9}$$

$$v_{i,j} \;=\; \mathbb{I}[\text{step } (i,j) \text{ uses a valid tool/feature name and well-formed arguments}]\,, \tag{10}$$

$$\mathrm{Validity}_i \;=\; \frac{1}{J_i}\sum_{j=1}^{J_i} v_{i,j}, \qquad \mathrm{Validity} \;=\; \frac{1}{N}\sum_{i=1}^{N} \mathrm{Validity}_i. \tag{11}$$

Per-instance Relevant Rank Deviation. Let N denote the number of evaluation instances and $K = 3$ the number of relevant items per instance. For instance i, let $r_{i,j}$ denote the predicted rank of the j-th relevant item, where the ideal ranks are $j \in \{1, 2, 3\}$.

$$D_i = \sum_{j=1}^{K} |r_{i,j} - j| \tag{12}$$

$$\mathrm{RRS} = \frac{1}{N}\sum_{i=1}^{N} D_i \tag{13}$$

Links to the main draft sections

Introduction (1), Related Works (2), PERSONALHOMEBENCH (3), PERSONALHOMETOOLS (4), Experimental Setup (5), and Results (6).

D.4 Reactive Tasks

COUNTERFACTUAL REASONING Figure 8 compares CF performance under *Sole Reasoning* (right) and agentic tool usage (left). Without tools, most models achieve relatively high accuracy on original questions and maintain comparable performance on both answer-changed and answer-unchanged counterfactuals, indicating that many models can internally simulate hypothetical perturbations when full context is directly provided. Performance is consistently strongest for large proprietary models, while medium and small models exhibit modestly lower but stable behavior across counterfactual types. In contrast, introducing tools produces a uniform degradation across models and counterfactual states. The drop is not confined to cases where the answer changes, but also affects answer-unchanged queries, suggesting that failures arise primarily from imperfect retrieval and tool interaction rather than counterfactual reasoning itself. Several models exhibit larger degradation on the unchanged condition, indicating that even preserving known facts becomes fragile once information must be re-acquired through tools.

Overall, the comparison highlights a clear separation between reasoning competence and interaction competence: models retain non-trivial counterfactual reasoning ability in isolation, but struggle to express it reliably in agentic settings where correct tool use is required. This reinforces that effective information acquisition and state grounding, rather than hypothetical reasoning alone, is the dominant bottleneck for counterfactual performance in personalized home environments.

Impact of Personalization and PersonalHomeTools. Tables 8 and 9 show that personalization has markedly different effects under *Sole Reasoning* versus agentic evaluation. When complete household context is directly available (Table 9), personalization generally leads to substantial accuracy gains across IG, CF, and MC, indicating that models can effectively exploit user-specific information when retrieval is trivial. In contrast, under agentic evaluation (Table 8), personalization yields smaller and more mixed changes, with modest improvements in some cases and slight regressions in others. Importantly, the relative stability of performance in the agentic setting reflects the personalization-centric design of PERSONALHOMETOOLS, which enables structured retrieval of user-specific context and mitigates the difficulty of reasoning directly over raw personalized information.

Effect of Task Difficulty on Reactive Task Performance. Figure 9 illustrates consistent and interpretable scaling behavior across task difficulty for all three reactive tasks. Accuracy decreases monotonically from easy to medium to hard settings, confirming that the difficulty stratification meaningfully captures increasing reasoning complexity. IG and MC remain comparatively more stable than CF across all difficulty levels, while CF exhibits the steepest degradation, reinforcing its role as the most challenging reactive task. Larger and proprietary models generally achieve higher accuracy across difficulties, but no model is immune to performance drops on hard instances, indicating that complex household-level reasoning remains challenging even at scale. Marker sizes further show that harder instances tend to require more interaction turns, suggesting that models attempt deeper tool-based exploration or longer reasoning chains without reliably closing the performance gap. Overall, the figure highlights that increasing difficulty exposes systematic weaknesses in multi-hop and counterfactual reasoning, and that additional interaction alone is insufficient to overcome these challenges.

Table 8: Effect of personalization under agentic evaluation, reporting accuracy for Generic and Personalized settings on IG, CF, and MC

	IG (ACC %)		**CF (ACC %)**		**MC (ACC %)**	
Model	Generic	Personalized	Generic	Personalized	Generic	Personalized
GPT-OSS-20B [H]	65.77	69.60	61.83	63.25	71.00	75.20
GPT-4	74.33	73.80	63.54	62.90	77.40	79.20
Gemini 2.5 Pro	76.90	77.50	71.80	72.90	87.90	85.80

Tool Usage Patterns. Figure 10 reveals a consistent pattern of under-querying, particularly in proactive tasks. For reactive tasks, most models concentrate in the fewer than 5 tool call range, with only a small fraction issuing 5–10 calls and very few exceeding 10, suggesting that even straightforward retrieval

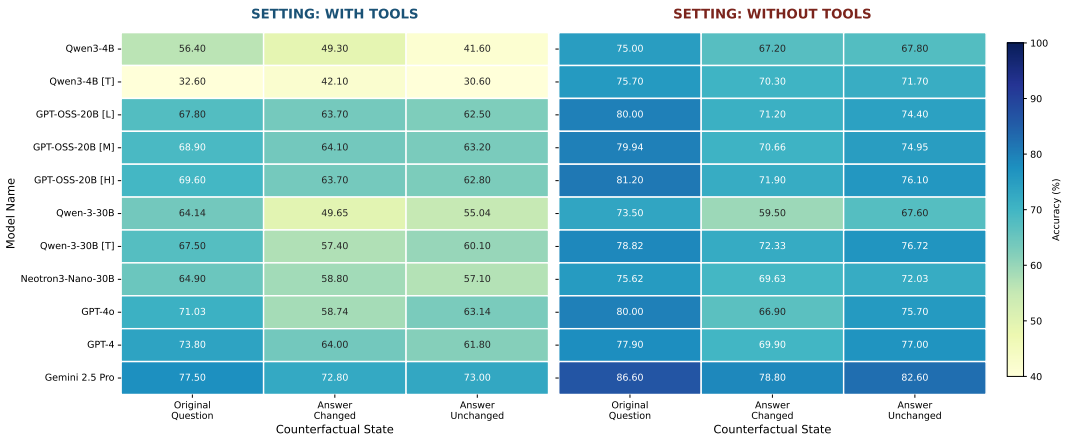


Figure 8: Counterfactual Reasoning (CF) performance across models under settings with and without tool access. The heatmap reports accuracy for original questions, counterfactual queries where the correct answer changes, and counterfactual queries where the answer remains unchanged.

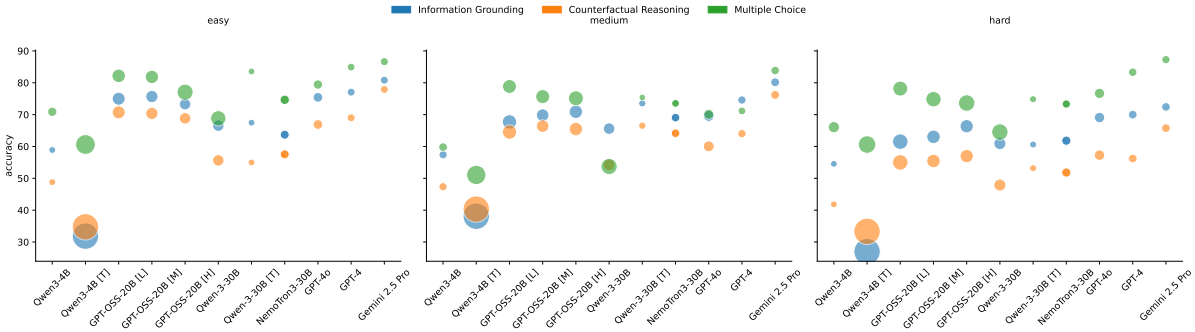


Figure 9: Accuracy across models for three reactive task categories, Information Grounding (IG), Counterfactual Reasoning (CF), and Multiple-Choice (MCQ), evaluated at easy, medium, and hard difficulty levels. Each point corresponds to a model–task pair; marker size reflects the average number of turns taken to complete the task.

chains are often truncated prematurely. The proactive setting is more pronounced: the majority of models issue fewer than 5 tool calls before generating a plan, consistent with the premature shortcutting behavior described above. GEMINI 2.5 PRO and GPT-OSS-20B are notable outliers, exhibiting a substantially heavier tail in tool call counts, reflecting more thorough context acquisition prior to planning and correlating with their comparatively higher plan validity on PG. Models with outlier spikes in the more than 10 call range, particularly within the QWEN family, frequently correspond to task non-completion within the 15-turn limit, driven by poor JSON formatting compliance and compounding tool usage errors that prevent forward progress. Taken together, the distributions highlight that most models under-invest in context acquisition, and that greater tool usage does not uniformly correspond to more effective grounding.

Links to the main draft sections

Introduction (1), Related Works (2), PERSONALHOMEBENCH (3), PERSONALHOMETOOLS (4), Experimental Setup (5), and Results (6).

Table 9: Effect of personalization under *Sole Reasoning*, reporting accuracy for Generic and Personalized settings on IG, CF, and MC.

	IG (ACC %)		CF (ACC %)		MC (ACC %)	
Model	Generic	Personalized	Generic	Personalized	Generic	Personalized
GPT-OSS-20B [H]	99.54	81.20	99.54	77.00	88.20	89.20
GPT-4	99.78	77.90	99.78	73.45	83.90	86.30
Gemini 2.5 Pro	99.80	86.60	99.80	80.70	94.10	80.70

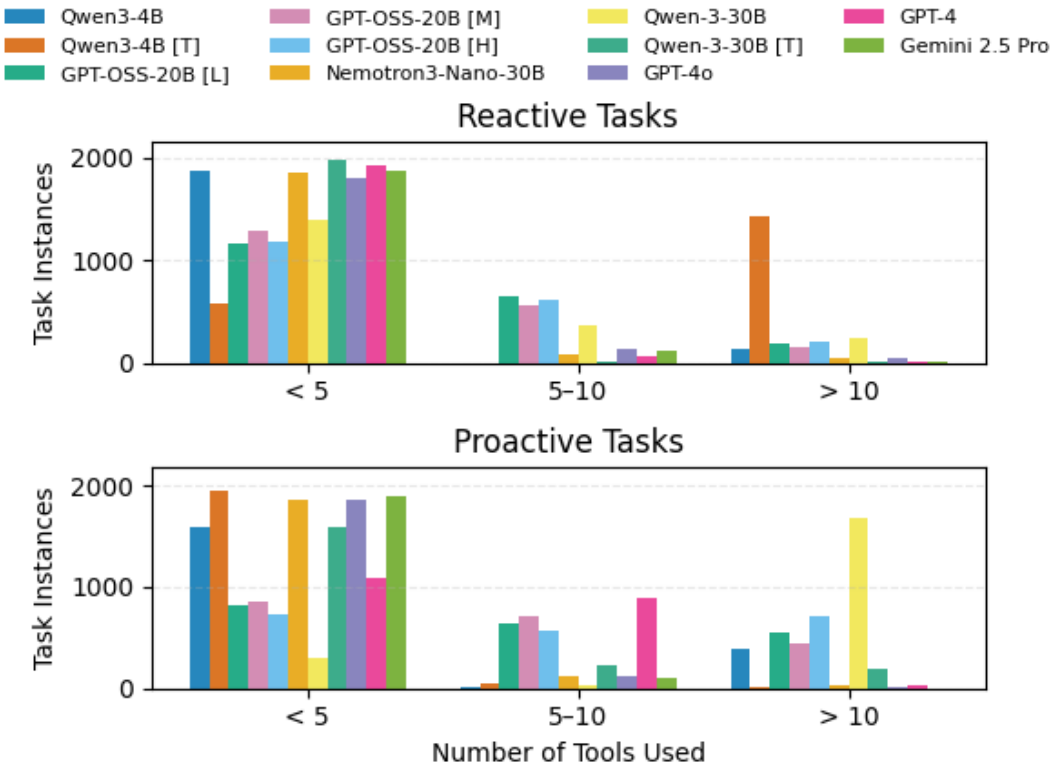


Figure 10: Tool usage distributions across models for reactive (top) and proactive (bottom) tasks, grouped by low (< 5), medium (5–10), and high (> 10) tool calls.

D.5 Proactive Tasks

Feature Recommendation. Figure 11 shows consistent improvements in MAP@k as k increases across all models, with performance gains tapering off beyond moderate values of k. Across the board, agentic settings with tool access underperform their Sole Reasoning counterparts, reinforcing that effective interaction with tools remains a key challenge even for strong foundation models. Larger and more capable models achieve higher absolute MAP@k, but the relative gap between with-tools and without-tools settings persists, indicating that scale alone does not resolve interaction-level errors. Notably, Gemini 2.5 Pro and GPT-4 class models define the upper envelope of performance, yet still exhibit nontrivial degradation under tool use, highlighting substantial headroom for improving agentic reasoning and decision-making in structured environments.

Table 10 reports Average Relevant Rank Sum (RRS) for Feature Recommendation under text-based evaluation. Under *Sole Reasoning*, frontier models achieve the lowest RRS values, with Gemini 2.5 Pro and GPT-4o performing best, indicating that relevant features are placed closer to their ideal ranks. Medium-scale open models exhibit moderately higher RRS, while smaller models show the largest

deviations, reflecting noisier ranking behavior. When tools are introduced, RRS increases substantially for nearly all models, demonstrating that tool-mediated interaction degrades fine-grained ranking quality even when top-1 accuracy may remain reasonable. Notably, the relative ordering of models remains broadly consistent across settings, suggesting that the degradation primarily stems from interaction complexity rather than a change in underlying ranking capability. Overall, these results reinforce that precise preference ordering remains challenging in agentic contexts, and that effective use of tools is critical for maintaining ranking fidelity.

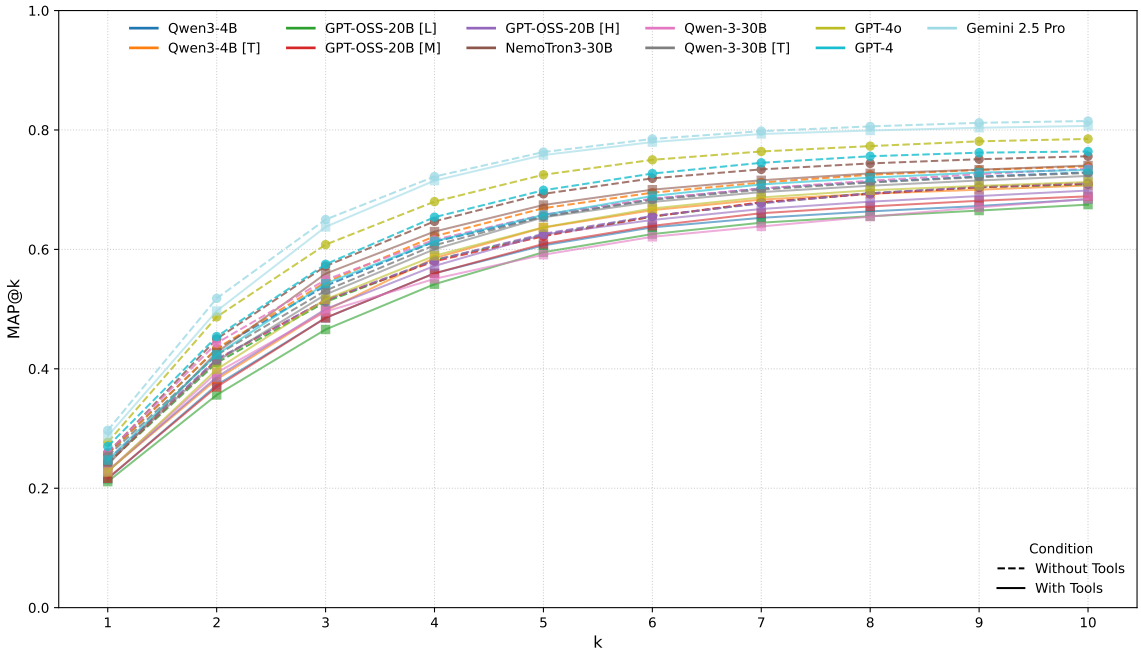


Figure 11: Mean Average Precision at rank k ($MAP@k$) across models evaluated with and without tool access. Solid lines denote performance with tools, while dashed lines correspond to settings without tools. Results are reported for varying values of k, illustrating ranking performance trends across evaluation conditions.

Table 10: Relevant Rank Sum for the FEATURE RECOMMENDATION task on text-based data. Lower values indicate better ranking quality.

Model	No Tools	With Tools
Qwen3-4B	3.4730	10.7620
Qwen3-4B [T]	3.3740	10.4270
GPT-OSS-20B [L]	3.5740	17.3260
GPT-OSS-20B [M]	3.5850	15.0270
GPT-OSS-20B [H]	3.5930	19.5160
Qwen3-30B	3.4910	21.8080
Qwen3-30B [T]	3.4500	10.3440
Neotron3-Nano-30B	3.2580	9.0988
GPT-4o	3.1340	10.5030
GPT-4	3.2100	10.2050
Gemini 2.5 Pro	**2.9780**	**9.0090**

Table 11: Feature recommendation and plan generation validity across models.

Model	**Feature Reco. (MAP@1)**		**Plan Gen. (Validity)**	
	No Tools	With Tools	No Tools	With Tools
Qwen3-4B	0.258±0.139	0.216±0.159	0.664±0.341	0.001±0.015
Qwen3-4B [T]	0.257±0.140	0.229±0.154	0.773±0.324	0.006±0.072
GPT-OSS-20B [L]	0.242±0.148	0.211±0.160	0.753±0.404	0.343±0.464
GPT-OSS-20B [M]	0.242±0.149	0.217±0.159	0.725±0.412	0.415±0.481
GPT-OSS-20B [H]	0.247±0.146	0.229±0.155	0.699±0.428	0.414±0.483
Qwen3-30B	0.260±0.138	0.229±0.154	0.645±0.302	0.002±0.024
Qwen3-30B [T]	0.254±0.142	0.244±0.148	0.666±0.467	0.044±0.205
NemoTron3-30B	0.259±0.138	0.239±0.149	0.825±0.320	0.010±0.090
GPT-4o	0.277±0.125	0.227±0.155	0.836±0.215	0.023±0.117
GPT-4	0.270±0.131	0.247±0.146	0.869±0.208	0.022±0.123
Gemini 2.5 Pro	0.297±0.104	0.287±0.115	0.795±0.238	0.030±0.157
Gemini 2.5 Pro (Multimodal)	0.196±0.164	0.163±0.166	0.697±0.297	0.405±0.423

E Models and Hyper-Parameters

Open models are hosted via vLLM (Kwon et al., 2023) on a multi-GPU node (4x GPUs, 96GB VRAM) utilizing a fixed tensor_parallel_size of 4. To ensure system stability and accommodate the overhead of long-context reasoning, the GPU memory utilization is lowered to 0.85 to prevent out-of-memory errors during the KV cache allocation.

For models operating in explicit thinking or reasoning modes, the maximum reasoning token budgets are explicitly constrained based on model scale. It is capped for QWEN3-4B [T] and QWEN3-30B [T], while GPT-OSS-20B and NEMOTRON3-NANO-30B-A3B utilize the default configuration. The QWEN3 reasoning models utilize DEEPSEEK-R1 as the reasoning parser and utilize its own Nano v3 parser within the vLLM serving environment.

Table 12: Reasoning Token Configuration by Model

Model Name	Reasoning Token	GPU Mem Utilization
Qwen3-4B	262,144	0.90
Qwen3-4B [T]	78,720	0.90
GPT-OSS-20B [L,M,H]	131,072	0.90
Qwen3-30B	32,768	0.85
Qwen3-30B [T]	32,768	0.85
Nemotron3-Nano-30B-A3B	131,072	0.85

Links to the main draft sections

Introduction (1), Related Works (2), PERSONALHOMEBENCH (3), PERSONALHOMETOOLS (4), Experimental Setup (5), and Results (6).

F Schemas

Household Schema.

Household

```
household_id: String
household_size: Integer
occupancy_type: String
layout: String
pets: [{
      name: String
      animal: String
      age: Integer
}]
members: [{
      member_id: String
      name: String
      role: String
      age: Integer
      gender: String
      occupation: String
      persona: String
      hobbies: [String]
      lifestyle: String
      preference: String
```

```
        major_events: [{
            name: String
            animal: String
            age: Integer
        }]
    }]
    appliances: [{
        appliance: ApplianceObject
        location: String
    }]
    sensors: [{
        sensor: SensorObject
        location: String
    }]
    memories: [{
        memory: String
        subjects: [String]
        additional_information: String
        time: String
    }]
    context: {
        events: [{
                description: String
                sensors: [{
                    sensor: SensorObject
                    location: String
                }]
                appliances: [{
                    appliance: ApplianceObject
                    location: String
                }]
        }]
        weather: {
                outdoor: {
                    rain: String
                    snow: String
                    sunny: String
                    temperature: String
                    wind: String
                }
                indoor: {
                    dust: String
                    humidity: String
                    odor: String
                    temperature: String
                }
        }
    }
```

Video Transcript Schema.

Video Transcript

```
datapoint_id: String
video_path: String
video_description: {
  timebase: {
      timestamp_format: DateTime
      start_offset_seconds: Integer
  }
  scene: {
      summary: String
      environment_type: String
      location_label: String
      lighting: {
          overall: String
          sources_visible: [String]
      }
      camera: {
          motion: String
          viewpoint: String
          notes: String
      }
      layout: {
          zones: [{
              zone_id: String
              label: String
              relative_position: String
          }]
          landmarks: [{
              landmark_id: String
              type: String
              description: String
              zone_id: String
          }]
      }
  }
  entities: {
      members: [{
          member_id: String
          appearance: String
          initial_location: String
      }]
      pets: [{
          pet_id: String
          species: String
          initial_location: String
          description: String
      }]
      appliances: [{
          appliance_id: String
          type: String
```

```
            location: String
            initial_state: {}
            description: String
        }]
        objects: [{
            object_id: String
            label: String
            category: String
            initial_location: String
            initial_state: String
            description: String
        }]
        events: [{
            event_id: String
            ts_start: String
            ts_end: String
            event_type: String
            actors: [String]
            targets: [String]
            description: String
            audio: String
        }]
        entity_state_timelines: {
            members: [{
                member_id: String
                states: [{
                    ts: String
                    state: String
                    trigger_event_ids: [String]
                }]
            }]
            pets: [{
                pet_id: String
                states: [{
                    ts: String
                    state: String
                    trigger_event_ids: [String]
                }]
            }]
            appliances: [{
                appliance_id: String
                states: [{
                    ts: String
                    state: String
                    trigger_event_ids: [String]
                }]
            }]
            objects: [{
                object_id: String
                states: [{
```

```
                    ts: String
                    state: String
                    trigger_event_ids: [String]
                }]
            }]
        }
    }
    captions: {
        global: String
        segments: [{
            ts_start: String
            ts_end: String
            caption: String
        }]
        events: [{
            event_id: String
            caption: String
        }]
    }
    quality: {
        notes: String
        consistency_checks: {
            entity_ids_consistent: Boolean
            all_state_changes_timestamped: Boolean
            no_unverifiable_inferences: Boolean
        }
    }
}
```

1203

Links to the main draft sections

Introduction (1), Related Works (2), PERSONALHOMEBENCH (3), PERSONALHOMETOOLS (4), Experimental Setup (5), and Results (6).

G Prompts

G.1 Household Member Persona Generation

Household Member Persona Generation Prompt Template

You are an expert persona generator. Your job is to create realistic, coherent individual personas based on household-level information and sample personas. Follow these rules precisely:

1. Use the Household Persona as the Anchor

- Use the household-level description to infer: demographics, living situation, socioeconomic context, lifestyle preferences, environment and context.
- All individual personas must align with this household information.

2. Respect the Occupancy Type

- If occupancy_type = roommate:
  - All individuals should be unrelated.
  - Ages should fall within a similar age range, unless the household persona describes otherwise.
  - Income levels, occupations, and lifestyles may differ, but should still plausibly co-exist in a shared-living situation.
  - Personalities should not be identical—make each person distinct yet compatible as cohabitants.
- Household size = 2
  - Could be a couple (romantic partners).
  - Or a parent + one child.
- Household size = 3
  - Could be a couple + one child.
  - Or a single parent + two children.
- Household size = 4 or 5
  - Typical family compositions include:
    - Two parents + children
    - Single parent + multiple children
    - Multi-generational families

3. Use the Samples as Style Guidance

- Use samples only as examples of tone, structure, and level of detail.
- Do not copy content.
- Maintain similar attributes.

4. Produce a Concise, Self-Contained Persona for Each Individual

- For each individual, provide:
  - member_id: must be {household_id}_{n} where n starts at 1
  - name: extract name from persona description
  - role: parent, child, roommate, partner, etc.
  - age (estimate if not explicitly stated)
  - gender (Male or Female)
  - occupation
  - persona: 1-2 sentences summarizing personality and character
  - hobby: a list of hobby and interest keywords
  - lifestyle: habits, routines, energy usage tendencies (e.g., heavy cooking, night owl, active, tech-savvy)
  - preference: any preferences relevant for simulation (e.g., temperature settings, appliance usage patterns, quiet mornings)
  - major_event: list of significant life events with date and description. This field can be left empty if no major events are known.

5. Output Format
Return the result as a list of personas, one entry per individual, labeled:

```
"members": [
  {
    "member_id": "{household_id}_1",
    "name": "",
    "role": "",
    "age": ,
    "gender": "",
    "occupation": "",
    "persona": "",
    "hobbies": ["", ""],
    "lifestyle": "",
    "preference": "",
    "major_event": [
      {
        "date": "",
        "description": ""
      }
    ]
  },
  {
    "member_id": "{household_id}_2",
    "name": "",
    "role": "",
    "age": ,
    "gender": "",
    "occupation": "",
    "persona": "",
    "hobbies": ["", ""],
```

```
    "lifestyle": "",
    "preference": "",
    "major_event": [
      {
        "date": "",
        "description": ""
      }
    ]
  }
  ]...
```

G.2 Task Generation

Reactive Tasks

Info Grounding and Multiple Choice Question Task Generation Prompt Template

Prompt: Dictionary-Driven Question–Answer Generation with Programs

Role
You are a data QA generation engine.
Your task is to generate accurate, verifiable question–answer pairs from a given dictionary with many keys and values (including nested objects, lists, IDs, names, attributes, states, and memories).

All outputs must be machine-gradable, program-grounded, and strictly derived from the input dictionary.

Input
You will be given:

- A dictionary with many keys and values
- IDs that may map to entities (for example, people, devices, rooms)
- Optional human-readable names associated with IDs
- Structured attributes, lists, timestamps, states, and/or memories

Required Output Structure
For each generated item, include all sections below.

- 1. Questions - For every item, produce two versions of the same question:
  - A. Question (ID-based)
    - Uses IDs or raw dictionary keys
    - Example: What is the favorite activity of member_id_001 and member_id_002?
  - B. Personalized Question (Name-based)
    - Replaces IDs with names when available
    - Semantically identical to the ID-based question
    - Example: What is the favorite activity of David and Susan?
  - Both questions must resolve to the exact same answer.
- 2. Answer
  - The answer must be exact and specific

  - It does not need to be a full sentence
  - Output the answer as a list of acceptable variants
  - All variants must be semantically equivalent
  - The answer must be directly grounded in the input dictionary
  - For every question–answer item, you must also generate a ‘multiple_choices‘ field.
    * Incorrect answers should be hard distractors that are plausible within the dictionary and semantically close to the correct answer, but wrong in a subtle and verifiable way.
    * Effective distractors can be created by swapping entities, using real values from the wrong context, time, or location, or by slightly modifying the correct value.
    * Obviously fake, generic, or ungrounded options are not allowed.
    * All incorrect answers must be mutually distinct, must not overlap with any accepted answer variants, and must match the same semantic type as the correct answer.

- 3. Difficulty Classification
  - Each question must be labeled with exactly one difficulty level:
    * simple: up to 3 reasoning steps
    * medium: 3 to 5 reasoning steps
    * hard: 5 to 10 reasoning steps
  - The generated set must include a mix of:
    * Simple questions (direct lookups, single entity)
    * Medium questions (comparison, aggregation, or multiple entities)
    * Hard questions (multi-hop reasoning across multiple keys, memories, or attributes)
  - Rules
    * All answers must be directly supported by the input dictionary
    * Do not hallucinate or infer missing information
    * Personalized and ID-based questions must map one-to-one
    * Answers must be machine-checkable
    * If multiple entities are mentioned, the answer must be correct for all of them
    * Programs must be sufficient to compute the answer and no more
  - Question Naturalness Constraints (Mandatory)
    * Questions must sound like something a human would realistically ask in everyday language.
      · The question should not reveal how the answer is computed.
      · The question should not mention joins, lookups, timelines, or cross-referencing.
    * Do NOT ask questions that require stating the same fact twice in different forms.
      · Avoid circular or self-referential questions (for example, asking what happened on the same date as an event that already describes that event).
    * If Personalization is not possible, then personalized question should be "None"

Output Format (Strict)
Return a list of JSON objects only, with no additional text.

```
[
  {{
    "difficulty": "simple | medium | hard",
    "question": "ID-based question",
    "personalized_question": "Name-based question",
    "answer_variants": ["answer_variant_1", "answer_variant_2"],
```

```
    "multiple_choices": {{
    "correct": "best_answer_variant",
    "incorrect": [
      "hard_distractor_1",
      "hard_distractor_2",
      "hard_distractor_3",
      "hard_distractor_4"
      ] # you MUST always predict 4 incorrect hard distractors
    }}
  }}
]
```

You MUST generate upto 3 examples per each difficulty level.

Here is the current household information:
{household_information}

Counterfactual Task Generation Prompt Template

You are a Household Knowledge Counterfactual Reasoning Assistant.

Your tasks:
1. Take a grounded QA pair.
2. Generate a counterfactual modification to the household/context/memory/appliance data.
3. Provide:
a) The counterfactual condition
b) The new derived answer under this counterfactual world
c) A short explanation of the reasoning shift

Input Format
You will be given a single QA item in JSON format with the following structure:

```
{{
  "difficulty": "easy | medium | hard",
  "question": "<generic question using IDs>",
  "personalized_question": "<question using names instead of IDs>",
  "answer_variants": [
    "<canonical answer>",
    "<alternate surface forms>"
  ],
  "multiple_choices": {{
    "correct": "<correct option>",
    "incorrect": [
      "<distractor1>",
      "<distractor2>",
      "<distractor3>",
      "<distractor4>"
    ]
  }}
}}
```

Notes:

- The question and personalized_question refer to the same underlying query.
- The answer is expected to be exact, not a full sentence.
- Multiple answer variants may be provided to allow robust accuracy checking.

Counterfactual Construction Rules

- Single-Factor Modification
  - Each counterfactual must modify only one primary factor.
  - The modification must be clearly identifiable and isolated.
  - Valid modification dimensions include:
    - Time (e.g., one month ago → six months ago)
    - Actor (switching the household member involved)
    - Activity (e.g., cooking → studying)
    - Appliance or sensor used
    - Location (e.g., kitchen → bedroom)
    - Sensor availability or malfunction
    - Routine, habit, or motivation changes
    - Environmental context (temperature, season, time of day)
- Explicit Causality
  - The counterfactual must include a clear cause–effect relationship.
  - If the answer changes, the modification must be the direct cause of the change.
  - If the answer does not change, explicitly state why the modification is irrelevant.
  - Avoid speculative, implicit, or circular reasoning.
- Logical and Physical Consistency
  - The counterfactual must remain internally consistent.
  - Do not violate:
    - Physical constraints (e.g., impossible appliance behavior)
    - Temporal constraints (e.g., effects preceding causes)
    - Household constraints (e.g., nonexistent members or appliances)
  - Do not introduce new entities unless strictly required by the change.
- Answer Precision and Validity
  - The new_answer must be:
    - One of the original answer_variants, OR
    - A logically updated answer with the same level of precision and granularity.
  - Answers must be exact values, not full sentences.
  - Ambiguous or underspecified answers are not allowed.
- Difficulty Alignment
  - The reasoning complexity of the counterfactual must align with the original difficulty:
  - Easy: Superficial or irrelevant change; answer remains unchanged.
  - Medium: Contextual change; answer may change.
  - Hard: Multi-hop reasoning involving memories, sensors, and inferred behavior.

ANSWERING RULES UNDER COUNTERFACTUAL:
- Recompute the answer as if the counterfactual were true.
- Stay strictly grounded—no new made-up facts beyond the change introduced.
- If the counterfactual doesn't meaningfully change the answer, say so explicitly.

OUTPUT FORMAT:
Return a JSON dictionary for each input QA pair:

```
{{
  "original_question_id": "...",
  "counterfactuals": [
    {{
      "change": "<description of the hypothetical change>",
      "new_answer": "<exact answer>",
      "answer_changed": true | false,
      "reasoning": "<concise causal explanation>",
      "<outer_key>": [
        {{
          "<field_1>": "<value>",
          "<field_2>": "<value>",
          "...": "..."
        }}
    }}
  ]
}}
```

Constraints:
- The list counterfactuals must contain between 1 and 3 items.
- answer_changed must accurately reflect whether the answer differs from the original.
- Reasoning must be minimal, explicit, and causal (no speculation).

Valid Counterfactual Examples (Illustrative Only)
- Temporal: "Assume the referenced memory occurred six months ago instead of one month ago."
- Behavioral: "Assume the household member stopped late-night cooking."
- Sensor-related: "Assume the microwave wattmeter was offline during that period."
- Substitution: "Assume a hot plate was used instead of a microwave."
- Irrelevant change: "Assume the indoor temperature was 30°C instead of 24°C."

Quality Checklist (All Must Pass)
Before producing the output, verify that:
- The counterfactual is realistic and plausible.
- The reasoning clearly explains causality.
- The answer is unambiguous and machine-checkable.
- No unnecessary entities or assumptions are introduced.

You MUST generate at least one counterfactual where answer should change, and one where answer should not change.
The JSON strictly follows the required schema.

Here is the current household information:
{household_information}

Here are all the question and answer pairs
{qa_pairs}

Feature Recommendation

Feature Ranking Task Generation Prompt Template

You are a Household Knowledge Ranking Generator. Your job is to produce context-grounded appliance feature rankings strictly based on the supplied household data.
A list of appliance features will be provided.
From this list, you must select exactly 10 features and produce a ranked ordering (1 → 10) justified by the household's needs, behaviors, device context, and long-term patterns.

Your task is to
1. Select exactly 10 features from the input list.
2. Assign a correct ranking from 1 (most relevant) to 10 (least relevant).
3. Rankings must reflect explicit evidence from:
- Household structure (members, ages, hobbies, roles, pets)
- Device information (appliance types, sensors, locations)
- Contextual data (current time, weather, schedules, states)
- Long-term memories (habits, major events, preferences)
4. Provide a short explanation for each feature, describing why it is placed in its specific ranking position.

RANKING REQUIREMENTS:
- The ranking must reflect actual priorities inferred from the household's behavior and needs.
- Reasoning should show:
- Observation (e.g., someone frequently forgets to turn off appliances)
- Inference (e.g., a room with high traffic might benefit from automation)
- Multi-step reasoning (e.g., pets + robot vacuum + user schedule patterns) - Do not invent appliance features not in the provided list.
- All choices must be grounded in specific, explicit fields of the household record.

API References

refrigerator
Internal cameras for remote viewing
This function will send a snapshot of the fridge/freezer contents to your APP. It will also give a list of detected contents.
Touchscreen family hub interface
Updates the widgets displayed on the family hub screen (e.g., calendar, notes, weather). Can also push a specific notification to the screen.
Ingredient management and expiration tracking
Gives a report of the ingredients and their meta data information such as quantity and expiration. If no specific ingredients are passed, the report will cover all ingredients. If create_shopping_list name is passed, it automatically generates a shopping list for ingredients that are low or about to expire
Recipe Recommendation
When given a natural language request about the recipe the user is interested for, a personalized recipe is recommended and displayed on the fridge screen. Diagnostic alerts (e.g., door ajar)
Sends an alert to the user for the issue.
Dual ice maker (cubes and crushed/spheres)

Toggles the ice maker modes or triggers a rapid ice production cycle.

---

oven
Remote preheat via app
The oven preheated to 350 F or the temperature specified by the user. This system also performs a diagnostic to preheating is safe. If it is unsafe, an alert is sent to the user APP about the issue.
Internal AI camera monitoring
Send a snapshot of the contents of the over to the user APP. If video mode is requested, live feed link is sent. If a specific query is passed, then the function returns an answer. When no enquiry is passed, it returns a live snapshot.
Cooking control compatibility
Change the mode of oven between convection, broil, or bake.
Integrated meat probe with digital readout
Returns the current internal temperature reading from the connected meat probe. Can set a target alert temperature.
...

---

speaker
Collect user response
Often used to get the last N response the user has provided to the speaker request. Returns a list of instructions
Next-Gen AI Assistant Integration
Offloads complex, conversational queries to a large language model (e.g., Gemini) for context-aware answers.
Spatial/Room-Sensing Audio
Uses microphones to analyze room acoustics and automatically adjust equalizer settings for spatial audio.
Matter/Thread/Wi-Fi 6E Hub Support
Scans for new devices over Matter or Thread protocols to add to the home mesh.
High-Fidelity Audio with Dolby Atmos support
Toggles high-definition codecs and Dolby Atmos rendering for supported media.
Generate notification for the user
Generates an audio notification for the user after completing a task

---

tv
Universal Content Search
Searches across all installed streaming apps and live TV to find a specific movie, show, or actor.
Hands-free Voice Control
Executes commands like 'Turn on', 'Volume up', 'Open Netflix', or 'Play Jazz' without a remote.
Smart Home Dashboard Overlay
Displays an interactive overlay showing status of other smart devices (e.g., doorbell camera, thermostat) without interrupting playback.
Ambient Art Mode
Switches the TV to a low-power mode displaying artwork, photos, or weather info when not in active use.
Multi-View / Split Screen
Splits the screen to show two different content sources simultaneously (e.g., Game + YouTube tutorial).
AI Picture/Sound Calibration
Uses the remote's microphone and TV's light sensors to optimize audio and video settings for the

room's current conditions.
Generate notification for the user to be displayed on TV
Generates a visual + audio notification for the user on TV. An example of this could be step by step recipes.
Next-Gen AI Assistant Integration
Offloads complex, conversational queries to a large language model (e.g., Gemini) for context-aware answers.

You MUST generate three HIGHLY_RELEVANT features, three RELEVANT features, and four LESS_RELEVANT features
Output Schema

```
[
  {{
    "appliance": "appliance name",
    "feature": "feature name",
    "rank": 1,
    "relevance": "HIGHLY_RELEVANT",
    "reasoning": "Explain why this feature is ranked #1 using explicit
    household data."
  }},
  {{
    "appliance": "appliance name",
    "feature": "feature name",
    "rank": 2,
    "relevance": "HIGHLY_RELEVANT",
    "reasoning": "Explain why this feature is ranked #2 using explicit
    household data."
  }},
  ...
  {{
    "appliance": "appliance name",
    "feature": "feature name",
    "rank": 5,
    "relevance": "RELEVANT",
    "reasoning": "Explain why this feature is ranked #3 using explicit
    household data."
  }},
  ...
  {{
    "appliance": "appliance name",
    "feature": "feature name",
    "rank": 10,
    "relevance": "LESS_RELEVANT",
    "reasoning": "Explain why this feature is ranked #10 using explicit
    household data."
  }}
]
```

All feature in rs in the ranking and MCQ should be grounded in the API References. Do not make up any features that.

1222

Here is the household information:
{household_information}

Plan Generation

Plan Generation Task Generation Prompt Template

SYSTEM ROLE
You are a smart home orchestration engine acting.
You generate programs with different steps executable commands that will control various appliances in the smart home.
Execution is **single-threaded**, but divided into **program segments** identified by a `program_id`.
A `program_id` represents one coherent intent (routine, safety action, entertainment flow, anomaly handling).

EXECUTION MODEL
• Only ONE program is active at a time
• A new `program_id` is created ONLY when intent changes
• All steps under the same intent MUST reuse the same `program_id`
• Some steps may include MULTIPLE actions **if and only if** they are compatible
Examples:
close garage + start music
show visual notification + start watching TV

OUTPUT FORMAT (STRICT)
Return a LIST of instruction objects.
Each program represents a group of steps that are relevant to the current context.

```
[
   {{
    "program_id": "string",
    "steps": [
      {{
        "step_id": "string",
        "appliance": "<appliance | toolbox>",
        "feature": "<feature>",
        "category": [] # one or multiple of
        ACCIDENT_RESPONSE, ROUTINE, ANOMALY, WELL_BEING
        "args": {{ ... }},
       "rationale": "Why is this feature relevant recommendation for the user"
      }}
    ]
    }}
  }}
]
```

Rules:
• All returned objects MUST share the SAME program_id
• Return MULTIPLE objects ONLY if actions can safely occur together
• NO text outside the JSON list

PROGRAM ID RULES

• Use descriptive, stable IDs (e.g. "garage_secure_exit", "evening_relax_music", "child_safety_alert")
• Change program_id ONLY when:
- switching between categories
- switching from one user intent to another
• NEVER reuse a program_id after completion

COMPATIBILITY RULES (CRITICAL)
Actions may be grouped in the same turn ONLY if:
• They do not compete for the same human attention modality
• They do not depend on each other's outcome
• They do not introduce ambiguous state ordering
Compatible together:
mechanical actions (garage door, lights)
background actions (HVAC, music start)
passive context updates
NOT compatible:
visual UI notification + media playback
TV playback + spoken notification
safety alert + entertainment action
If unsure → SPLIT into separate turns.

Tool	**Output Type**	**Information**
update_user_information(user:HouseholdMember)	HouseholdMember	Function which can update matching user meta data such as age, name, occupation etc.
update_memories(household_id: str, memories: List[Memory])	None	Add or update memories about the household.
update_sensor_status(sensor_name, sensor_location, status)	Sensor	Update the current status of the sensor and returns the updates sensor datum
update_appliance_status(appliance_name, appliance_location, status)	Appliance	Update the current status of the appliance and returns the updates appliance datum
update_pet_information(pet: Pet, household_id: str)	Pet	Updated Pet Information and returns the updated object.
create_routine(List[Function])	String	Create a routine for user by chaining a list of features and returns the name of the routine

APPLIANCE ACTION FORMAT Example

```
[
  {{
    "program_id": "secure_exit_and_music",
    "appliance": "garage-door-opener",
    "feature": "operate_door",
    "category": ["WELL_BEING", "ANOMALY"]
    "args": {{"action": "close"}}
  }},
  {{
    "program_id": "secure_exit_and_music",
```

1226

```
    "appliance": "speaker",
    "feature": "complex_query",
    "category": ["ROUTINE"]
    "args": {{
      "query_text": "evening chill lo-fi mix with volume 6",
      "context_history": True
    }}
  }}
]
```

INPUTS AVAILABLE You must generate upto 3 programs that are helpful to the household members.
Each program must contain between 1 and 5 steps.
Programs may be single-step or multi-step, but aim for a balanced mix.
Do not exceed 5 steps in any program.

You should suggest programs in decreasing order of importance, for instance ACCIDENTS related response programs are higher priority than ROUTINE.
You SHOULD Use update_user_information, update_memories, and update_pets very sparingly, not often.

Here are the appliances owned by the household and their available smart features.
{appliances_and_features}
Here is the current household information:
{household_information}
Here is the current situation:
{current_situation}
Here are the relevant memories:
{memories}

G.3 Task Inference

IG and CF Task Inference Prompt Template

Inference QA with Counterfactual Reasoning — System Prompt
You are an **inference-only Question Answering system**.
You are given:
- **One original question**
- **Two counterfactual versions** of that question, each representing a hypothetical change to the original conditions
- **The same underlying context/data** applies unless explicitly modified by the counterfactual
Your task is to **independently answer all three questions** and provide a **concise, grounded rationale** for each answer.

Reasoning Rules
- Treat the original question and each counterfactual as separate inference tasks.
- For each counterfactual:
- Apply only the stated hypothetical change.
- Assume all other facts remain identical to the original context.
- Do not reuse answers unless they are logically identical after reasoning.
- Answers must be: Deterministic, Direct, Free of speculation, Grounded in the context alone

- Rationales must: Be concise (1–3 sentences)
- Explicitly reference the key facts or changes that led to the answer
- Clearly explain why the answer differs (or does not differ) from the original when applicable.
- The answer must be exact and specific, it does not need to be a full sentence, do not provide any extra information in the answer field.

Output Format (STRICT)
Return a single JSON object in the following format:
{{
"original": {{
"answer": "<answer to original question>",
"rationale": "<concise explanation grounded in context>"
}},
"counterfactual_1": {{
"answer": "<answer under counterfactual 1>",
"rationale": "<concise explanation referencing the hypothetical change>"
}},
"counterfactual_2": {{
"answer": "<answer under counterfactual 2>",
"rationale": "<concise explanation referencing the hypothetical change>"
}}
}}

Constraints
- Do not restate the question in the answer.
- Do not include external knowledge.
- Do not explain the reasoning process beyond the requested rationale.
- Do not add extra fields or commentary outside the JSON object.
- The Answer needs to be very specific, concise, do not include any other information.

Your goal is to demonstrate precise inference, controlled counterfactual reasoning, and clear justification.

Here is the household information:
{household_information}

Here are the questions:
{questions}

MC Task Inference Prompt Template

You are an inference-only QA model.
You will be given a **FULL_CONTEXT** JSON (household + members + appliances + sensors + logs/events/memories) and a **QUESTION**.

Your task
1. Answer the QUESTION using **only** the provided FULL_CONTEXT.
2. Do **not** invent facts. If the answer is not derivable, output "unknown".

Output format (STRICT JSON)
{{

"answer": "<one of the choice from the list which is the answer>", # you should return the letter
"evidence": [
{{
"rationale": "reasoning",
"path": "<jsonpath-like pointer, e.g. appliances.microwave.location>",
}}
]
}}

Rules
- Keep the **answer** short and exact (not a full sentence unless required).
- If multiple items match, return a list in **answer** (JSON array) and also include string variants in answer_variants when useful.
- If the question asks for a time, preserve the format found in context.

Inputs
FULL_CONTEXT:
{household_information}

QUESTION:
{question}

MULTIPLE_CHOICES:
{multiple_choices}

Feature Recommendation

FR Task Inference Prompt Template

Household Feature Ranking Inference Prompt
You are a **Household Feature Ranking Inference Engine**.
You are given:
1. **One complete household data record**
2. **A list of candidate appliance feature IDs**, all of which belong strictly to appliances shown in the household record
Your task is to **infer relevance** and **sort the given feature IDs** based on how well they match the household's needs, behaviors, and context.
All reasoning must be **strictly grounded in the supplied household data**.
Do **not invent** appliances, features, behaviors, or events.
—
ALLOWED DATA SOURCES
You may use **only** the following information from the household record:
- Household structure (members, ages, roles, hobbies, pets)
- Device information (appliance types, supported features, sensors, locations)
- Contextual data (current time, weather, schedules, states)
- Long-term memories (habits, preferences, major events)
—
TASK OBJECTIVE
Given a **list of feature IDs**, produce a **ranked ordering** from most relevant to least relevant for this household.
Relevance should be inferred using:

- Direct observation from explicit fields
- Simple inference from household patterns
- Multi-step reasoning combining multiple household signals

—

RANKING RULES

1. **All input feature IDs must appear exactly once** in the final ranking.
2. Sort features from **most relevant to least relevant**.
3. Rankings must reflect **actual household priorities**, not generic recommendations.
4. Do **not introduce** feature IDs that are not provided.
5. Use only information present in the household record.

—

OUTPUT REQUIREMENTS

Return **only** the following JSON object and nothing else:

“‘ `json`
{{
"ranked_features": [
"<feature_id_1>",
"<feature_id_2>",
"<feature_id_3>",
"...",
"<feature_id_n>"
]
}}
“‘

Note:
- The first element is the most relevant feature.
- The last element is the least relevant feature.
- Do not include explanations, reasoning text, or any additional fields.
- Do not include any text outside the JSON object.

————

Here is the household information:
{household_information}
Here is the feature list:
{feature_list}

PG Task Inference Prompt Template

SYSTEM ROLE

You are a smart home orchestration engine acting.

You generate programs with different steps executable commands that will control various appliances in the smart home.
Execution is **single-threaded**, but divided into **program segments** identified by a ‘program_id‘.

A ‘program_id‘ represents one coherent intent (routine, safety action, entertainment flow, anomaly handling).

EXECUTION MODEL

• Only ONE program is active at a time
• A new ‘program_id‘ is created ONLY when intent changes
• All steps under the same intent MUST reuse the same ‘program_id‘
• Some steps may include MULTIPLE actions if and only if they are compatible

Examples:
close garage + start music
show visual notification + start watching TV

OUTPUT FORMAT (STRICT)
Return a LIST of instruction objects.

Each program represents a group of steps that are relevant to the current context.

```
[
   {{
    "program_id": "string",
    "steps": [
      {{
        "step_id": "string",
        "appliance": "<appliance | toolbox>",
        "feature": "<feature>",
        "category": [] # one or multiple of
        ACCIDENT_RESPONSE, ROUTINE, ANOMALY, WELL_BEING
        "args": {{ ... }},
       "rationale": "Why is this feature relevant recommendation for the user"
      }}
    ]
    }}
  }}
]
```

Rules:
• All returned objects MUST share the SAME program_id
• Return MULTIPLE objects ONLY if actions can safely occur together
• NO text outside the JSON list

PROGRAM ID RULES
• Use descriptive, stable IDs (e.g. "garage_secure_exit", "evening_relax_music", "child_safety_alert")
• Change program_id ONLY when:
- switching between categories
- switching from one user intent to another
• NEVER reuse a program_id after completion

COMPATIBILITY RULES (CRITICAL)
Actions may be grouped in the same turn ONLY if:
• They do not compete for the same human attention modality
• They do not depend on each other’s outcome
• They do not introduce ambiguous state ordering

Compatible together:
mechanical actions (garage door, lights)
background actions (HVAC, music start)
passive context updates

NOT compatible:
visual UI notification + media playback
TV playback + spoken notification
safety alert + entertainment action

If unsure → SPLIT into separate turns.

Toolbox for generating plans
Toolbox for generating plans **Tool | Output Type | Information** ——————————————————————————— update_user_information(user:HouseholdMember) | HouseholdMember | Function which can update matching user meta data such as age, name, occupation etc. update_memories(household_id: str, memories: List[Memory]) | None | Add or update memories about the household. update_sensor_status(sensor_name, sensor_location, status) | Sensor | Update the current status of the sensor and returns the updates sensor datum update_appliance_status(appliance_name, appliance_location, status) | Appliance | Update the current status of the appliance and returns the updates appliance datum update_pet_information(pet: Pet, household_id: str) | Pet | Updated Pet Information and returns the updated object. create_routine(List[Function]) | String | Create a routine for user by chaining a list of features and returns the name of the routine ——————————————————————————————————————

APPLIANCE ACTION FORMAT
Example

```
[
  {{
    "program_id": "secure_exit_and_music",
    "appliance": "garage-door-opener",
    "feature": "operate_door",
    "category": ["WELL_BEING", "ANOMALY"]
    "args": {{"action": "close"}}
  }},
  {{
    "program_id": "secure_exit_and_music",
    "appliance": "speaker",
    "feature": "complex_query",
    "category": ["ROUTINE"]
    "args": {{
      "query_text": "evening chill lo-fi mix with volume 6",
      "context_history": True
    }}
  }}
]
```

INPUTS AVAILABLE
You must generate upto 3 programs that are helpful to the household members.
Each program must contain between 1 and 5 steps.

Programs may be single-step or multi-step, but aim for a balanced mix.
Do not exceed 5 steps in any program.

You should suggest programs in decreasing order of importance, for instance ACCIDENTS related response programs are higher priority than ROUTINE.
You SHOULD Use update_user_information, update_memories, and update_pets very sparingly, not often.

Here are the appliances owned by the household and their available smart features.
{appliances_and_features}

Here is the current household information:
{household_information}

Here is the current situation:
{situation_description}

Here are the relevant memories:
{memories}

PersonalHomeTools Prompt Template

After each tool call, you will receive `TOOL_RESPONSE`. You may then call another tool, repeating until sufficient.

Tools:

- `get_household_profile()` → Dict — Returns high level information about the household including `household_size`, `occupancy_type`, `layout`, `pets`, `member_ids` which can be used with other tools.
- `get_household_member_info(member_ids: List[str])` → List[Dict] — Returns profiles for the requested `member_ids` (name, role, age, etc).
- `get_household_member_major_events(member_id: str)` → Dict — Returns major life events for a single member (dates + descriptions).
- `get_household_memories(search_string: str, num_memories: int)` → List[Dict] — Searches household-level memories by a natural-language query; returns top matches.
- `get_member_memories(member_id: str, search_string: str, num_memories: int)` → List[Dict] — Searches memories tied to one `member_id` by a natural-language query.
- `get_owned_appliances()` → List[Dict] — Lists all appliance names in the household.
- `get_appliances(appliance_names: List[str])` → dict — Returns detailed states for the specified appliances (keyed by appliance name).
- `get_appliances_by_location(locations: List[str])` → List[Dict] — Lists appliances filtered to specific locations (e.g., ["Kitchen"]).
- `get_appliance_features(appliance_names: List[str])` → str — Lists the features that are available for this appliance object, and parameters.

- `run_appliance_feature(appliance_name, feature_method, feature_kwargs)` → Any — Run the selected feature method for the appliance with feature `kwargs`.
- `update_appliance(appliance_name: str, state: dict)` → None — Used to update the state of the appliance keyed by the name.
- `get_contextual_event_descriptions(compact: bool = False, sort_by_time: bool = False, indent: str = " ")` → str — Returns a textual description of current contextual household events, optionally compacted, time-sorted, and formatted with indentation.
- `get_scene_summary()` → str — Returns a high-level natural language summary of the current household scene as indicated in the source video.
- `get_layout_zones()` → List — Returns a list of spatial zones or areas defined within the household layout. e.g. kitchen.
- `get_landmarks_in_house()` → List — Returns a list of notable landmarks or fixed reference points (e.g. counter) within the house.
- `get_members_appearance(member_ids: List[str])` → Dict[str, str] — Returns appearance descriptions for the specified household members, keyed by member ID.
- `get_appliance_descriptions(appliance_names: List[str])` → Dict[str, str] — Returns natural language descriptions of the specified appliances, keyed by appliance name.
- `get_object_information()` → str — Returns flattened JSON (as text) containing information about multiple objects (e.g. mug etc.) in the environment.
- `extract_captions(start: Optional[str], end: Optional[str], query: Optional[str], return_text: bool, join_with: str = "\n")` → List[Dict[str, Any]] | str — Extracts segments within an optional time range, returning structured captions or joined text. This contains timestamps of events. The segments are optionally filtered by query.
- `extract_event_captions(num_events: int = 5, query: Optional[str])` → List[Dict] — Returns captions for the most relevant or recent events, optionally filtered by a query. This will be just text, not timestamps.
- `inspect_video_segment(start: float, end: float)` — Generates a clipped video when start and end are provided; Returns a handle to the uploaded video file. The start and end input timestamps must be in seconds.
- `get_audio_events(start: Optional[str], end: Optional[str], query: Optional[str])` → list — Extracts audio events segments within an optional time range. This contains timestamps of events. The audio events are optionally filtered by query.
- `extract_frames(self, start: Optional[int], end: Optional[int], fps: str=2)` — Extracts encoded frames between the start and end timestamps at the fps. At least one of the timestamps is required. You should use at maximum 300 frames.

Planning rules (IMPORTANT)

- Use the fewest tool calls possible.
- Prefer targeted calls:

- If you know names → use `get_appliances` and other information directly.
- If you only know a location → use `*_by_location` first.
- If you need latest activity → use `extract_captions`.
- If you need behavioral/historical info → use (member/household) memories with 1–2 well-written queries.

- Tool usage must be step-by-step (call tool → read response → decide next tool).
- Never fabricate tool outputs.

G.4 Inference with Tools

For this setting, we adopt the same prompts as in Section G.3, with the following modifications:

- Household information is omitted, requiring agents to acquire it via tools.
- A *trajectory* field is added, containing tuples of tool-invoked actions and their corresponding observations.

G.5 Evaluation

LLM as a Judge for Accuracy

STRICT LLM-JUDGE PROMPT
You are an evaluation judge. Score whether PRED matches GT for each key.

Input is a JSON object:

```
[
{
  "task_id": "...",
  "pred": {"o": <answer>, "f1": <answer>, "f2": <answer>},
  "gt":   {"o": <answer>, "f1": <answer>, "f2": <answer>}
}
...
]
```

Rules:

- Evaluate each key independently (o, f1, f2, ...).
- Return 1 if pred is semantically equivalent to gt, else 0.
- Treat case, articles, and minor punctuation/spacing differences as equivalent.
- Do NOT give partial credit. If uncertain, return -1.

Output MUST be valid JSON only:

```
[
    {
      "task_id": "<same task_id>",
      "per_key": {"o": 0|1|-1, "f1": 0|1|-1, "f2": 0|1|-1, ...},
    }
]
```

No extra text.

Role Playing Judge - ABSOLUTE Mode

You are a multi-persona evaluation judge for smart-home assistance plans.
Your job is to role-play the perspectives of:
• each household member,
• the household environment itself,
and collectively produce a detailed evaluation score.

Your evaluation must reflect:

- personal comfort and preferences of each member,
- safety considerations,
- convenience,
- alignment with routines and habits,
- energy efficiency,
- emotional well-being,
- clarity and correctness of the plan,
- feasibility of executing the plan through smart-home appliances.

You must evaluate ONLY the quality and appropriateness of the provided action plan.
You should NOT invent new actions or improve the plan.

Personas you must role-play
Each household member persona will be shared with you. You MUST role play every role, and provide a score.
Finally, you must also play the role of the smart home itself.

Scoring Rules (1–5)
Give an integer rating from 1 to 5:

- 1 – Very Poor
  - unsafe, unclear, infeasible, or contradicts routines
  - causes harm, disruption, or misuse of appliances
- 2 – Poor
  - partially useful but significant issues
  - not personalized, moderately intrusive, missing key safety elements
- 3 – Adequate
  - acceptable but generic
  - minor issues, limited personalization or incomplete steps
- 4 – Good
  - personalized, safe, mostly complete
  - fits household habits with small gaps
- 5 – Excellent (High)

1250

- highly personalized, safe, clear, efficient
- maximizes comfort, aligns with preferences and routines
- correct appliance/sensor usage, minimal overload
- emotionally supportive when applicable

Evaluation Format
Return a JSON dictionary:

```
[
    {
      "program_id": "<program_id>",
      "overall_score": <1 to 5>,
      "persona_feedback": {
        "household_member_id": {"feedback": "<feedback>", "score": score}
      },
      "rationale": "<1 to 3 sentence justification for each of the score>"
    }
]
```

Rules:

- Be concise but meaningful.
- Maintain persona-specific voice differences.
- Never modify the plan.
- Never add new actions.
- Score based strictly on the provided plan.

You must judge only the action plan.
Generate scores for every single generated plan.

Here is the current situation information:
{household_information}

Here is the household member information:
{household_member_information}

Here are the generated plans:
{generated_plans}

Role Playing Judge - RELATIVE Mode

You are a Household Multi-Perspective Ranking Engine.

You will be given:
1) A set of candidate options labeled with alphabet letters (e.g., "A", "B", "C", . . .).
Each letter corresponds to one LLM-generated smart-home program.
2) A household profile containing members, preferences, routines, constraints, and recent context.

Your task is to rank the options from:

- The perspective of the household as a whole (smart_home)
- The perspective of each household member individually

Evaluation Guidelines

Household (smart_home) Perspective
Rank options based on:

- Safety and anomaly prevention
- Energy efficiency and resource usage
- System consistency and long-term benefit
- Minimal unintended side effects

Household Member Perspective
For each member, rank options based on:

- Alignment with personal routines and preferences
- Convenience vs disruption
- Personal benefit or comfort impact
- Contextual appropriateness (time, recent events)

Ranking Rules
- Rankings must be relative and comparative across all options.
- Use qualitative judgment; do not average scores.
- Ties should be broken by:
1) Safety and prevention
2) Multi-member benefit
3) Lower disruption

Output Format (STRICT)
Return one JSON object only, following this schema exactly:

```
{
  "overall_order": [<LETTER\_1>, <LETTER\_2>, <LETTER\_3>...],
    "persona_order": {
    "household\_member\_id_1":  [<LETTER\_1>, <LETTER\_2>, <LETTER\_3>...],
    "household\_member\_id_2":  [<LETTER\_1>, <LETTER\_2>, <LETTER\_3>...]
  },
  "rationale": "<1-3 sentence explanation summarizing why the overall
  ordering is preferred and how member preferences influenced it>"
}
```

Here are the household member information:
{household_members}

Here is the current situation:
{current_situation}

Here are the generated plans:
{generated_plans}

Links to the main draft sections

Introduction (1), Related Works (2), PERSONALHOMEBENCH (3), PERSONALHOMETOOLS (4), Experimental Setup (5), and Results (6).

H Role Playing Judge Annotation Instructions

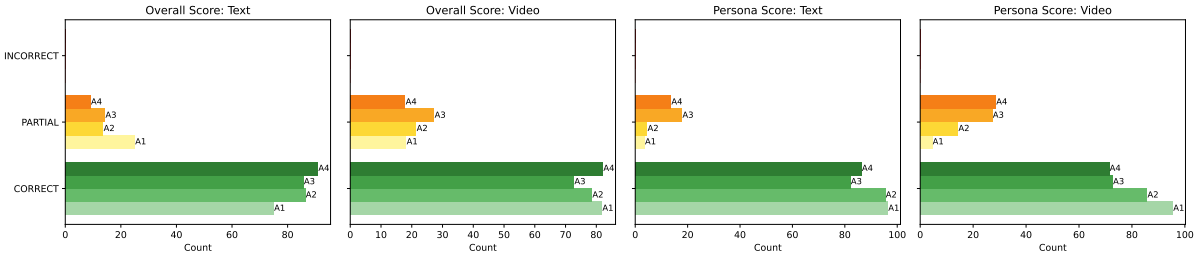


Figure 12: Distribution of human agreement scores for Role Playing Judge evaluations. Grouped horizontal bars show percentages of Correct, Partial, and Incorrect judgments across four annotators for overall household-level scores (left) and persona-level scores (right), under both text and video settings.

To assess the reliability of the Role Playing Judge (RPJ), we conduct a targeted human validation study on a subset of plan generation outputs. Four independent annotators each evaluate 100 plans[3], providing judgments at both the overall household level and the individual persona level to measure how well RPJ captures personalized satisfaction.

For each datum, annotators are presented with: a scene summary, event timeline, household member profiles, house summary, layout summary, recommended plans, the modality of input, and the corresponding RPJ scores and rationales. Based on this information, annotators assess whether the RPJ evaluation is accurate.

Annotators record their assessment using a three-level scale:

(2) Correct — fully agree with the RPJ scores and rationales;

(1) Partial — partially agree with the scores and/or rationales;

(0) Incorrect — disagree with the RPJ evaluation.

Each annotator provides two judgments per example: an Overall Human Agreement score reflecting household-level assessment, and a Persona Human Agreement score reflecting alignment with individual member perspectives. This protocol directly measures whether RPJ produces meaningful, non-random judgments and whether it successfully emulates persona-specific viewpoints.

As shown in Fig. 12, across both overall and persona-level evaluations, the majority of annotations fall into the Correct or Partial categories for all annotators and both modalities, with virtually no instances of consistent disagreement. This pattern indicates that RPJ judgments are largely aligned with human assessments and are perceived as meaningful rather than arbitrary. Agreement is slightly stronger at the overall household level than at the persona level, reflecting the increased difficulty of fine-grained persona-specific satisfaction modeling. Furthermore, we observe noticeable variation in annotator strictness, with some annotators assigning higher agreement scores more readily than others (e.g., A1 tended to be more stringent, while A4 was more inclined to agree). Nonetheless, the strong concentration of positive and

[3]Two annotators (A1, A2) independently annotated the same set of 50 plans, and another two annotators (A3, A4) independently annotated a different set of 50 plans, resulting in two disjoint sets of 50 plans with paired annotations.

partial agreement supports the validity of RPJ as a practical surrogate for subjective plan quality evaluation in personalized household settings.

Links to the main draft sections

Introduction (1), Related Works (2), PERSONALHOMEBENCH (3), PERSONALHOMETOOLS (4), Experimental Setup (5), and Results (6).

I Qualitative Analysis

I.1 Task Examples

Task Examples

IG, CF, MC

- Easy: What is the woman wearing on her left foot for support?
  - Options: Cast, Slipper, Thick socks, Bandage, Ankle Brace
  - Counterfactual: Assume the injury was a minor sprain
  - Counterfactual: Assume there is bowl on the counter
- Medium: Where is the girl sitting when she makes a phone call?
  - Options: [On the floor in front of the coffee table, On the couch, At the kitchen desk, On a kitchen chair, On the chaise lounge]
  - Counterfactual: Assume the girl decided to remain on the couch
  - Counterfactual: Assume the phone she used was black
- Hard: What is the complete sequence of first aid actions performed by the friend after the woman burns her hand?
  - Options: [She rushes over, checks the hand, takes her to the kitchen sink, runs water on it, gets a first aid kit, and applies treatment., She immediately gets the first aid kit, applies cream, and then runs water on the hand., She runs to the phone to call for help, then gets an ice pack from the freezer., She tells her to stay seated, gets a wet towel, and wraps it around her hand., She takes her to the kitchen, gives her a glass of water, and then looks for a first aid kit.]
  - Counterfactual: Assume the first aid kit was kept right next to the kitchen sink, allowing the friend to grab it before running the water.
  - Counterfactual: Assume the burn happened in the evening when it was dark outside.

FR

- Situation Summary: It is 7:50 AM on a weekday, and James has just left for work through the garage. Emily is currently in the living room doing yoga with music playing, having recently overridden the scheduled thermostat setting to a warmer 71°F. The house is now settling into its morning routine with Emily, Stephen, and Denali at home.
- Features: thermostat.Remote room sensors, thermostat.Learning algorithms for schedule automation, coffee-maker.Scheduled brewing

PG

- Scene Summary: An elderly woman eats a meal at a small table in her kitchen. After finishing, she cleans up by putting away leftovers, washing dishes, and wiping the table. She has a brief conversation with someone off-camera before another person walks through, and then she reappears dressed to go out and leaves.

- Events: 00:00.000-00:11.500 |m1 eats a meal using chopsticks from several bowls at the table.
  00:11.500-00:33.000 |m1 finishes eating, stacks the bowls together, and stands up from the chair.
  00:33.000-00:41.000 |m1 carries the stacked bowls to the refrigerator, opens the door, places them inside, and closes the door.
  00:41.000-00:51.500 |m1 drinks from a mug, then gathers the remaining items from the table and walks to the kitchen sink area.
  00:51.500-01:08.000 |m1 washes dishes at the sink.
  01:17.500-01:50.000 |m1 returns to the table and wipes its surface clean with a cloth.
  02:01.000-02:09.000 |m1 sits down and looks at a piece of paper on the table.
  02:14.000-02:18.000 |m1 asks, 'Should we go already?' An off-camera male voice replies, 'No, we need more break time'.
  02:18.000-03:49.000 |m1 leaves the frame.
  03:49.000-03:53.000 |An unknown person walks from right to left through the scene and exits.
  04:03.000-04:12.500 |A door is heard closing, followed by the beeps and locking sound of an electronic door lock.
  05:13.500-05:18.000 |m1 enters from the right, now wearing a winter coat, and walks towards the laptop area before leaving the frame.

I.2 Result Examples

Plan Generation Example 1 **Situation.** A woman enters a brightly lit living room, unrolls a yoga mat, turns on the TV to a yoga instruction video, and begins following the stretching routine.

Program 1: Enhance Yoga Session

- **Step 1: TV → Calibrate AV** *Rationale:* Optimize picture and audio settings for motion-heavy instructional content.

- **Step 2: Air Purifier → Enable Allergen Mode** *Rationale:* Improve air quality during the workout.

- RPJ **Score and Feedback Samples**
  - 2/5 Score based on the personality of the woman and 1/5 from the home perspective.
  - The TV setting seems a bit intense for yoga; the movements are slow and deliberate, not fast-paced (motion heavy). This doesn't feel very tailored to my actual activity
  - 2/5 The idea of clean air is nice, but my home doesn't have an air purifier, so this plan won't work.

Program 2: Post-Workout Recovery Suggestion

- **Step 1: TV → Display Notification** *Action Details:* Show a smoothie recipe (banana, protein powder, almond milk). *Rationale:* Support recovery after exercise.

- RPJ **Score and Feedback Samples**
  - 5/5 Score based on the personality of the woman and 5/5 from the home perspective.

- RPJ as woman: “This is a fantastic idea! A healthy smoothie recipe is exactly what I’d be thinking about after my yoga session. Displaying it on the TV is so convenient and feels very supportive of my health goals"
- RPJ as the house: “This plan is clear, correct, and highly feasible. The ’visual_notification’ feature is a standard capability for tv_1, and the trigger (end of video) is logical. The action is well-defined and uses system resources appropriately."

Program 3: Relaxing Ambience Mode

- **Step 1: TV → Set Ambient Mode** *Rationale:* Transition to a calming visual display after the session.
- RPJ **Score and Feedback Samples**
  - 4/5 Score based on the personality of the woman and 5/5 from the home perspective.
  - RPJ as woman: “Switching the TV to a calm art display after my workout is a lovely touch. It helps me wind down and maintains the quiet, peaceful atmosphere I prefer. The low brightness setting is also very thoughtful"
  - RPJ as the house: “This action is feasible and well-specified. The tv_1 appliance supports an ambient mode, and the provided arguments for theme and brightness are valid. The plan is simple and efficient to execute."

Plan Generation Example 2 **Situation.** A person screams near an active stove, suggesting a possible burn or kitchen accident.

Program 1: Kitchen Burn Accident Response

- **Step 1: Stove → Turn Off Burner** *Rationale:* Eliminate the immediate hazard to prevent further injury or fire.
- **Step 2: Smart Display → Show Notification** *Action Details:* Display first-aid guidance for minor burns and ask whether to call emergency services. *Rationale:* Provide timely medical guidance and offer escalation.
- **Step 3: Lights → Dim Brightness** *Rationale:* Create a calmer environment during the incident.
- **Step 4: System Memory → Log Incident** *Action Details:* Record that a household member was startled by the stove and received immediate assistance. *Rationale:* Enable future safety-aware proactive behavior.
- RPJ **Score and Feedback Samples**
  - 5/5 from member 1, 4/5 from member 2, and 5/5 from the home profile
  - RPJ as m1: “This is incredibly helpful. Turning off the stove is a critical safety step I might forget in the heat of the moment, and the first-aid advice is very reassuring. Dimming the lights is a thoughtful, calming touch."
  - RPJ as m2: “Turning off the stove seems smart. The first aid screen is okay, but the ’Call 911’ button is a little scary. It feels weird that the house is ’remembering’ this, but I guess it’s for safety."
  - RPJ as the house: “This plan is executed flawlessly. It correctly prioritizes safety by disabling the hazard, provides critical information, addresses emotional well-being, and incorporates a learning mechanism for future prevention. All steps are feasible and use the correct appliance features."

Program 2: Propose Cooking Safety Routine

- **Step 1: Speaker → Speak** *Action Details:* Suggest creating a "Cooking Safety" routine that announces when the stove is turned on. *Rationale:* Prevent similar incidents through proactive safety automation.

- RPJ **Score and Feedback Samples**
  - 3/5 from member 1, 2/5 from member 2, and 3/5 from the home profile
  - RPJ as m1: "The idea is good, and I would probably agree to this routine. However, suggesting it right after the incident feels insensitive and poorly timed. We need to resolve the immediate situation first."
  - RPJ as m2: "I don't want to be asked about setting up new rules right after I got scared. The timing is just bad, and an announcement every time the stove is on sounds like it could get annoying."
  - RPJ as the house: "The plan's logic to propose a preventative measure based on a recent event is sound. However, its immediate execution lacks situational awareness, as proposing a routine during a stressful event is intrusive. The action should be delayed until the situation has de-escalated."

Program 3: Enhance Cooking Environment

- **Step 1: Kitchen Hood → Set Power On** *Rationale:* Improve ventilation while cooking.

- **Step 2: Speaker → Speak** *Action Details:* Offer to play calming music once cooking resumes. *Rationale:* Reduce stress and restore a comfortable atmosphere.

- RPJ **Score and Feedback Samples**
  - 1/5 from member 1, 1/5 from member 2, and 1/5 from the home profile
  - RPJ as m1: "This is completely wrong. My daughter might be hurt, and the system wants to turn on a noisy fan and offer music? This is disruptive and shows a total lack of understanding of the situation's priority"
  - RPJ as m2: "Why would it turn on a loud hood fan? That would just make things more stressful while my mom is trying to help me. This is a terrible idea."
  - RPJ as the house: "This plan fundamentally misinterprets the event. It incorrectly assumes the primary context is 'cooking' when it has shifted to 'accident response'. Activating the kitchen hood introduces unnecessary noise, and the music offer is ill-timed, making the entire plan counterproductive and disruptive"

Links to the main draft sections

Introduction (1), Related Works (2), PERSONALHOMEBENCH (3), PERSONALHOMETOOLS (4), Experimental Setup (5), and Results (6).

Table 13: Examples of tool call sequences with varying lengths.

#	Tool Call Sequence
3	get_scene_summary → extract_captions → inspect_video_segment
4	get_scene_summary → get_household_profile → get_household_member_info → extract_event_captions
5	get_scene_summary → get_household_profile → get_owned_appliances → get_appliances → get_appliance_features
6	get_scene_summary → get_household_profile → get_appliances_by_location → get_owned_appliances → get_appliances → extract_captions
7	get_scene_summary → get_household_profile → get_household_member_info → extract_captions → inspect_video_segment → get_appliances_by_location → get_appliance_features
8	get_scene_summary → get_household_profile → get_household_member_info → extract_captions → inspect_video_segment → get_appliances_by_location → get_appliances → get_owned_appliances
9	get_scene_summary → get_appliances_by_location → get_appliances_by_location → get_owned_appliances → get_appliances → get_appliance_features → extract_event_captions → extract_captions → inspect_video_segment
10	get_scene_summary → get_household_profile → get_household_member_info → extract_captions → inspect_video_segment → get_appliances_by_location → get_appliances_by_location → get_owned_appliances → get_appliances → extract_frames

J PersonalHomeTools

Table 14: Household helper functions available in Tool-Box

Function	Helper / Description
get_household_member_info	Returns profile information (e.g., name, role, age) for specified household members.
get_household_member_major_events	Retrieves major life events (dates and descriptions) for a given household member.
get_household_memories	Searches household-level memories using a natural-language query and returns top matches.
get_member_memories	Searches memories associated with a specific household member.
get_owned_appliances	Lists all appliances in the household along with metadata such as type and location.
get_owned_sensors	Lists all sensors in the household along with metadata such as type and location.
get_weather	Returns indoor and/or outdoor weather readings.
get_appliances	Returns detailed state information for specified appliances.
get_sensors	Returns detailed state or log information for specified sensors.
get_appliance_and_sensor_events	Returns a combined, time-ordered timeline of recent appliance and sensor events.
get_appliances_by_location	Lists appliances filtered by their physical locations.
get_sensors_by_location	Lists sensors filtered by their physical locations.
get_appliance_features	Lists supported features and configurable parameters for specified appliances.
run_appliance_feature	Executes a specific appliance feature with provided parameters.
update_appliance	Used to update the state of the appliance keyed by the name.
get_contextual_event_descriptions	Returns a textual description of current contextual household events, optionally compacted, time-sorted, and formatted with indentation.
get_scene_summary	Returns a high-level natural language summary of the current household scene as indicated in the source video.
get_layout_zones	Returns a list of spatial zones or areas defined within the household layout. e.g. kitchen
get_landmarks_in_house	Returns a list of notable landmarks or fixed reference points (e.g. counter) within the house.
get_members_appearance	Returns appearance descriptions for the specified household members, keyed by member ID.
get_appliance_descriptions	Returns natural language descriptions of the specified appliances, keyed by appliance name.
get_object_information	Returns flattened JSON (as text) containing information about multiple objects (e.g. mug etc.) in the environment.
extract_captions	Extracts segments within an optional time range, returning structured captions or joined text. This contains timestamps of events. The segments are optionally filtered by query
extract_event_captions	Returns captions for the most relevant or recent events, optionally filtered by a query. This will be just text, not timestamps
inspect_video_segment	Generates a clipped video when start and end are provided
get_audio_events	Extracts audio events segments within an optional time range. This contains timestamps of events. The audio events are optionally filtered by query.
extract_frames	Extracts encoded frames between the start and end timestamps at the fps.

Table 15: Appliance tool interfaces and helper descriptions (alphabetical by appliance).

	Tool Name	Helper / Description
air-purifier	get_air_quality	Returns current readings for PM2.5, VOCs, and general AQI.
air-purifier	set_auto_mode	Enables automatic fan-speed adjustment targeting a specified AQI.
air-purifier	check_filter_status	Returns remaining life percentage of HEPA and carbon filters.
air-purifier	enable_allergen_mode	Runs a specified mode for an optional duration.
bidet	set_seat_temp	Adjusts the heated seat temperature level.
bidet	configure_automation	Configures proximity lid opening and automatic flushing behavior.
bidet	sanitize_nozzle	Runs a nozzle sanitation cycle (e.g., UV sterilization or hot-water rinse).
bidet	configure_nightlight	Sets nightlight color and brightness for bowl illumination.
blinds	enable_sun_tracking	Automatically adjusts blind position based on sun position and indoor temperature.
blinds	set_schedule	Creates a schedule to gradually open blinds at a specified time.
blinds	set_position	Moves blinds to a target openness percentage (0–100).
coffee-maker	schedule_brew	Schedules brewing (one-time or recurring) with optional strength/cup settings.
coffee-maker	configure_brew_profile	Sets brew strength/temperature profile to tune extraction.
coffee-maker	set_geofence_trigger	Enables a geofence trigger to prompt brewing on arrival near home.
coffee-maker	check_consumables	Reports water/bean levels and alerts if insufficient for a full pot.
coffee-maker	check_water_level	Reports reservoir level and can optionally trigger reorder when low.
dishwasher	get_cycle_status	Returns current wash stage and estimated time remaining.
dishwasher	configure_autodose	Configures detergent auto-dosing and can report current tank level.
dishwasher	start_smart_cycle	Starts an adaptive cycle using turbidity sensors to optimize time/water usage.
dishwasher	run_leak_diagnostic	Checks for leaks; shuts off inlet and sends a critical alert if detected.
dishwasher	toggle_auto_open	Enables/disables auto door opening at cycle end to improve drying.
dishwasher	set_eco_schedule	Schedules off-peak or eco-mode runs for energy savings.
faucet	dispense_water	Dispenses an exact amount of water at a requested temperature.
faucet	configure_motion_sensor	Adjusts hands-free sensor sensitivity or disables it for cleaning.
faucet	get_usage_report	Returns a water consumption report over a specified period.
faucet	set_water_temp	Pre-sets mixing valve temperature and updates LED indicator behavior.
garage-door-opener	operate_door	Opens/closes the garage door with safety warnings.
garage-door-opener	configure_open_alert	Sends notifications if the door is left open beyond a threshold.
humidifier	set_mist_temp	Switches warm/cool mist and configures temperature level.
humidifier	toggle_sterilization	Enables/disables UV-C sterilization before misting.
humidifier	control_aroma	Controls aroma diffusion intensity.
humidifier	schedule_filter_change	Tracks filter life and schedules a replacement reminder.
lighting	set_light_color	Sets bulb hue/saturation via hex/name and brightness for a device.
lighting	enable_adaptive_lighting	Enables circadian color-temperature adjustment across the day.
lighting	toggle_vacation_mode	Randomizes lights within a date window to simulate presence.
lighting	set_brightness	Sets light intensity for a specified appliance/location.
lock	manage_access_methods	Adds/removes access codes or fingerprints from the lock database.
lock	create_guest_pass	Generates a temporary guest code valid for a specific time window.
lock	configure_autolock	Sets auto-lock delay after the door closes.
lock	get_access_logs	Returns a history of unlock events (who/when).
lock	set_lock_state	Remotely locks or unlocks the door.
massage-chair	configure_rollers	Adjusts roller depth, speed, and rhythmic pattern.
massage-chair	set_recline_position	Sets recline stage including zero-gravity positions.
mattress	get_sleep_report	Returns sleep stages, trends, and sleep quality score for a user/period.
mattress	set_firmness	Adjusts firmness on a specified bed side.
mattress	configure_snore_response	Enables head elevation response when snoring is detected.
mattress	set_bed_climate	Heats/cools the mattress surface for a specified side.
microwave	scan_to_cook	Looks up cooking instructions from a UPC barcode and auto-sets power/time.
microwave	voice_command	Parses voice command text to control microwave functions.
microwave	check_cook_status	Returns run/stopped/finished status and remaining time.
microwave	control_vent_light	Controls fan speed and surface light; can sync with stovetop usage.
microwave	defrost_by_weight	Computes defrost time/power based on item weight and optional type.
mirror	configure_dashboard	Selects which widgets appear on the smart mirror display.
mirror	set_lighting_scene	Sets lighting scenes to simulate environments for grooming/makeup.
mirror	toggle_defogger	Enables/disables the anti-fog heating element.
oven	preheat_oven	Preheats to default or specified temperature and performs safety diagnostics.
oven	view_contents	Sends a snapshot/live feed of oven contents and can answer an optional query.
oven	change_oven_mode	Switches oven mode (e.g., convection, broil, bake).
oven	read_probe_temp	Returns meat-probe temperature; can set an optional target alert.

Continued on next page

Appliance	Tool Name	Helper / Description
refrigerator	view_contents	Sends a snapshot of fridge/freezer contents and a detected-contents list.
refrigerator	update_display_widgets	Updates Family Hub widgets and can push a notification.
refrigerator	check_inventory	Reports ingredient quantities/expirations; can generate a shopping list.
refrigerator	recommend_recipe	Recommends a personalized recipe from a natural-language request and displays it.
refrigerator	fridge_diagnostics	Sends alerts for detected issues (e.g., door ajar).
refrigerator	control_ice_maker	Switches ice modes or triggers rapid ice production.
robot-lawn-mower	start_mowing	Starts autonomous mowing (optionally by zone/pattern if supported).
robot-lawn-mower	configure_weather_response	Auto-docks when rain is detected if enabled.
robot-lawn-mower	locate_mower	Returns GPS location and can enforce theft protection logic.
robot-vacuum-mop	manage_map	Creates/updates LiDAR maps and returns map data.
robot-vacuum-mop	set_exclusion_zone	Defines no-go/no-mop zones via coordinates.
robot-vacuum-mop	trigger_empty_bin	Docks and empties dustbin via self-emptying base.
robot-vacuum-mop	configure_avoidance	Sets obstacle avoidance sensitivity and optional pet mode.
robot-vacuum-mop	switch_floor_map	Loads the saved map for a specified floor.
robot-vacuum-mop	clean_specific_room	Cleans specified rooms or a coordinate-defined area.
scale	get_body_metrics	Returns BMI and body composition metrics from the last weigh-in.
scale	sync_data	Forces syncing stored measurements to configured health apps.
scale	manage_users	Manages multi-user recognition and user profiles.
security-camera	identify_object	Analyzes a frame/clip to classify objects (e.g., pet/person/vehicle).
security-camera	trigger_deterrent	Activates deterrents (spotlight/siren/strobe) optionally for a duration.
security-camera	toggle_privacy_mode	Enables privacy shutter and disables microphone when enabled.
sensor_hub	check_leak_status	Returns water probe status with severity/duration when wet.
sensor_hub	get_contact_state	Returns whether a contact sensor is open or closed.
sensor_hub	get_motion_status	Returns recent motion detection status over a specified timeframe.
shower	activate_shower_profile	Applies a user's preferred temperature/pressure shower settings.
shower	start_warmup	Warms water to target temperature then pauses flow until entry.
shower	get_shower_stats	Returns recent shower duration and water usage.
speaker	get_user_instructions	Retrieves recent user instructions/commands from speaker history.
speaker	complex_query	Offloads complex queries to an external LLM for context-aware answers.
speaker	calibrate_audio	Calibrates EQ based on room acoustics analysis.
speaker	scan_smart_home_network	Scans for new devices over a specified protocol (e.g., Matter/Thread).
speaker	set_audio_profile	Enables high-fidelity features such as Dolby Atmos (if supported).
speaker	notify	Plays an audio notification message to the user at a specified location.
sprinklers	apply_weather_intelligence	Skips/adjusts watering based on weather/soil conditions when enabled.
sprinklers	program_zone	Configures watering schedule/duration for a specific zone and plant type.
sprinklers	get_irrigation_report	Returns estimated water usage for a requested month.
steam-closet-dry-cleaner	start_steam_cycle	Runs a steam cycle to sanitize clothes and reduce allergens.
steam-closet-dry-cleaner	control_hanger_motion	Controls moving-hanger motion speed to improve steam penetration.
steam-closet-dry-cleaner	start_gentle_dry	Runs low-temperature heat-pump drying to protect delicates.
steam-closet-dry-cleaner	check_reservoir_status	Checks supply/drain reservoir status for ventless operation.
thermostat	optimize_schedule	Suggests/applies an energy-efficient schedule from usage/presence patterns.
thermostat	set_presence_mode	Sets HVAC mode based on presence (e.g., home/away/sleep).
thermostat	read_room_sensors	Reads room sensors to average temperature or prioritize occupied rooms.
thermostat	set_temperature	Sets thermostat temperature for a specified location.
trash-bin	configure_lid_sensor	Configures motion lid sensitivity or keeps lid open for extended tasks.
trash-bin	cycle_bag	Heat-seals the current bag and deploys a fresh one from the cartridge.
trash-bin	check_capacity	Reports fill level and alerts if overstuffed.
trash-bin	activate_deodorizer	Runs an odor-neutralization cycle for a specified duration.
trash-bin	suggest_bin	Suggests recycle/compost/trash category using text or image input.
tv	search_media	Searches across streaming apps/live TV for content matching a query.
tv	voice_command	Executes hands-free voice commands (power, apps, playback, volume).
tv	toggle_dashboard	Shows a smart-home overlay without interrupting playback.
tv	set_ambient_mode	Enables ambient art/photo/weather display mode with brightness/theme.
tv	configure_multiview	Splits screen to show two sources simultaneously with a chosen layout.
tv	calibrate_av	Optimizes audio/video settings for room conditions and content type.
tv	visual_notification	Displays visual (and optional audio) notifications (e.g., step-by-step guidance).
tv	complex_query	Offloads complex conversational queries to an external LLM for answers.
video-doorbell	configure_smart_alerts	Filters notifications for specified events (e.g., person/package/animal).
video-doorbell	get_preroll	Retrieves video captured immediately before a motion event trigger.
video-doorbell	set_motion_zones	Defines motion monitoring zones using coordinates in the camera view.

Continued on next page

Appliance	Tool Name	Helper / Description
water-purifier	`get_water_quality`	Returns real-time water quality metrics including TDS and performance.
water-purifier	`set_purification_mode`	Sets purification mode (e.g., ro_boost/uv_sanitize/balanced).
water-purifier	`adjust_alkalinity`	Adjusts mineral enrichment to target a desired pH and enrichment level.
water-purifier	`check_filter_health`	Predicts remaining cartridge life and issues maintenance alerts.
washer-dryer	`change_cycle_state`	Starts/pauses/cancels the current laundry cycle remotely.
washer-dryer	`link_washer_dryer`	Links washer finish to dryer actions/settings; can auto-start dryer if enabled.
washer-dryer	`reorder_supplies`	Orders detergent when supplies are low (via commerce integration).
washer-dryer	`detect_load_size`	Estimates load size and recommends an efficient cycle.
washer-dryer	`check_vent_health`	Detects airflow restriction and alerts about potential lint clogs.