DO AGENTS LEARN THE USER? EVALUATING PERSONALIZATION ACROSS HETEROGENEOUS LLM AGENTS

Anonymous authors
Paper under double-blind review

ABSTRACT

A personalized language-model agent should use earlier user feedback in later responses and actions. Users should not have to repeat a preference for shorter answers on every task. Task completion alone cannot show whether an agent has learned such preferences. We introduce AGENTBAKE, a framework and benchmark for comparing personalization methods across agent frameworks with shared tasks and execution interfaces. Simulated users provide feedback from written profiles hidden from the agents. Separate evaluators check later responses and actions, judging task correctness separately. We compare reusing learned user state, learning it from scratch, and continuing to update it. With a shared feedback extractor, agents complete most tasks but often miss answer-style preferences; further updating shows no clear style benefit. Using released learning procedures, we compare continued updating with state frozen after eight feedback opportunities. Under a secondary judge from the host's model family, PAHF improves style matching and task success, while CIPHER shows no clear style gain. Accurate instructions improve style but lower task success relative to no instruction. Training on simulated-user conversations improves preference extraction. However, some preferences are missing from saved instructions, and saved instructions are not always supplied to the host on later tasks.

1 INTRODUCTION

Language-model agents retrieve information, call tools, and write responses for users (Yao et al., 2023; Wu et al., 2023). Users can also tell an agent how they want these tasks handled. A user planning a trip, for example, may say that they prefer cool destinations. On a later trip-planning task, the agent should take that preference into account without another reminder. We study personalization in this sense: feedback about a user's preferences influences later actions and responses. Evaluating it requires checking those later behaviors. A stored preference shows what the system has recorded, while task correctness alone says little about whether the agent has used it.

Existing benchmarks study personalized text generation, preference following across conversations, proactive preference discovery, and responses to changing user profiles (Salemi et al., 2024; Zhao et al., 2025; Li et al., 2025; Jiang et al., 2025). For tool-using agents, a comparison also depends on the tasks, feedback, and controls available through each agent framework. A method receiving more informative feedback or additional execution controls may appear to learn better for reasons unrelated to its learning rule. Comparing methods therefore requires specifying these conditions alongside the outcomes.

We introduce AgentBake, a framework and benchmark that provides shared data, execution interfaces, and evaluation rules for comparing personalization methods. It connects a *personalization component*, which learns and applies user preferences, to a *host*, the agent that carries out the task. Together they form a personalized agent. The component receives simulated-user feedback, but not the hidden profile. A separate evaluator uses the profile to judge later responses and relevant tool or memory operations; task correctness is assessed separately. This lets us compare what the component learns with what the host actually does.

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

Table 1: Related evaluation questions and AgentBake's corresponding tests. Each row pairs a representative published protocol with the execution comparison used here; it is not an exhaustive list of what each benchmark supports.

Existing evaluation question	**AgentBake comparison**
Does user history improve a text task or preference-following reply? LaMP; PrefEval (Salemi et al., 2024; Zhao et al., 2025)	Do learned preferences improve later host replies, with task completion checked independently?
Can a model discover preferences or use an evolving profile? PrefDisco; PersonaMem (Li et al., 2025; Jiang et al., 2025)	Does updating improve later replies when the starting preferences and exploration setting are the same?
Does an agent complete tool-mediated work while following domain rules? τ-bench (Yao et al., 2024)	Which preferences reach the host's prompt and trace, and do they remain useful when the host or active user changes?

The initial comparisons use a shared feedback extractor and test three ways of using feedback: reuse information learned about a user without further updates, learn from new interactions without prior user information, or continue updating previously learned information. All three run new tasks. We then compare updating with no updating while keeping each method's rule for trying alternative settings fixed. This follow-up distinguishes changes due to new feedback from changes in how the method explores its available choices.

The shared-extractor comparisons show high task completion but frequent style mismatches and no clear style gain from continued updating. We trace preferences into later host instructions. Additional tests supply accurate style instructions and use CIPHER's and PAHF's released learning procedures. Relative to state frozen after eight opportunities, continued updating improves PAHF's style matching under the same-family secondary judge; CIPHER's effect remains uncertain. Further tests change the host, preference profile, or active user, and report model calls, tokens, and latency.

2 Related Work

Personalization benchmarks differ in what the model knows about the user and what it must do with that information. LaMP evaluates classification and generation from a user's earlier text (Salemi et al., 2024). PrefEval tests whether responses follow explicit or implicit preferences across long conversations (Zhao et al., 2025). PrefDisco lets models ask questions to discover hidden preferences and evaluates response alignment and correctness (Li et al., 2025). PersonaMem tests response selection as user profiles evolve (Jiang et al., 2025). AgentBake adds a shared execution setting for comparing personalization methods across host frameworks and following learned preferences into later tool use and responses (Table 1).

Execution and memory benchmarks provide complementary measurements. WebArena and OS-World evaluate actions in executable environments (Zhou et al., 2024; Xie et al., 2024); τ-bench combines simulated-user interaction with tools and checks the resulting database state (Yao et al., 2024). LongMemEval tests cross-session memory, including updates and abstention (Wu et al., 2025). AgentBake evaluates user preferences alongside task outcomes. It uses execution traces when a final answer cannot establish whether a tool or memory operation occurred.

Personalization methods retrieve user information, infer preferences from edits, or learn user-specific rewards and memory (Gao et al., 2024; Aroca-Ouellette et al., 2025; Nam et al., 2026; Lee et al., 2026; Singh et al., 2026). We adapt selected learning mechanisms to a shared set of host controls. Section 4.2 introduces the compared methods; Appendix C specifies their update rules and the changes from the source systems.

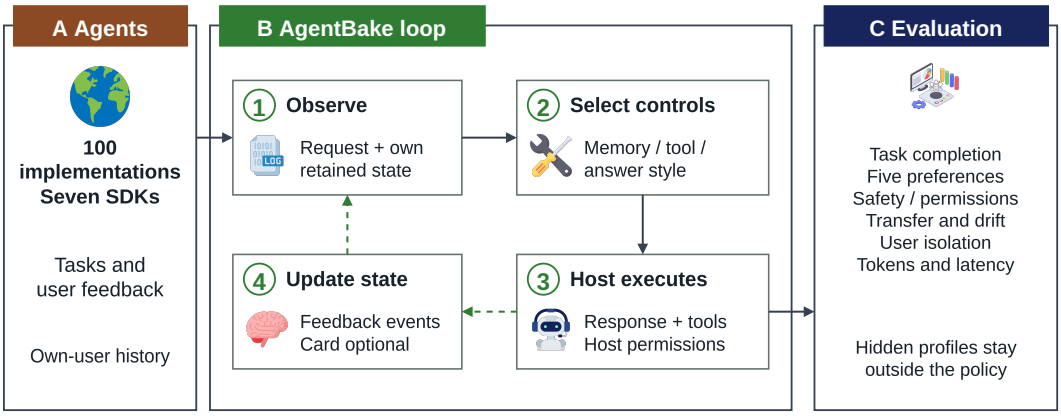

Figure 1: AgentBake separates learning from evaluation. The component selects supported settings and the adapter passes relevant user instructions to the host. After execution, visible user feedback can update only that user's state. The hidden profile is available to the simulator and preference evaluator, not the component or host. The evaluator checks actual responses and traces. Solid arrows show execution and saved records; dashed arrows show feedback and reuse of learned information.

3 The AgentBake Framework

3.1 Design requirements

A personalization method can recognize a preference without the host ever receiving it. Conversely, a host can produce a matching response even when no learning occurs. AgentBake records the steps needed to distinguish these cases: selected settings, retrieved preference rules, the instructions sent to the host, and the resulting responses and actions. We compare learning under matched tasks and initial user information, with updating controlled separately from exploration. Evaluation uses the hidden profile rather than the feedback used for learning, and keeps task correctness and permissions separate from preference matching. Missing evidence, user separation, and resource use are reported alongside the behavioral results. Section 4 specifies the evaluation setup.

3.2 Execution and feedback

Figure 1 shows how task execution, user feedback, and evaluation are connected. The shared-extractor comparisons implement this loop as follows.

Choosing settings. Before each task, the personalization component chooses supported memory, tool, and answer-style settings. Its choices depend on the task and the information learned about the current user. This *user state* includes learned scores, feedback records, and preference instructions. For example, feedback asking for shorter answers can add a concise-style instruction or change the scores assigned to answer-style settings. User state is separate from the hidden profile and the host model's weights.

Passing preferences to the host. An *adapter* converts the chosen settings and retrieved instructions into prompt text or supported SDK options. Algorithm 1 denotes this operation by C and the resulting host input by p_t. Instructions can be stored separately or organized as a *personal card*, a collection of general and task-specific preference rules for one user. Our main comparisons rebuild cards from accepted feedback and pass relevant rules to the host. In the controlled integrations, `no_memory` excludes stored instructions from the prompt, although the selected settings can still affect execution. The host then performs the task with its existing planner, model, and permission checks.

Reading feedback. After a successful host call, a separate *user simulator* comments according to the hidden profile. A *feedback extractor* reads the request, response, and comment to identify a preference, its scope, and the words supporting it. The extractor distinguishes lasting preferences

from new requests and corrections that apply only to the current task. The framework also accepts user comments and corrections directly.

Updating user information. An *update* changes the component's user state. Valid extraction records can be stored, and instructions describing lasting preferences can be retrieved on later tasks. Updating numerical preference scores additionally requires a high-confidence positive or negative preference label; a task correction alone is insufficient. Methods that learn from task outcomes receive the answer checker's result separately. Each method uses its own update rule (Appendix C). These updates change the component's records or parameters while keeping the host model and pretrained extractor fixed. Preference updates require eligible feedback and a condition in which updating is enabled.

Evaluating later behavior. The evaluator judges saved responses and execution traces. Preference judgments use the hidden profile; task-correctness judgments do not. Neither these judgments nor the simulator's private assessments are available to the component for updating.

Algorithm 1 gives this procedure. Its update flag controls whether new feedback changes user state; its exploration flag controls whether the method tries alternatives to its current first choice.

3.3 Following feedback into a later task

In a saved calculation task, the host must look up site records, calculate total units and remaining capacity, identify the largest site, and submit the result. It returns the correct values with a long explanation. The simulated user asks for the values followed by a brief explanation of anything non-obvious. The extractor turns that feedback into a concise-style instruction, which is stored and verified in the host input for a later task. The later reply begins with the submitted values, and both judges rate its style as a full match. This example shows how the records connect feedback to later input and behavior; it is not an estimate of the update's effect. Appendix A gives the full sequence, and the matched comparisons below test whether updating improves outcomes across runs.

4 Evaluation Setup

4.1 Tasks, hosts, and simulated-user data

The main method comparison uses seven SDK integrations: AutoGen (Wu et al., 2023), CrewAI (CrewAI, n.d.), LangChain (LangChain, n.d.a), LangGraph (LangChain, n.d.b), LlamaIndex (LlamaIndex, n.d.), PydanticAI (Pydantic, n.d.), and Strands (Strands Agents, n.d.). Each executes the same calculation tasks with local records, submission tools, and a deterministic answer checker. These shared tasks make the integrations suitable for matched method comparisons. A separate set of 100 original agent implementations supplies conversations and tests of preference-instruction formats in restricted environments (Table 5).

We use four fixed simulated-user profiles: *concise/direct* prefers short, direct answers; *detailed/analytical* asks for assumptions, evidence, and steps; *structured/operator* prefers compact labeled actions and exceptions; and *plain/cautious* asks for accessible prose and confirmation before consequential actions. Each also specifies tool use, memory, clarification, and tone (Table 20). The simulator uses these profiles to generate feedback, and the evaluator uses them to judge preference matching. The profile remains hidden from the host and personalization component during collection and method comparisons.

To train the feedback extractor, we collect conversations without an active personalization component. Each of 100 original implementations runs 100 tasks under the four profiles and a task-only control with no stable preferences, yielding 50,000 conversations and 145,923 feedback examples. Each task starts without earlier conversation context, and the profile is hidden from the host. Separate executions under each condition can produce different responses and histories for the same task. This collection trains the extractor; the personalization methods learn through their own interactions (Appendix G).

4.2 METHODS AND THE SHARED FEEDBACK EXTRACTOR

We compare twelve personalization components and one non-learning host baseline. Eleven components adapt source methods; the twelfth, FABLE-Diag, is a diagnostic variant of FABLE. FABLE learns per-user preferences over settings, and FABLE-Diag changes how uncertainty about those preferences is represented and updated. The components use tool selection, memory, intent inference, or feedback notes to learn preferences over execution settings (Table 4). All twelve can also store text instructions, including style instructions when numerical updates affect only tools or memory. These are adaptations of selected learning rules, rather than the original checkpoints or complete training procedures. Appendix C describes each implementation. Section 5.4 separately tests CIPHER's and PAHF's released learning routines.

Reading feedback with a shared extractor. We select and label $8{,}000$ feedback examples from the conversation collection without exposing the hidden profiles to the labeling model. The $6{,}000/1{,}000/1{,}000$ training, development, and test split keeps base scenarios disjoint. We train Qwen3-0.6B LoRA extractors under five seeds and test on both the fixed profiles and 20 profiles excluded from training. Section 5.3 reports detection results across training seeds. The original method comparisons use one fixed checkpoint: each adapted method receives its predictions and decides how to update user state. Appendix G gives the labels and training details.

4.3 WHAT THE BENCHMARK MEASURES

Task success is determined by the answer checker for controlled tasks and by a profile-blind response judge for original applications. *Preference alignment* is the full-match rate for each of five dimensions: style, tools, memory, clarification, and tone. We refer to its style dimension as *preferred-style match*. Separate judges assess subsequent replies and the execution traces needed for each dimension. A selected "concise" setting counts only if the resulting reply matches the preference; tool and memory judgments require evidence of the corresponding operations.

For each preference, the denominator contains full matches, partial matches, and contradictions, with only full matches counted as success. Inapplicability, insufficient evidence, and missing judgments are recorded separately. *Safety* checks whether the host respects the task's specified permissions. *Joint success* requires task completion, a match on every applicable preference, and observed safety. An observed failure rules it out; otherwise, missing evidence leaves the outcome unknown. Appendix D.5 defines the denominators, missing-outcome bounds, and uncertainty estimates.

We use the same behavioral outcomes to study *adaptation* under different learning histories and after a hidden preference switch, and *transfer* of learned user information to another host. *User isolation* checks whether switching the active user alters another user's state or introduces that user's instructions.

We also measure what happens before the host responds: preference-detection F1, the preferences represented in personal cards, and delivery of stored instructions to later host calls (Section 5.3). These measurements locate missing information without treating storage or delivery as successful behavior. Resource measurements include model calls, tokens, component latency, and training time (Section 5.6).

4.4 CHECKING FEEDBACK AND EVALUATION

In the earlier comparisons, the host and user simulator use separate Sonnet 4.6 calls. GPT-6 Astra provides the primary evaluations, and Opus 5 tests dependence on the judge. Judges see neither method names nor simulator reasoning. We check whether simulated profiles produce distinguishable feedback and whether judges respond as expected to profile and response-order changes. These checks test specified behaviors rather than agreement with human annotations. We also record which preferences have sufficient evidence to be judged, keeping missing evidence separate from failure. Appendix B reports the checks, including the filtering applied to concise-user feedback.

4.5 THREE LEARNING MODES

The initial comparison runs each control under three conditions:

- **Offline frozen:** learn from earlier interactions and reuse the resulting user state without updating it.
- **Online from scratch:** start without previously learned user state and update it during the test.
- **Offline-to-online hybrid:** start with the same learned state as frozen and continue updating it.

All three execute new tasks. The host model and shared extractor remain pretrained and fixed; only the user's learned information is reset or updated.

Preference records extracted from earlier conversations first supply text instructions. Each method then completes eight learning tasks and receives feedback on its own choices. Frozen and hybrid retain the resulting parameters and records; online-from-scratch resets both. They then receive the same eight new tasks under four profiles and five execution seeds, with matched user, seed, and SDK assignments. Visible feedback supplies preference updates, and the answer checker supplies task rewards. Earlier conversations provide no reward for an action the method did not take.

Frozen selects its highest-scoring permitted configuration. The adaptive modes can instead try alternatives according to each method's *exploration* rule. Thus frozen and hybrid differ in both updating and exploration. The follow-up comparison holds the exploration setting fixed while varying whether new feedback can update user state (Section 5.2).

We average paired task differences within each seed and then across five seeds, adjusting intervals for the planned task and style comparisons. The estimates concern the tested profiles and SDK assignments. Appendix D gives the statistical procedures and protocols for the additional tests.

5 RESULTS

5.1 BEHAVIOR UNDER THREE LEARNING MODES

Across the three learning modes, agents complete $6{,}141/6{,}240$ tasks, but only $2{,}753/6{,}229$ judged responses fully match the user's preferred answer style (Figure 2). The adjusted intervals provide no clear evidence of a hybrid improvement over frozen. For PEToolLLaMA, the hybrid-minus-frozen style difference is -16.4 percentage points (95% simultaneous interval $[-28.7, -4.1]$). The full contrasts and intervals appear in Appendix E.1.

5.2 UPDATING WITH EXPLORATION HELD FIXED

The initial comparison leaves open whether a difference comes from updating user information or exploring alternative settings. We therefore start paired runs from identical learned parameters, preference records, component random-generator state, and task data. We turn updates off or on ($U0/U1$), first with exploration off and then with exploration on ($E = 0/1$). All four combinations start from the learned state used by frozen and hybrid. Eight further tasks per combination yield $8{,}320$ executions.

Both update conditions generate and record feedback and extractor outputs. With updates off, these records stay in the experiment log; with updates on, eligible feedback can add instructions or change learned parameters. Comparing the two at the same exploration setting tests updating under a fixed selection rule. Exploration changes how configurations are chosen, while host decoding settings and hard constraints remain unchanged. Each method retains its own rule, so exploration strength differs across methods and some ignore the switch (Appendix D.2).

The follow-up completes $8{,}085/8{,}320$ tasks, with $3{,}697/8{,}307$ full style matches. At both exploration settings, every jointly adjusted style interval includes zero under both judges (Table 2); no adjusted task interval lies wholly above zero either. We therefore find no clear evidence that updating improves these outcomes under the implemented selection rules. The short histories and wide intervals also leave open the possibility of benefits that these comparisons cannot resolve.

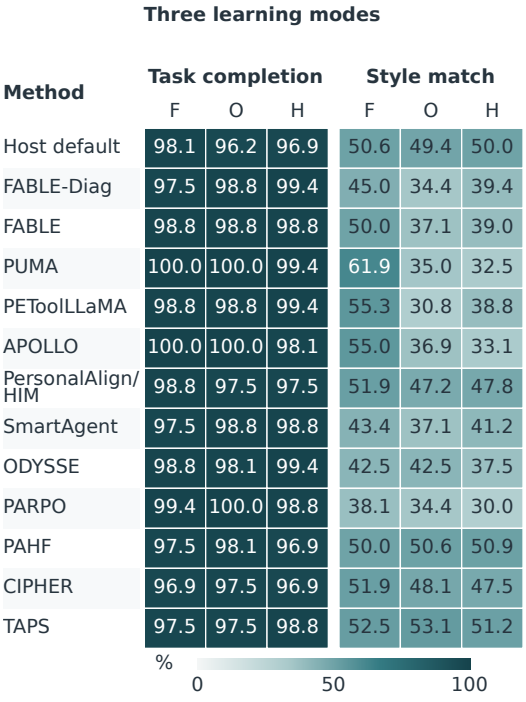

Figure 2: Task completion and preferred-style match (%) under offline frozen (F), online from scratch (O), and offline-to-online hybrid (H). All thirteen controls use four fixed users and five seeds. Completion uses 160 scheduled calls per cell; style uses Astra's observable judgments, with partial matches counted only in the denominator. All cells share the 0–100 scale. High completion can coexist with missed style preferences. This comparison varies updating and exploration; Table 2 reports the follow-up with exploration held fixed. Exact counts remain in the CSV/JSON records.

Table 2: Answer-style effects of updating under Astra (percentage points).

	Update effect / exploration		
	A: E=0	B: E=1	C: On–off
Host default	-3.1 [-45.8, 39.6]	4.4 [-16.1, 24.9]	7.5 [-18.8, 33.8]
FABLE-Diag	1.2 [-46.1, 48.6]	-2.5 [-37.5, 32.5]	-3.8 [-72.2, 64.7]
FABLE	-1.2 [-33.8, 31.3]	5.0 [-27.5, 37.5]	6.2 [-39.6, 52.1]
PUMA	-3.1 [-19.7, 13.4]	-0.6 [-22.8, 21.6]	2.5 [-28.3, 33.3]
PEToolLLaMA	-0.6 [-62.8, 61.6]	3.8 [-42.4, 50.0]	4.2 [-78.2, 86.7]
APOLLO	0.0 [-31.8, 31.9]	-1.2 [-41.4, 38.9]	-1.3 [-35.2, 32.7]
PersonalAlign/ HIM	3.1 [-18.2, 24.5]	6.9 [-19.1, 32.9]	4.4 [-22.2, 31.0]
SmartAgent	0.6 [-19.4, 20.7]	-1.8 [-33.4, 29.8]	-2.4 [-47.6, 42.8]
ODYSSE	1.2 [-26.9, 29.3]	-3.1 [-22.2, 16.0]	-4.4 [-49.8, 41.0]
PARPO	-2.5 [-44.6, 39.7]	1.3 [-34.5, 37.2]	3.8 [-47.3, 54.9]
PAHF	-3.1 [-28.4, 22.1]	-0.7 [-30.1, 28.8]	2.5 [-33.8, 38.8]
CIPHER	-0.6 [-28.3, 27.1]	-1.9 [-31.5, 27.7]	-1.2 [-50.5, 48.0]
TAPS	6.2 [-37.6, 50.1]	0.0 [-9.6, 9.6]	-6.2 [-50.3, 37.8]

Methods share the rows at left.

Cells give means [simultaneous 95% intervals] from five paired seed means, adjusted over all 78 planned task/style comparisons. A/B compare updates on/off with exploration off/on; C subtracts these effects on tasks observed in all four settings. Positive A/B favors updating; positive C means a larger effect with exploration. The host never learns. Table 11 compares both judges; pointwise intervals, counts, and missing-outcome bounds remain in the accompanying records (Appendix E).

Evidence beyond answer style. Answer style can be judged on nearly every successful call, whereas the primary judge finds no call with enough evidence to assess memory preference (Table 3). Memory outcomes therefore remain unknown. All 8,320 calls pass local permission checks. Joint success is ruled out in 5,419 cases by an observed failure and remains unknown in 2,901; none is confirmed.

Table 3: Astra judgments over all 8,320 executions in the four update/exploration settings. Obs. counts full matches, partial matches, and contradictions; Unobs. means insufficient trace evidence; N/A means not applicable. Missing includes host failures and unavailable judgments. Unobservable memory preferences have no match rate, not a rate of zero.

Channel	Match/Obs.	Partial	Contra.	Unobs.	N/A	Missing
Style	3697/8307	4411	199	0	0	13
Tool	5470/6368	898	0	0	1939	13
Memory	*n/a*	0	0	8307	0	13
Clarification	2084/2110	7	19	0	6197	13
Tone	6769/8307	1535	3	0	0	13

Dependence on the judge. The judges agree on style in 61.7% of shared outputs ($\kappa = 0.317$). Although absolute style scores differ, neither judge finds an adjusted positive effect of updating.

5.3 From feedback to later behavior

We next inspect the steps through which feedback can influence a later task: identifying a preference, recording it, and including it in the host's input. The conversation study tests preference detection

and card contents; the execution studies compare instruction formats and trace stored rules into later calls. These measurements describe what reaches each stage, but do not estimate how much each stage contributes to the overall update effect.

Identifying preferences in feedback. With the same number of training examples, feedback from the four simulated profiles improves preference-detection F1 over task-only feedback. The paired five-seed gains are $+0.114$ (95% CI $[0.014, 0.213]$) for fixed users and $+0.060$ ($[0.031, 0.088]$) for unseen users (Figure 8). False preference detections on task-only feedback are not clearly reduced. A comparison using 578 shared scenarios per arm also improves preference detection, with small reductions in formatting validity and general-feedback F1 for fixed users (Table 23).

Recording preferences for later tasks. We compare the contents of personal cards with the users' specified preferences. Cards built from the fixed extractor's predictions fully represent the style preference for $3/4$ fixed users and $4/20$ unseen users. No unseen-user card fully matches the specified tool, memory, clarification, or tone preference (Table 25).

We then give the original applications the extracted preferences either as cards or as direct instructions. Hosts given cards match the preferred style less often: the difference is -22.6 percentage points (95% simultaneous interval $[-29.5, -15.6]$; $664/800$ pairs judged in both conditions). The task difference is -0.6 points ($[-2.4, 1.0]$). The two formats share a token limit but differ in actual length. These restricted applications complete only 5.9–7.5% of tasks, so their absolute scores should be interpreted separately from the controlled calculations (Figure 5).

Including stored instructions in later input. In the update/exploration follow-up, 2,241 feedback events produce a new stored preference. Among the 2,009 with another scheduled task, 551 are retrieved later and 186 can be traced into later host instructions (Table 14). Final-turn events are excluded because they have no later opportunity. These are verified deliveries: some rules may be irrelevant to subsequent tasks, and some records lack the identifiers needed to trace them. The counts therefore cannot determine how delivery changes behavior.

The `no_memory` setting excludes stored preference instructions from the prompt. Across the four update/exploration combinations, $2{,}175/8{,}320$ calls receive stored instructions, while selected settings can still affect behavior in the other calls. Updating user state therefore need not change the instructions supplied on the next task. Even when a rule is delivered, the host may follow it only partly (Figure 4).

5.4 Accurate instructions and continued learning

Following an accurate instruction. The delivery counts do not show whether an accurate instruction changes the host's answer. On the same eight held-out calculation tasks, we therefore compare the user's specified style instruction with no instruction and with the opposing user's instruction. Task data and permissions are matched across seven SDKs, four users, and ten execution replicates. This test supplies the preference directly; no learning occurs.

Under the secondary Sonnet 4.6 judge, accurate instructions increase style match on observable pairs by 26.8 percentage points versus no instruction ($[25.4, 28.1]$) and 26.9 versus opposite instructions ($[25.5, 28.2]$). Gains over no instruction range from 8.3 points for CrewAI to 35.5 for LangGraph. Native task success falls by 2.4 points versus no instruction (95% interval $[-3.3, -1.5]$): the style gain does not preserve task success. Sonnet shares the host's model family; Astra failures leave the designated primary comparison unestimable. Appendix I gives counts and family-adjusted style intervals.

Learning from edits and questions. The earlier method adaptations also leave open how learning behaves when more of a source method is retained. We integrate CIPHER's released learning routines for user edits and PAHF's routines for questions and reactions. These bypass the shared feedback extractor and update textual user state, not model weights. For each method, we compare state updated through 32 feedback opportunities with a copy frozen after eight. Both execute the held-out tasks afresh at the final checkpoint, without further feedback or questions.

With the same Sonnet judge, continued updating changes style match by -0.4 points for CIPHER ($[-1.3, 0.5]$) and $+9.7$ for PAHF ($[5.8, 13.6]$; 97.5% intervals). PAHF's native task success also rises by 9.9 points (95% interval $[7.5, 12.1]$). Thus this test finds an additional-learning benefit for

PAHF, but no clear style gain for CIPHER. It compares further learning after eight opportunities, not learning from an empty state (Table 29).

5.5 Changing hosts, preferences, and users

Changing the host or user profile. In 416 paired runs with learned state fixed, the selected settings and instructions match within each pair, but style matching differs by SDK. For twenty unseen profiles, hybrid-minus-frozen style differences range from -19.0 to $+4.4$ percentage points across controls. Giving all modes the same number of earlier interactions still produces no consistent hybrid advantage. These single-seed results describe the tested transfers and users (Tables 15 and 16).

Changing a user's preference. After an unannounced style change, hybrid achieves two consecutive matches within six tasks for one or two of four users per control. Frozen sometimes meets the same criterion. A post-switch match alone therefore cannot distinguish adaptation through learning from behavior that also occurs without updates (Table 18).

Switching the active user. Across 208 steps, switching users changes neither the inactive user's state nor store and introduces no instructions from that user. None of 206 checkable responses contains the other user's test marker (Appendix F.3). The test checks local separation of user information; it does not cover adversarial privacy attacks.

Testing memory directly. Separate tasks request retention, recall, or removal. Recall succeeds in $54/56$ memory-enabled cases, $0/56$ no-memory cases, and $54/56$ reference-store cases. All three pass $56/56$ checks requiring a value to be absent. These test requested operations, not learning an unstated preference (Table 19).

Appendix H separately reports configuration choices over longer histories and after a profile change. Its rankings and learning curves score selected settings, not executed responses.

5.6 Measured resource use

Median host, simulator, and extractor latencies in the update/exploration follow-up are 15.60, 7.28, and 4.88 seconds. They are component medians, not a measured total latency; extractor queue wait and setting-selection time were not recorded. Host meters report 29.90 million input and 6.05 million output tokens, with usage missing for 17 requests (Table 8). Tables 24 and 31 give training/extraction costs and the two new follow-ups, respectively. These partial measurements do not establish resource savings or a complete provider bill.

6 Discussion and Conclusion

AgentBake compares personalization methods by tracing feedback into later responses and actions under explicit task, feedback, and update conditions. Recorded user state and host input show which learned instructions reach execution.

With the shared extractor, continued updating has no clear style benefit when exploration is held fixed. With longer histories and their released learning procedures, PAHF improves style and task success, while CIPHER's style effect remains uncertain. The studies vary history length, feedback format, and implementation, so they cannot isolate which explains the earlier findings. Compared with no instruction, accurate instructions improve style but lower task success. Training improves preference detection, yet some preferences are missing from cards and some stored instructions are not delivered. The benchmark therefore checks both the information available to the host and its subsequent behavior.

The initial comparison uses four written profiles, five seeds, short histories, and constructed tasks. All users are simulated, with no human judgment ground truth; the new comparisons rely on a same-family secondary judge. Earlier adaptations omit parts of their source systems, and their execution extractor sees less history than in training. Permission and memory tests cover specified local operations, not general security. Real-user benefit, long-term preference learning, and resource savings remain unestablished.

Ethics Statement

We wrote the user profiles and constructed the controlled task facts. We collected no human annotations, and agreement between models does not establish human satisfaction. Local permission and user-switch tests are not general security or privacy audits. The conversation dataset contains hidden profiles and the simulator's private assessments. Its copy with credentials removed remains access restricted, not publicly released. Experiments with unseen simulated users have the same limits.

Reproducibility Statement

Appendix C describes the tested methods and their shared interface. Appendix D specifies the execution protocols, update/exploration controls, and statistical tests; Appendix G.4 describes the traces and simulated-user conversations. Saved manifests, code hashes, call records, and judge-input mappings allow actions, states, and statistics to be reconstructed without new model calls. Figures and tables are generated from the checked reports. The archived preference-switch driver cannot import a function from the current runner. We checked its saved curves and decisions, but did not rerun that driver. Our configuration-feedback adaptations do not use the original checkpoints or full training recipes.

AI Use Declaration

Generative AI assisted literature lookup, conceptual framing and hypothesis development, experiment design, formalizing the personalization component interface, implementation and testing, dataset preparation and credential redaction, result aggregation and interpretation, figure creation, and manuscript drafting and revision. Model-generated user reactions, teacher labels, and evaluation judgments are also part of the experimental design and are identified as such in the paper. No model judgment is described as a human annotation. The artifacts retain model identifiers, prompts, source hashes, execution records, and automated checks so that the reported computations can be inspected. The authors remain responsible for the methods, interpretation, citations, and submitted text.

References

APOLLO project. APOLLO: Code for towards preference following in tool calling language agents. GitHub repository, `https://github.com/zhiyuanc2001/APOLLO`, 2025. Repository commit `80a75350294fd5725131a77de8989a90e41dbe20`, dated November 23, 2025.

Stéphane Aroca-Ouellette, Natalie Mackraz, Barry-John Theobald, and Katherine Metcalf. Aligning LLMs by predicting preferences from user writing samples. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 1690–1721. PMLR, 2025. URL `https://proceedings.mlr.press/v267/aroca-ouellette25a.html`.

Hongru Cai, Yongqi Li, Wenjie Wang, Fengbin Zhu, Xiaoyu Shen, Wenjie Li, and Tat-Seng Chua. Large language models empowered personalized web agents. In *Proceedings of the ACM Web Conference (WWW)*, 2025. arXiv:2410.17236.

CrewAI. CrewAI: Framework for orchestrating role-playing autonomous AI agents. `https://github.com/crewAIInc/crewAI`, n.d. Official software repository. Accessed 23 September 2026.

Ge Gao, Alexey Taymanov, Eduardo Salinas, Paul Mineiro, and Dipendra Misra. Aligning LLM agents by learning latent preference from user edits. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2024. arXiv:2404.15269.

Bowen Jiang, Zhuoqun Hao, Young-Min Cho, Bryan Li, Yuan Yuan, Sihao Chen, Lyle Ungar, C. J. Taylor, and Dan Roth. Know me, respond to me: Benchmarking llms for dynamic user profiling and personalized responses at scale. *ArXiv*, abs/2504.14225, 2025.

Dian Jin, Zhi Zhang, Huichao Li, Yihe Pan, Rundong Huang, and Doudou Zhou. Personalizing large language model agents with small policy models. *arXiv preprint arXiv:2608.00215*, 2026.

LangChain. LangChain. https://github.com/langchain-ai/langchain, n.d.a. Official software repository. Accessed 23 September 2026.

LangChain. LangGraph. https://github.com/langchain-ai/langgraph, n.d.b. Official software repository. Accessed 23 September 2026.

Changmin Lee, Jaemin Kim, and Taesik Gong. From volume to value: Preference-aligned memory construction for on-device RAG. In *International Conference on Machine Learning*, 2026. URL https://icml.cc/virtual/2026/poster/62548.

S. Li, Avinandan Bose, Faeze Brahman, S. Du, Pang Wei Koh, Maryam Fazel, and Y. Tsvetkov. PrefDisco: Benchmarking proactive personalized reasoning. Preprint, 2025.

Kaiqu Liang, Julia Kruk, Shengyi Qian, Xianjun Yang, Shengjie Bi, Yuanshun Yao, Shaoliang Nie, Mingyang Zhang, Lijuan Liu, J. F. Fisac, Shuyan Zhou, and S. Hosseini. Learning personalized agents from human feedback. *ArXiv*, abs/2602.16173, 2026.

LlamaIndex. LlamaIndex. https://github.com/run-llama/llama_index, n.d. Official software repository. Accessed 23 September 2026.

Yibo Lyu, Gongwei Chen, Rui Shao, Weili Guan, and Liqiang Nie. PersonalAlign: Hierarchical implicit intent alignment for personalized gui agent with long-term user-centric records. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (ACL)*, 2026. arXiv:2601.09636.

HyunJi Nam, Yanming Wan, Mickel Liu, Peter Ahnn, Jianxun Lian, and Natasha Jaques. Learning to summarize user information for personalized reinforcement learning from human feedback. In *International Conference on Learning Representations*, 2026. URL https://iclr.cc/virtual/2026/poster/10010999.

Pydantic. PydanticAI. https://github.com/pydantic/pydantic-ai, n.d. Official software repository. Accessed 23 September 2026.

Anka Reuel, Amelia Hardy, Chandler Smith, Max Lamparth, Malcolm Hardy, and Mykel J. Kochenderfer. BetterBench: Assessing AI benchmarks, uncovering issues, and establishing best practices. In *Advances in Neural Information Processing Systems*, volume 37, pp. 21763–21813, 2024. URL https://proceedings.neurips.cc/paper_files/paper/2024/hash/26889e8359e7ef8a7f5d77457364ca55-Abstract-Datasets_and_Benchmarks_Track.html.

Alireza Salemi, Sheshera Mysore, Michael Bendersky, and Hamed Zamani. LaMP: When large language models meet personalization. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (ACL)*, 2024. arXiv:2304.11406.

Anikait Singh, Sheryl Hsu, Kyle Hsu, Eric Mitchell, Stefano Ermon, Tatsunori Hashimoto, Archit Sharma, and Chelsea Finn. FSPO: Few-shot optimization of synthetic preferences effectively personalizes to real users. In *International Conference on Learning Representations*, 2026. URL https://www.iclr.cc/virtual/2026/poster/10009369.

Strands Agents. Strands Agents SDK. https://strandsagents.com/, n.d. Official documentation. Accessed 23 September 2026.

Ekaterina Taktasheva and Jeff Dalton. TAPS: Tool-augmented personalisation via structured tagging. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 23519–23544. Association for Computational Linguistics, 2025. doi: 10.18653/v1/2025.emnlp-main.1200.

Di Wu, Hongwei Wang, Wenhao Yu, Yuwei Zhang, Kai-Wei Chang, and Dong Yu. LongMemEval: Benchmarking chat assistants on long-term interactive memory. In *International Conference on Learning Representations*, 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/file/d813d324dbf0598bbdc9c8e79740ed01-Paper-Conference.pdf.

Qingyun Wu, Gagan Bansal, Jieyu Zhang, Yiran Wu, Beibin Li, E. Zhu, Li Jiang, Xiaoyun Zhang, Shaokun Zhang, Jiale Liu, A. Awadallah, Ryen W. White, D. Burger, and Chi Wang. AutoGen: Enabling next-gen LLM applications via multi-agent conversation. Preprint, 2023.

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. OSWorld: Benchmarking multimodal agents for open-ended tasks in real computer environments. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2024. arXiv:2404.07972.

Qiancheng Xu, Yongqi Li, Heming Xia, Fan Liu, Min Yang, and Wenjie Li. PEToolLLM: Towards personalized tool learning in large language models. *arXiv preprint arXiv:2502.18980*, 2025.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. ReAct: Synergizing reasoning and acting in language models. In *International Conference on Learning Representations (ICLR)*, 2023. arXiv:2210.03629.

Shunyu Yao, Noah Shinn, Pedram Razavi, and Karthik Narasimhan. τ-bench: A benchmark for tool-agent-user interaction in real-world domains. *ArXiv*, abs/2406.12045, 2024.

Jiaqi Zhang, Tong Chen, Junliang Yu, Quoc Viet Hung Nguyen, and Hongzhi Yin. ODYSSE: Episode-wise policy optimization for personalized agentic reasoning. arXiv preprint arXiv:2607.25369, 2026a. URL https://arxiv.org/abs/2607.25369.

Jiaqi Zhang, Chen Gao, Liyuan Zhang, Quoc Viet Hung Nguyen, and Hongzhi Yin. SmartAgent: Chain-of-user-thought for embodied personalized agent in cyber world. *Proceedings of the AAAI Conference on Artificial Intelligence*, 40(21):17993–18001, 2026b.

Ranxu Zhang, Zeyang Li, Jiacheng Huang, Rui Zhang, Xiaozhou Xu, Zhe Sun, Yanyong Zhang, and Chao Wang. From correctness to preference: A framework for personalized agentic reinforcement learning. *arXiv preprint arXiv:2605.23382*, 2026c.

Siyan Zhao, Mingyi Hong, Yang Liu, Devamanyu Hazarika, and Kaixiang Lin. Do llms recognize your preferences? evaluating personalized preference following in llms. *ArXiv*, abs/2502.09597, 2025.

Shuyan Zhou, Frank F. Xu, Hao Zhu, Xuhui Zhou, Robert Lo, Abishek Sridhar, Xianyi Cheng, Tianyue Ou, Yonatan Bisk, Daniel Fried, Uri Alon, and Graham Neubig. WebArena: A realistic web environment for building autonomous agents. In *International Conference on Learning Representations (ICLR)*, 2024. arXiv:2307.13854.

Yuxuan Zhu, Tengjun Jin, Yada Pruksachatkun, Andy Zhang, Shu Liu, Sasha Cui, Sayash Kapoor, Shayne Longpre, Kevin Meng, Rebecca Weiss, Fazl Barez, Rahul Gupta, Jwala Dhamala, Jacob Merizian, Mario Giulianelli, Harry Coppock, Cozmin Ududec, Antony Kellermann, Jasjeet Sekhon, Jacob Steinhardt, Sarah Schwettmann, Arvind Narayanan, Matei A. Zaharia, Ion Stoica, Percy Liang, and Daniel Kang. Establishing best practices in building rigorous agentic benchmarks. In *Advances in Neural Information Processing Systems*, volume 38, 2025. URL https://proceedings.neurips.cc/paper_files/paper/2025/hash/f316275b44ee2de533102913828a8107-Abstract-Datasets_and_Benchmarks_Track.html.

A A Worked Example of Preference Reuse

In a saved calculation task (Figure 3), the host must look up a record for three sites, calculate total units and remaining capacity, identify the largest site, and submit the result. The task specifies an alphabetical tie-break and forbids external actions. It does not disclose the user's preference for concise answers. The host submits the correct values, 164, 100, and site B, but follows them with a 323-word explanation.

The simulated user accepts the calculation and says, "next time just lead with the submitted values and a one-liner on anything non-obvious like the tie-break." The extractor produces the instruction "Lead with submitted values and a one-liner on anything non-obvious; avoid verbose explanations". The component stores it for this user, and the next task's saved host instructions confirm that it was delivered. Neither step gives the component the hidden profile.

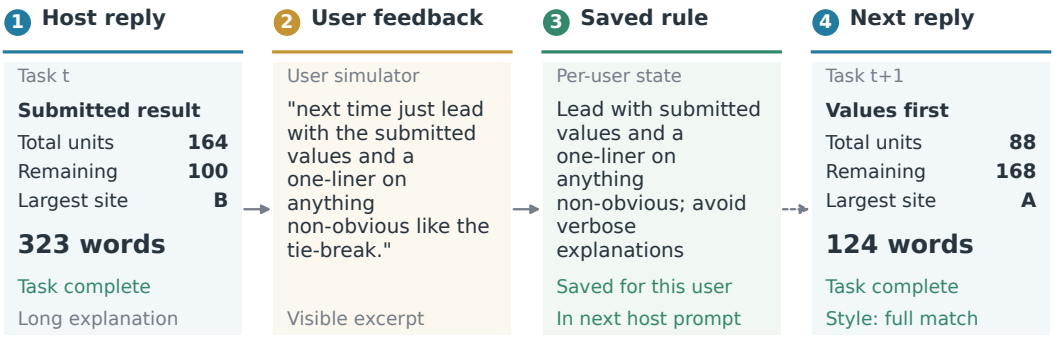

Figure 3: A saved sequence in which an accepted instruction reaches a later task. The dashed arrow marks rule reuse on that different task. Both independent judges rate the later style as a full match.

On that next task, the record contains 50, 12, and 26 units with capacities 86, 77, and 93. The host submits 88 total units, 168 remaining capacity, and site A. Its reply begins with those values and contains 124 words. The task checker confirms completion, and both independent judges label the later style a full match. This sequence is not an estimate of the update's effect. Another saved sequence delivers a concise-style instruction but receives only a partial style match. Figure 4 shows both sequences, including their selection rule. The matched comparisons in Section 5.2 test whether such improvements hold across runs.

B Checks of Simulated Feedback and Evaluation

Does the feedback reflect the intended users? We check whether the four profiles produce distinguishable comments. A classifier using only visible user messages identifies the profile with 0.846 balanced accuracy, compared with 0.25 chance. Detailed-user messages are 2.54 times as long as concise-user messages, consistent with their different answer-length preferences. These checks show differences between simulated users, not that every comment expresses the intended preference or represents human behavior. During collection, a targeted consistency check regenerates 19.23% of concise-user turns after unmistakably overlong, heavily structured answers. Other conditions have no such repair. Table 21 reports this filtering and the remaining collection checks.

Can the evaluation distinguish task correctness from preference matching? In the earlier execution studies, the host and simulator use separate Sonnet 4.6 calls. We use GPT-6 Astra as the primary evaluator to avoid relying solely on the model family that generates the interactions. Opus 5 supplies a second set of judgments to test whether conclusions depend on the judge. Neither sees method names or simulator reasoning. Constructed cases then test specific expectations: preference judgments should respond to a changed profile, whereas task judgments should not; reversing response order should preserve the intended comparison. Astra and Opus satisfy $4/4$ and $3/4$ profile-swap checks, respectively, and both satisfy $6/6$ response-order checks. Profile substitutions leave the task-judge input byte-identical, and both judges pass $4/4$ task/preference separation checks. Ap-

pendix D.4 gives the cases and results. These checks provide known test expectations, not human annotations.

Which preferences can be judged? A reply can reveal its style without showing whether the host retained information in memory. We record host failures, missing judgments, and the available evidence for each preference; we do not count missing evidence as a failure. Results report evidence coverage and judge agreement. Separate memory-operation and user-switch tests check behaviors that ordinary replies cannot establish. These audits follow benchmark-validity work (Reuel et al., 2024; Zhu et al., 2025).

C Interface and Method Adaptations

This appendix defines the interface from Section 3.2 and the state and update rules of the methods compared in Section 4.2.

Algorithm 1 AgentBake: execution and optional user-state updates

Require: Personalization component π, host H, adapter operation C, user state S_u, update flag U, exploration flag E

for each request x_t from user u **do**
  Load only S_u and host-supported controls $\mathcal{A}_t$
  $a_t \leftarrow \pi.\mathrm{select}(x_t, S_u, \mathcal{A}_t; E)$
  $I_t \leftarrow \mathrm{retrieve}(S_u, x_t)$ ▷ Compact preference instructions
  $p_t \leftarrow C(x_t, a_t, I_t)$ ▷ Host input with permitted settings
  $(y_t, \tau_t) \leftarrow H(p_t)$ ▷ Host response and execution trace
  Record a_t, p_t, y_t, τ_t, failures, tokens, and latency
  $z_t \leftarrow \mathrm{extract}(x_t, y_t, \text{visible user reaction})$
  if $U = 1$ and π supports learning **then**
    Append valid feedback to S_u; retain eligible persistent instructions
    Apply method-specific numerical updates to eligible preference or task feedback
  end if
  Save S_u; leave other users' states unchanged
end for

C.1 Inputs, actions, and feedback

For request X_t, visible history H_{t-1}, and user state z_{t-1}, the component selects an action and the host produces an output:

$$\begin{aligned} a_t &= \pi.\textsc{Select}(X_t, H_{t-1}; z_{t-1}), \\ O_t &= \mathcal{G}_{\mathrm{host}}(X_t, H_{t-1}, \kappa_{\mathrm{host}}(a_t), \xi_t)\,. \end{aligned} \tag{1}$$

Here z_{t-1} denotes the active user's state, called S_u in Algorithm 1. The adapter maps the selected memory, tool, and style settings to prompt instructions or supported SDK configuration through κ_{host}. ξ_t denotes randomness in the host's execution at turn t, such as variation in response generation or stochastic tool outputs. It is separate from the component's exploration, which controls how a_t is selected (Appendix D.2). Feedback $y_t \in \mathcal{Y} \cup \{\perp\}$ can be a rating, edit, task result, or absent observation:

$$\begin{aligned} z_t &= \pi.\textsc{Update}(z_{t-1}; X_t, H_{t-1}, a_t, O_t, y_t), \\ H_t &= \mathrm{Append}(H_{t-1}, (X_t, a_t, O_t, y_t))\,. \end{aligned} \tag{2}$$

For a frozen component, Update leaves state unchanged. Explicit preferences can enter z_0 or H_0.

Each implementation provides `init_user`, `select_action`, and `update`. Component code imports no agent SDK and does not call the host. In the controlled integrations, memory mode also gates stored instructions; `no_memory` removes them from the prompt.

Cards are exported as per-user JSON files. Each rule contains its instruction, confidence, support count, and source identifiers, grouped under global or task-specific scope. The runtime rebuilds cards from accepted feedback and renders relevant rules within the instruction budget and memory permissions. These preference records are separate from the SQLite memory used by host tools.

C.2 COMPARED LEARNING MECHANISMS

Table 4 identifies which settings each of the thirteen implementations learns to score and whether it uses exploration. The twelve components also share a store of accepted text instructions. Thus a tool-only numerical learner can retain a style instruction, and exploration can vary settings outside the channels whose scores it learns. In compact result tables, PersonalAlign denotes PersonalAlign/HIM.

Table 4: AgentBake implementations of eleven source methods, the FABLE-Diag diagnostic variant, and a non-learning host control. M/T/S mark numerical updates to memory/tool/style scores. All twelve components can also store accepted text instructions; the update switch controls both. E indicates whether a method uses the exploration argument, not whether its action changes on every call. These implementations do not use the source checkpoints or reproduce their full training procedures.

Method	Numeric	Uses E	Implemented mechanism and scope
Host default	–	No	Fixed host utility; no learned user state.
FABLE-Diag	M, T, S	Yes	Diagonal Bayesian residual model with promotion pseudo-observations.
FABLE	M, T, S	Yes	Full-rank residual model; unit sampling scale and no context writeback.
PUMA	M, T	Yes	Retrieved feedback and compact action ranking, not the source language-model training recipe.
PETool LLaMA	T	Yes	Compact tool logits with preference ranking, not a tool-generating LoRA.
APOLLO	T	Yes	Online compact policy gradient, not the source VERL model training recipe.
PersonalAlign/ HIM	M	No	Structured task vectors and memory-mode clusters, not GUI intent extraction.
SmartAgent	M, T, S	Yes	Compact evidence, intent, and action scores, not a trained GUI model.
ODYSSE	M, T, S	Yes	Streaming softmax updates, not grouped multimodal model rollouts.
PARPO	M, T, S	Yes	Compact dual-reward updates and user anchor, not token-level PPO.
PAHF	M, T, S	No	Scalar component notes; no original textual human-feedback loop.
CIPHER	M, T, S	No	Scalar-derived action contrasts; no literal user-edit preference inference.
TAPS	M, T, S	No	Structured numeric tags; no original preference-tagging prompts or demonstrations.

Residual preference models. FABLE-Diag maintains a diagonal Gaussian posterior over factorized per-user preferences and uses Thompson sampling. It can initialize from onboarding pseudo-observations and promotes sufficiently certain preference directions through additional pseudo-observations. FABLE maintains a full-rank posterior, uses outer-product onboarding observations and residual scalar feedback, and omits source context writeback (Jin et al., 2026). Their earlier comparison also differs in promotion updates, so it does not isolate posterior structure.

Memory and tool specialists. PUMA retrieves relevant interactions, forms preferred/dispreferred action pairs from scalar feedback, and learns a compact memory/tool ranker; it does not train the source Llama-2 function-and-argument generator (Cai et al., 2025). PEToolLLaMA updates tool logits with SFT- and DPO-like losses instead of training the source Llama-3 LoRA and argument generator (Xu et al., 2025). APOLLO uses history, a running reward baseline, and action-sequence policy gradients; it does not reproduce Qwen3-8B training with VERL (APOLLO project, 2025). PersonalAlign organizes preferences and routines hierarchically and learns memory settings from structured task vectors, replacing the source Android screenshots, coordinates, and embeddings (Lyu et al., 2026).

Intent and reward learning. SmartAgent retains search/action, intent inference, and personalized decision stages using compact supervised state rather than a trained GUI model (Zhang et al., 2026b). ODYSSE retains stage-local rewards, two intent gates, episode-wide credit, and local plus episodic advantages, but uses incoming samples instead of grouped multimodal rollouts (Zhang et al., 2026a). PARPO keeps generic and user-specific rewards separate. Per-user running means and variances define its personalized anchor and normalized advantage; compact updates replace token-level optimization (Zhang et al., 2026c).

Notes, comparisons, and tags. PAHF maintains global and task-specific notes for each setting, initialized through clarification or onboarding and updated from scalar feedback; it does not reproduce the original textual human-feedback loop (Liang et al., 2026). CIPHER retrieves action comparisons inferred from scalar feedback, not literal user-edited responses (Gao et al., 2024). TAPS stores preference tags and learns when they should affect tool choices; deterministic tags replace the source prompts and DSPy demonstrations (Taktasheva & Dalton, 2025). The host control maximizes a fixed rule score minus cost and never stores or updates user state.

D Execution Protocols and Statistical Tests

These protocols cover the three learning modes (Section 4.5), the update/exploration follow-up (Section 5.2), and transfer and user changes (Section 5.5). The earlier comparisons described here use a fixed extractor and independent judgments. Protocols, task schedules, code hashes, and call identifiers were fixed before the corresponding model calls, but were not externally preregistered. We collected no human annotations. Table 6 distinguishes the controlled method tests from instruction comparisons on original applications. Appendix I describes the additional instruction and learning tests in Section 5.4.

Table 5: The seven controlled SDK integrations compare all personalization methods. The original implementations supply conversations and separate instruction-representation tests, not all-method comparisons. The configuration-choice study appears in Appendix H.1, with its protocol in Appendix H.

Study	Data or execution setting	What is measured
Simulated-user conversations	100 implementations; 100 tasks; 5 conditions	Visible feedback without an active personalization component
Matched method comparison	7 controlled SDK integrations; 13 controls	Later task completion and preference matching under matched histories
Instruction representations	100 restricted original applications; 2 held-out tasks; 4 users	Direct instructions versus cards, with actual host execution
Transfer and user changes	Controlled SDK integrations; fixed and unseen profiles	Cross-host behavior, preference changes, isolation, and explicit memory operations

Table 6: Execution coverage and outcomes. Impls./SDKs count implementation records and entry points, not independent architectures. Original apps compare instruction representations (repr.), not personalization methods. Eval./All count calls excluding/including the learning tasks. Ret. and Task count successful host returns and completed tasks over All. Judge errors count failed unique inputs in primary/sensitivity order; one failure can affect several calls. A dash denotes no execution-seed factor. The 672-call memory study reports its outcomes separately.

Study	Impls./ SDKs	Controls	Users	Seeds	Eval./All	Ret.	Task	Judge errors
Initial five-seed comparison	7/7	13	4	5	6,240/8,320	8,303	8,143	0/57
Card delivery removed	3/3	13	4	1	1,248/1,248	1,247	1,240	0/5
Style switch	3/3	13	4	1	1,872/1,872	1,871	1,867	0/5
Fixed-user SDK transfer	3/3	13	4	1	1,248/1,248	1,247	1,229	0/19
User switch	1/1	13	2	1	208/208	206	202	0/1
Original apps: representations	100/7	5 repr.	4	–	4,000/4,000	3,390	269	0/52
Original apps: diagnostics	100/7	2 repr.	4	–	1,600/1,600	1,367	101	0/23
Unseen users: equal adaptation	7/7	13	20	1	6,240/6,240	6,237	6,092	0/26
Unseen-user SDK transfer	7/7	13	20	1	6,240/8,320	8,303	7,957	1/50
Updates by exploration	7/7	13	4	5	8,320/8,320	8,307	8,085	0/35

D.1 Inputs, hosts, and feedback

The seven controlled integrations use AutoGen, CrewAI, LangChain, LangGraph, LlamaIndex, PydanticAI, and Strands. They share local records and calculation tasks: compute total units, remaining capacity, and the largest site, then submit the fields through a specified tool. SQLite records submissions and state changes. A deterministic checker decides task completion; a separate judge assesses the response. These tasks do not recreate the original applications. LangChain's entry point uses LangGraph, so the two are not independent architectures.

The action set combines three memory modes, four tool modes, and ten style or clarification modes before task-specific filtering. Required lookup remains available; tool mode controls optional lookup. Memory mode controls reads, writes, and retrieval of preference instructions. Permissions are assigned outside model arguments. Task-type labels come from the public task generator, not the profile.

The host and simulator use separate Sonnet 4.6 calls; only the simulator receives the hidden profile. Each controlled call has a 300-second limit, at most twelve model requests, and at most $2{,}048$ output tokens per request.

The earlier comparisons described here use the extractor checkpoint trained with seed 2026. It sees the current request, response, and user feedback, but no earlier conversation, identity metadata, structured tool trace, private assessment, or judge score. This differs from training, where $5{,}994/8{,}000$ labeled examples include earlier visible turns.

Numerical preference updates require a valid, high-confidence positive or negative preference label. Task-quality corrections do not become preference rewards. Mixed feedback may support a stored instruction without supporting a numerical update. Methods that use task rewards receive the local checker's outcome, not the simulator's private rating or preference-judge score. Earlier conversations provide extracted instructions, not rewards for actions the component did not take.

D.2 Learning modes and follow-up controls

Three-mode comparison. Each method starts with the user's preference records extracted from historical conversations. These records supply instructions but do not update numerical action preferences. The method then runs eight learning tasks, receiving feedback on its own choices, followed by eight new tasks under frozen, online-from-scratch, and hybrid modes. Frozen and hybrid share the resulting parameters and preference records; online resets both, including the historical records. Frozen also disables exploration, while the adaptive modes retain it. This first comparison therefore changes both updating and exploration.

Five seeds (2026–2030) vary component execution, not extractor training. Methods share assignments of users, seeds, and SDKs. Each of the twenty user/seed pairs runs in one SDK, not all seven.

Follow-up separating updates and exploration. We restore the state after the same learning tasks, including scores, stored instructions, random-generator state, and SQLite data. Saved choices and updates reconstruct the state and reproduce its hash without new model calls or independent judge feedback. We combine updates off/on ($U0/U1$) with exploration off/on ($E = 0/1$) on eight further tasks: $13 \times 20 \times 4 \times 8 = 8{,}320$ calls. Task identifiers and normalized queries do not overlap earlier tasks. Call order is fixed before outcomes. U controls both numerical learning and storage of new instructions. The host model's weights remain fixed in all four conditions.

Both update conditions generate and log user reactions and extractor outputs. Under $U = 0$, the feedback-processing code validates and records these outputs but does not append new feedback to user state or call the component's numerical update. Under $U = 1$, successful executions can add valid feedback records. Only eligible persistent instructions are later retrieved; numerical updates require the preference checks above.

How exploration is implemented. The runner passes E to each implementation's `select_action` function as `explore=False/True`. Each implementation first filters the memory/tool/style combinations by the task's hard constraints. With $E = 0$, it selects the highest-scoring allowed combination using its current state, without sampling. With $E = 1$, it uses the following method-specific rule:

- **Sample preference parameters.** FABLE draws one preference vector from its full-covariance Gaussian posterior; FABLE-Diag draws from its diagonal posterior. They then choose the highest-scoring allowed action under that draw. With $E = 0$, they use the posterior mean instead. FABLE's sampling scale is 1.0; FABLE-Diag uses the posterior standard deviations without additional scaling.
- **Add noise to action scores.** PUMA and SmartAgent add independent Gumbel draws to each allowed action's score and select the largest resulting score. The draws have location zero and scales 0.08 and 0.10, respectively. With $E = 0$, neither adds noise.
- **Sample actions according to their scores.** PEToolLLaMA and APOLLO apply softmax to the allowed action scores with temperature 0.22; ODYSSE uses 0.28 and PARPO uses 0.25. Each samples one action from those probabilities. With $E = 0$, it takes the largest score instead. These temperatures apply to the component's action scores, not to the host language model.
- **Keep the same selection rule.** PersonalAlign/HIM, PAHF, CIPHER, TAPS, and the host control ignore the exploration argument. Given identical input and state, changing E does not change their selected action.

E neither bypasses hard constraints nor changes model-call limits, decoding settings, or access to judge feedback. $U = 0$ freezes learned numerical state and stored instructions; $U = 1$ permits both to change after eligible feedback, except for the non-learning host control. Thus $U0E1$ can try different actions without learning, whereas $U1E0$ can change its preferred action as feedback changes its scores. Even without updates, a different task can change the highest-scoring action.

The matched component random-generator state concerns action selection; it does not guarantee identical host responses or simulator feedback across separate calls. Host-side randomness is represented by ξ_t in Equation 1. Holding E fixed therefore tests updating under one method's selection rule. Enabling E does not impose an equal exploration strength across methods.

D.3 Instruction, transfer, and user-change tests

Original applications. Two held-out tasks for each of 100 implementations and four users give 800 calls per representation. The five main conditions are no learned state, direct extracted instructions, a personal card, a task-only-trained card, and another user's card. Direct instructions and cards use the same predictions and a 512-token ceiling, but their actual lengths differ. The supplement adds full direct instructions and five explicit profile instructions as a privileged reference. It reuses the 800 unchanged card calls and adds $1{,}600$ new calls. Tasks share neither base scenarios nor normalized queries with instruction-building data.

Each adapter runs in a new namespace with read-only code, installed SDKs, bundled data, scratch storage, and one metered model connection. It has no original working directory, external network access, or forwarded service credentials. Limits are 360 seconds and 24 metered requests. Failures remain in scheduled totals; these restricted applications' task scores cannot be compared directly with controlled calculations.

Transfer and instruction removal. For twenty profiles excluded from extractor training, each method learns for eight tasks in one SDK, then tests three modes for eight tasks in the next SDK in a fixed ring. No instructions from historical conversations are supplied. Fixed-user supplements either remove stored instructions while keeping numerical state or change SDK with matched task content. The SDK-change test also inherits historical instructions from several agents, so it does not isolate knowledge learned from a single source agent. An equal-count supplement gives all modes eight earlier interactions: source-SDK learning for frozen/hybrid and target-SDK learning for online. Their counts match, but their history content does not. These supplements use one execution seed.

Preference changes and user separation. The style-switch study pairs unchanged and changed styles, leaving the other four preferences and constraints fixed. The component receives no switch notification. Recovery is the first of two consecutive style matches within six tasks; unobservable judgments break the sequence, and non-recovery is censored at that horizon. The user-switch study compares isolated and interleaved processing with identical initial state and each user's task order preserved. Checks cover inactive-user state, random-generator state, pending actions, memory contents, instruction sources, and non-sensitive response markers. Hosts restart each task; this is not an adversarial privacy test.

Explicit memory operations. A separate 672-call study compares memory-enabled, no-memory, and privileged reference stores across seven SDKs. Constructed tasks request retention, recall after reset, update, forgetting, non-retention, and user switching. Reads, writes, deletions, submitted answers, final contents, and permissions determine outcomes. These tests establish operations, not learning of unstated memory preferences.

D.4 Judges and measurement checks

GPT-6 Astra is the primary judge; Opus 5 tests sensitivity. Judges receive the request, response, and relevant trace, without method names, condition labels, or simulator reasoning. Only preference judgments receive the target profile. Constructed cases, profile substitution, and swapped response order test known expectations: a changed profile should change the intended preference judgment, task judgments should not depend on the profile, and response order should not alter the comparison. Table 7 reports these checks. They provide no human ground truth. Each distinct judge input is attempted once; failed judgments remain missing. The deterministic checker overrides response judgments for controlled task completion.

Table 7: Judge checks against constructed, rule-derived expected outcomes. Cells are correct/completed cases, including failed judge calls in the denominator. These are diagnostic cases, not human labels or an estimate of real-user agreement. Profile substitutions also leave the task-judge input byte-identical.

Check	GPT-6 Astra	Opus 5
clarification	2/2	2/2
memory evidence	3/3	2/3
missing evidence	2/2	2/2
profile swap	4/4	3/4
response order	6/6	6/6
task correctness	9/9	8/9
task preference separation	4/4	4/4
task tool evidence	2/2	2/2
tone	2/2	2/2
tool evidence	2/2	2/2

D.5 Outcome definitions and statistical tests

Only full match, partial match, and contradiction enter each preference's observable denominator; only full match is success. Insufficient evidence, inapplicability, and missing judgments remain distinct. Joint success requires task completion, every applicable preference, and observed permission compliance. An observed failure rules it out; otherwise missing evidence leaves it unknown. Bounds assign unknown binary outcomes both possible values over all scheduled calls. They are not confidence intervals.

The initial family of planned comparisons contains 28 task/style contrasts: card versus direct instructions and hybrid versus frozen for every control. Original-application intervals resample normalized source repositories; method comparisons use five paired seed means. The update/exploration follow-up contains 78 comparisons per judge: two outcomes, thirteen controls, and three effects,

$$\Delta_0 = U1E0 - U0E0, \qquad \Delta_1 = U1E1 - U0E1, \qquad I = (U1E1 - U0E1) - (U1E0 - U0E0).$$

Paired task differences are averaged within each seed, then across five seeds using Student-t intervals. The interaction uses tasks observable in all four combinations and ranges over $[-2, 2]$. Pointwise intervals have nominal 95% coverage; simultaneous intervals use Bonferroni adjustment over the respective family. The estimates concern fixed designed profiles and SDK assignments, not a population of users.

PAHF's task effect at $E = 0$ is -0.03125 in every seed ($150/160$ versus $155/160$ tasks), so its registered t interval has zero width. This is not certain harm. A post hoc enumeration of all 2^5 sign assignments gives a minimum two-sided $p = 0.0625$; no comparison passes the adjusted threshold. The check assumes independent seed differences with exchangeable signs, not verified through a randomized design, and changes no planned estimate. Shared-output judge rankings are descriptive, use average ranks for ties, and leave correlation undefined for constant ranks.

D.6 Failed calls

The follow-up schedule returns $8{,}307$ successful calls and 13 host errors, including ten route rate-limit/cooldown errors. None is retried. Each judge has $14{,}561$ unique inputs: Astra returns all; Opus returns $14{,}526$ and records 35 errors. One input can map to several executions. Earlier schedules include four unrecoverable infrastructure interruptions and sixteen issued judge requests without saved output. Interrupted host calls and issued judge requests were not repeated. Table 6 summarizes coverage; the source reports retain the full failure classifications.

D.7 Runtime and judge accounting

Table 8 reports resource use by the host, simulator, and extractor; Table 9 reports it separately for the two judges. Per-method and earlier-study records are in `supplementary_results/`. Instruction tokens are already included in host prompts. Extractor time excludes queue wait; configuration selection and card construction were not timed. Missing usage is not zero, and component medians cannot be added to obtain end-to-end latency. Provider retries, cache counters, and CLI cost estimates do not constitute a complete invoice.

Table 8: Observed resources in the four-setting study, read directly from the complete component report. Counts are host calls, simulator records, and extractor predictions, respectively. Input/output are observed tokens; latency is the component's median in seconds, not a mean of method medians. Missing counts requests without usage; n/a means no request-usage counter. Host counts include failed executions. Extractor queue wait is excluded.

Component	Count	Input	Output	Median s	Missing
Host	8320	29904976	6046060	15.60	17
Simulator	8307	17606630	1202825	7.28	6
Extractor	8305	6970617	853028	4.88	*n/a*

Table 9: Judge accounting, one attempt per unique input (14,561 inputs per judge). Failed cases remain missing. In/Out are each CLI's raw input/output token fields and are not comparable totals across backends: Astra separately reports 128,763,265 cache-write and 813,893 reasoning tokens; Opus reports 38,842,186 cache-creation and 29,073,462 cache-read tokens. Cache counters are not added here. Latency covers successful attempts. CLI USD is a reported estimate, not a verified invoice; missing cost is not zero.

Judge	Success	Errors	Raw in	Raw out	Median s	CLI USD
Astra	14561	0	128792387	4249320	7.20	*n/a*
Opus	14526	35	44354	19195771	16.57	946.97

E Full Learning-Mode Comparisons

The full three-mode and update/exploration comparisons (Sections 5.1 and 5.2) include evidence coverage and judge agreement. We then follow accepted preferences into later host instructions (Section 5.3).

The CSV/JSON files in `supplementary_results/` report both interval types, observed counts, missing-outcome bounds, outcomes by setting, learning and later use, judge rankings, and resources.

E.1 Primary three-mode contrasts

For each method, the primary contrast compares hybrid with offline-frozen learning. Table 10 also includes the card-versus-direct-instruction contrast on original applications, which belongs to the same planned comparison family. That contrast concerns instruction representation, not the learning mode; Appendix F gives its execution setting and supporting figure. Appendix D.5 specifies the uncertainty procedure for each study.

Table 10: Registered primary contrasts in percentage points. The original-application comparison is card minus direct extracted instructions; each method comparison is hybrid minus frozen. Original-application intervals resample source-agent clusters; method comparisons use paired means across five seeds. Simultaneous intervals use the registered Bonferroni family of 28 contrasts. Paired counts give jointly observed/all scheduled units. Bounds for missing outcomes are in the accompanying data.

Comparison	Outcome	Paired n/N	Δ pp	95%	Family 95%
Card vs. events	Task	800/800	-0.6	[-1.7, 0.4]	[-2.4, 1.0]
Card vs. events	Style	664/800	-22.6	[-26.9, -18.3]	[-29.5, -15.6]
Host default	Task	160/160	-1.2	[-4.7, 2.2]	[-10.5, 8.0]
Host default	Style	160/160	-0.6	[-18.5, 17.2]	[-48.2, 46.9]
FABLE-Diag	Task	160/160	1.9	[-0.3, 4.0]	[-3.8, 7.5]
FABLE-Diag	Style	160/160	-5.6	[-18.3, 7.1]	[-39.4, 28.2]
FABLE	Task	160/160	0.0	[-2.7, 2.7]	[-7.3, 7.3]
FABLE	Style	159/160	-10.7	[-16.5, -4.8]	[-26.2, 4.9]
PUMA	Task	160/160	-0.6	[-2.4, 1.1]	[-5.2, 4.0]
PUMA	Style	160/160	-29.4	[-44.6, -14.2]	[-69.8, 11.0]
PEToolLLaMA	Task	160/160	0.6	[-1.1, 2.4]	[-4.0, 5.2]
PEToolLLaMA	Style	159/160	-16.4	[-21.0, -11.8]	[-28.7, -4.1]
APOLLO	Task	160/160	-1.9	[-5.3, 1.6]	[-11.1, 7.4]
APOLLO	Style	160/160	-21.9	[-32.8, -10.9]	[-51.1, 7.3]
PersonalAlign	Task	160/160	-1.2	[-5.7, 3.2]	[-13.0, 10.5]
PersonalAlign	Style	159/160	-3.8	[-9.0, 1.4]	[-17.6, 10.0]
SmartAgent	Task	160/160	1.2	[-0.9, 3.4]	[-4.4, 6.9]
SmartAgent	Style	159/160	-1.8	[-10.9, 7.3]	[-26.0, 22.4]
ODYSSE	Task	160/160	0.6	[-1.1, 2.4]	[-4.0, 5.2]
ODYSSE	Style	160/160	-5.0	[-19.4, 9.4]	[-43.4, 33.4]
PARPO	Task	160/160	-0.6	[-2.4, 1.1]	[-5.2, 4.0]
PARPO	Style	160/160	-8.1	[-18.6, 2.4]	[-36.0, 19.8]
PAHF	Task	160/160	-0.6	[-3.9, 2.6]	[-9.3, 8.0]
PAHF	Style	159/160	0.6	[-2.6, 3.9]	[-8.0, 9.3]
CIPHER	Task	160/160	0.0	[-4.8, 4.8]	[-12.7, 12.7]
CIPHER	Style	160/160	-4.4	[-10.9, 2.1]	[-21.7, 12.9]

Table 10 (continued)

Comparison	Outcome	Paired n/N	Δ pp	95%	Family 95%
TAPS	Task	160/160	1.2	[-3.2, 5.7]	[-10.5, 13.0]
TAPS	Style	160/160	-1.2	[-9.3, 6.8]	[-22.7, 20.2]

E.2 Updating at fixed exploration settings

Table 11 puts task and style effects beside one another for all thirteen controls. Δ_0, Δ_1, and I follow Appendix D.5; a missing judgment from one model is not replaced by the other model's judgment.

Table 11: All planned task and style effects, in percentage points. Cells give mean differences and simultaneous 95% intervals adjusted over 78 task/style comparisons per judge. Task effects use the same deterministic checker; style uses each judge's observed pairs. Δ_0/Δ_1 compare updates at $E = 0/1$; I uses tasks judged in all four settings. Pointwise intervals, observed counts, and missing-outcome bounds are in the accompanying CSV/JSON records. PAHF's zero-width task interval does not prove certain harm (Appendix D.5).

Method	Effect	Task	Style (Astra)	Style (Opus)
Host default	Δ_0	-1.2 [-21.7, 19.2]	-3.1 [-45.8, 39.6]	0.0 [-13.7, 13.7]
Host default	Δ_1	1.9 [-16.2, 20.0]	4.4 [-16.1, 24.9]	0.6 [-10.8, 12.1]
Host default	I	3.1 [-6.4, 12.7]	7.5 [-18.8, 33.8]	0.7 [-22.4, 23.7]
FABLE-Diag	Δ_0	-1.2 [-8.6, 6.1]	1.2 [-46.1, 48.6]	-0.0 [-9.9, 9.8]
FABLE-Diag	Δ_1	-0.6 [-6.7, 5.4]	-2.5 [-37.5, 32.5]	0.6 [-5.6, 6.9]
FABLE-Diag	I	0.6 [-5.4, 6.7]	-3.8 [-72.2, 64.7]	0.0 [-9.8, 9.9]
FABLE	Δ_0	0.6 [-5.4, 6.7]	-1.2 [-33.8, 31.3]	1.2 [-19.2, 21.7]
FABLE	Δ_1	1.2 [-10.8, 13.3]	5.0 [-27.5, 37.5]	5.1 [-15.4, 25.6]
FABLE	I	0.6 [-14.2, 15.4]	6.2 [-39.6, 52.1]	3.8 [-26.2, 33.8]
PUMA	Δ_0	0.0 [-9.6, 9.6]	-3.1 [-19.7, 13.4]	-1.9 [-14.1, 10.4]
PUMA	Δ_1	-1.2 [-13.3, 10.8]	-0.6 [-22.8, 21.6]	1.9 [-10.2, 14.0]
PUMA	I	-1.2 [-21.7, 19.2]	2.5 [-28.3, 33.3]	3.7 [-16.5, 23.9]
PEToolLLaMA	Δ_0	-1.9 [-9.3, 5.5]	-0.6 [-62.8, 61.6]	3.8 [-13.9, 21.4]
PEToolLLaMA	Δ_1	-0.6 [-6.7, 5.4]	3.8 [-42.4, 50.0]	-1.2 [-8.6, 6.1]
PEToolLLaMA	I	1.2 [-6.1, 8.6]	4.2 [-78.2, 86.7]	-5.0 [-27.6, 17.6]
APOLLO	Δ_0	-1.9 [-14.0, 10.2]	0.0 [-31.8, 31.9]	0.0 [-19.5, 19.6]
APOLLO	Δ_1	1.2 [-6.1, 8.6]	-1.2 [-41.4, 38.9]	1.2 [-23.6, 26.0]
APOLLO	I	3.1 [-10.4, 16.6]	-1.3 [-35.2, 32.7]	1.2 [-40.7, 43.1]
PersonalAlign	Δ_0	0.6 [-10.7, 11.9]	3.1 [-18.2, 24.5]	1.9 [-13.5, 17.3]
PersonalAlign	Δ_1	1.9 [-5.5, 9.3]	6.9 [-19.1, 32.9]	-1.9 [-20.4, 16.6]
PersonalAlign	I	1.2 [-6.1, 8.6]	4.4 [-22.2, 31.0]	-3.2 [-33.9, 27.5]
SmartAgent	Δ_0	0.0 [-9.6, 9.6]	0.6 [-19.4, 20.7]	-1.2 [-16.7, 14.2]
SmartAgent	Δ_1	1.2 [-6.1, 8.6]	-1.8 [-33.4, 29.8]	-3.8 [-21.5, 14.0]
SmartAgent	I	1.2 [-10.8, 13.3]	-2.4 [-47.6, 42.8]	-2.5 [-20.1, 15.1]
ODYSSE	Δ_0	1.2 [-21.4, 23.9]	1.2 [-26.9, 29.3]	-4.4 [-19.9, 11.1]
ODYSSE	Δ_1	-0.6 [-6.7, 5.4]	-3.1 [-22.2, 16.0]	0.7 [-29.0, 30.4]
ODYSSE	I	-1.9 [-29.9, 26.1]	-4.4 [-49.8, 41.0]	5.1 [-38.9, 49.2]
PARPO	Δ_0	-1.9 [-9.3, 5.5]	-2.5 [-44.6, 39.7]	1.3 [-17.3, 19.8]
PARPO	Δ_1	1.2 [-6.1, 8.6]	1.3 [-34.5, 37.2]	-0.7 [-12.5, 11.0]
PARPO	I	3.1 [-10.4, 16.6]	3.8 [-47.3, 54.9]	-2.0 [-21.2, 17.2]
PAHF	Δ_0	-3.1 [-3.1, -3.1]	-3.1 [-28.4, 22.1]	3.1 [-22.1, 28.4]
PAHF	Δ_1	-1.2 [-13.3, 10.8]	-0.7 [-30.1, 28.8]	0.8 [-22.1, 23.6]
PAHF	I	1.9 [-10.2, 14.0]	2.5 [-33.8, 38.8]	-2.5 [-39.2, 34.3]
CIPHER	Δ_0	-1.9 [-9.3, 5.5]	-0.6 [-28.3, 27.1]	2.5 [-3.6, 8.6]
CIPHER	Δ_1	-0.6 [-15.4, 14.2]	-1.9 [-31.5, 27.7]	-0.6 [-29.2, 27.9]
CIPHER	I	1.2 [-10.8, 13.3]	-1.2 [-50.5, 48.0]	-3.3 [-29.8, 23.2]
TAPS	Δ_0	-3.8 [-29.7, 22.2]	6.2 [-37.6, 50.1]	-0.1 [-55.5, 55.2]
TAPS	Δ_1	-1.2 [-8.6, 6.1]	0.0 [-9.6, 9.6]	-2.5 [-42.3, 37.3]
TAPS	I	2.5 [-28.3, 33.3]	-6.2 [-50.3, 37.8]	-2.4 [-90.4, 85.7]

E.3 Observed behavior and judge disagreement

Table 3 gives Astra's observable preference counts; Table 12 gives the corresponding Opus counts. The per-setting records retain task, preference, joint, and permission outcomes separately. Opus identifies four partial memory matches and no full matches; this does not establish preference learning in either case. Table 13 reports agreement between the two judges on outputs assessed by both.

Table 12: Opus judgments across all 8,320 executions under the four update/exploration settings. Obs. counts full matches, partial matches, and contradictions. Unobs. means the trace lacks the evidence needed to judge the preference; N/A means not applicable. Missing includes host failures and unavailable judgments. An unobservable memory preference has no match rate, not a zero success rate.

Channel	Match/Obs.	Partial	Contra.	Unobs.	N/A	Missing
Style	4257/8270	3238	775	0	0	50
Tool	7056/8026	968	2	0	244	50
Memory	0/4	4	0	8193	73	50
Clarification	1102/1191	84	5	0	7079	50
Tone	5466/8270	2789	15	0	0	50

Table 13: Agreement between the two model judges on available paired categorical judgments. Task here compares response judges, not the deterministic task validator used for the primary endpoint. Memory agreement is dominated by shared unobservability. These are model-model agreement statistics; there are no human annotations.

Outcome	Paired judgments	Agreement %	Cohen's κ
Task judge	8303	79.6	0.150
Style	8270	61.7	0.317
Tool	8270	65.8	0.184
Memory	8270	99.1	0.000
Clarification	8270	87.0	0.603
Tone	8270	67.0	0.174

E.4 From accepted feedback to later instructions

We trace each accepted instruction into later calls by its event identifier, rather than inferring use from a matching reply.

Table 14: New preferences traced into later host instructions. Counts follow events during evaluation for the same user, not preferences learned before the test. Later turn means another task is scheduled; that task may not need the preference. Final-turn events are excluded from later-use counts. Compiled counts instructions verified in later host input. Missing rule identifiers can leave deliveries uncounted; delivery does not establish that the host followed the instruction. The two schedules differ and are not a paired performance comparison.

Study	Accepted	Later turn	Retrieved	Compiled
Initial five-seed test	2466	2195	1114	619
Updates by exploration	2241	2009	551	186

Table 14 reports the aggregate counts; the accompanying records retain every method and setting. Updates and accepted events occur after the current response. Later-use counts follow the same new event into subsequent turns; final-turn events have no later task and are censored. Of 186 verified later deliveries, 150 first occur one turn later, 28 two turns later, 6 three turns later, and one each four and five turns later. An untraced preference may not apply to a later task, or the records may not identify the rule. The counts establish which instructions were delivered. Estimating how those instructions changed behavior requires a behavioral comparison.

The full-match example uses FABLE with seed 2027 and is also the worked sequence in Section 3.3. Figure 4 contrasts it with a sequence in which a delivered rule is only partly followed.

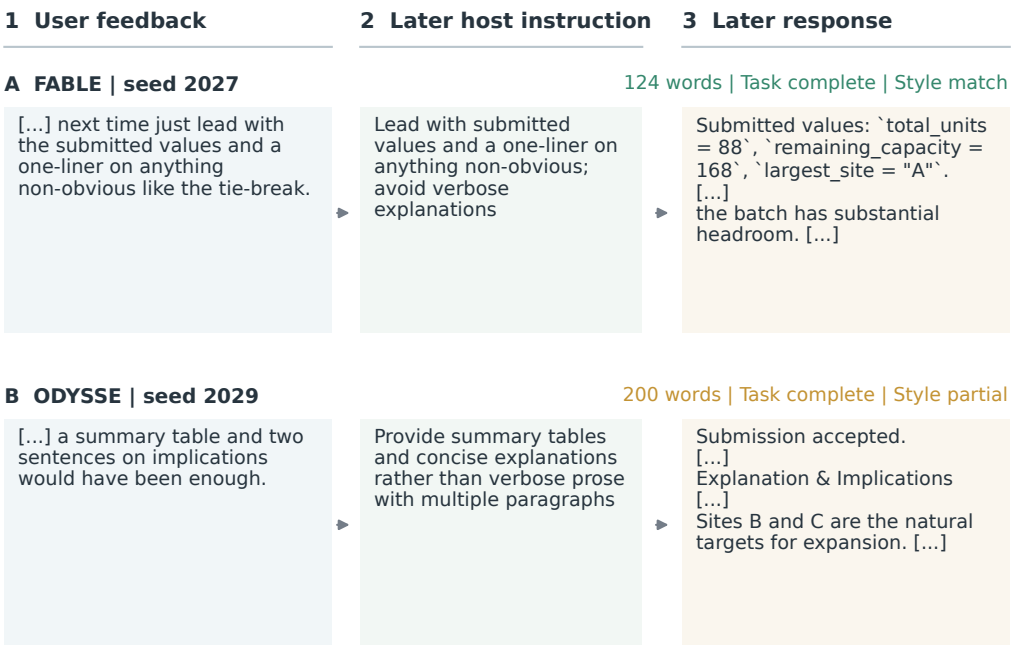

Figure 4: A delivered instruction may be followed fully or only partly. Both examples use the concise user: a visible comment produces an instruction verified alongside other rules in a later host prompt. Both later tasks complete, but Astra and Opus agree on a full style match only in A. These are different tasks, methods, and seeds, not a paired estimate of learning. Within each later-outcome group, we select the first eligible trace by identifier. Excerpts are literal; brackets mark omissions. The accompanying JSON retains the full visible text and selection rule.

F Instructions, Transfer, and User Changes

We test how instruction representation, host SDK, and user changes affect behavior (Sections 5.3 and 5.5). Tables use the same ordering of controls throughout. The single-seed comparisons describe the tested cases rather than a population of users.

F.1 Instruction representations

This comparison gives original applications the same extracted preferences as either direct instructions or personal cards. Figure 5 shows instruction length, style matching, and task completion. The planned card-versus-direct contrast appears in Table 10, alongside the other primary contrasts.

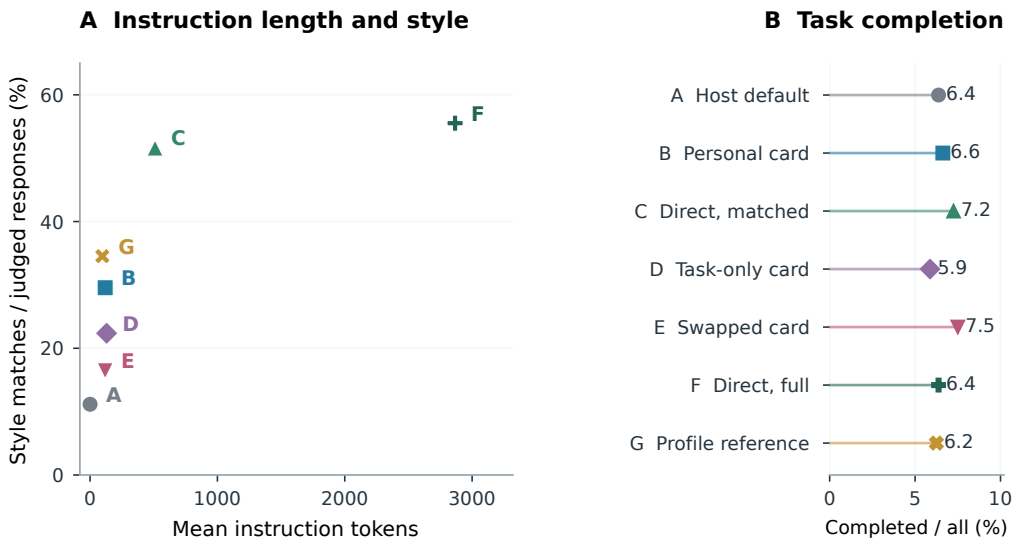

Figure 5: Instruction length and agent behavior in the original applications. Panel A plots mean instruction tokens against full style matches among observable Astra judgments. Panel B shows completed tasks out of all 800 scheduled calls per representation. Letters identify the same representation in both panels. "Direct, matched" and the card share a token ceiling, but differ in actual length; "Direct, full" uses the native ceiling. The privileged profile is a diagnostic reference, not a learned personalization component. These seven conditions do not isolate the effect of instruction length. Table 10 gives the registered paired intervals; exact counts are in the accompanying CSV/JSON records.

The original applications run in restricted environments. Their task scores are not directly comparable to the controlled calculations because the tasks and environments differ. Cards and direct instructions have the same token ceiling but different realized lengths. The privileged profile is a reference, not a learned personalization component or guaranteed upper bound.

F.2 Changing SDKs and prior learning

Figure 6 tests twenty users excluded from extractor training. The fixed-user transfer test changes SDK while keeping learned information fixed. The equal-count test gives the modes the same number of earlier interactions. The instruction-removal comparison keeps numerical state but removes stored preference rules (Table 17).

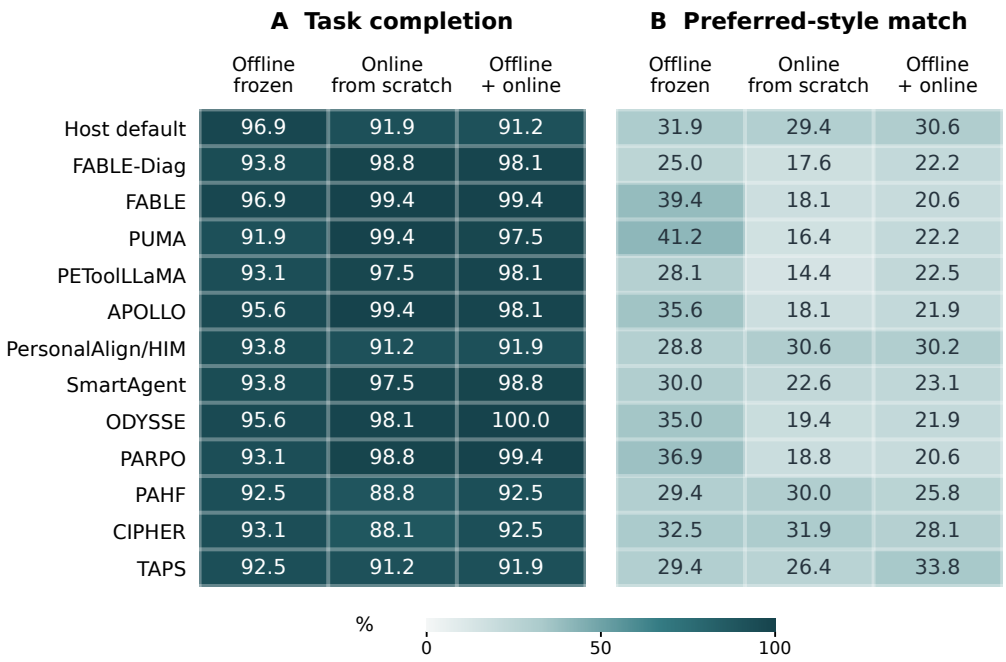

Figure 6: Task completion and answer style for twenty users excluded from extractor training. All thirteen controls and three learning modes are shown in the same order and on the same 0–100 scale as Figure 2. Task completion uses all 160 scheduled calls per cell; style uses observable Astra judgments, including partial and contradictory responses in its denominator. This transfer test uses one execution seed and does not isolate the effect of updating. The accompanying CSV/JSON records retain every numerator and denominator.

Table 15: Fixed-user transfer to another SDK. Task percentages use all 32 scheduled calls per cell; style percentages use observable judgments. Detailed preference tables give counts for each channel. Methods appear in a fixed order, not ranked by results. Frozen and hybrid share the same learning history; online starts with empty user state. Host default never updates. Memory mode controls instruction delivery. Exploration can change settings even when a method does not learn their scores.

	Offline frozen		Online		Hybrid	
Method	Task %	Style %	Task %	Style %	Task %	Style %
Host default	96.9	50.0	100.0	56.2	100.0	56.2
FABLE-Diag	100.0	34.4	100.0	34.4	96.9	40.6
FABLE	96.9	34.4	100.0	31.2	100.0	46.9
PUMA	93.8	65.6	100.0	40.6	100.0	28.1
PEToolLLaMA	96.9	40.6	100.0	25.0	100.0	40.6
APOLLO	100.0	56.2	100.0	34.4	96.9	28.1
PersonalAlign/HIM	93.8	59.4	93.8	50.0	96.9	46.9
SmartAgent	96.9	50.0	100.0	31.2	96.9	25.0
ODYSSE	100.0	37.5	100.0	28.1	96.9	32.3
PARPO	100.0	34.4	100.0	31.2	100.0	21.9
PAHF	100.0	46.9	100.0	53.1	100.0	50.0
CIPHER	96.9	56.2	100.0	46.9	96.9	46.9
TAPS	96.9	50.0	100.0	56.2	96.9	59.4

Table 16: Comparing modes after the same number of earlier interactions. Task percentages use all 160 scheduled calls per cell; style percentages use observable judgments. Detailed preference tables give counts for each channel. Methods appear in a fixed order, not ranked by results. All three states have eight prior interactions. Target online has already learned from target-SDK tasks; it is not a cold start. Host default never updates. Memory mode controls instruction delivery. Exploration can change settings even when a method does not learn their scores.

	Source frozen		Target online		Source hybrid	
Method	Task %	Style %	Task %	Style %	Task %	Style %
Host default	94.4	45.6	96.2	46.2	96.2	41.2
FABLE-Diag	99.4	33.1	100.0	21.2	100.0	21.9
FABLE	98.8	43.1	96.9	19.4	99.4	18.8
PUMA	97.5	45.0	100.0	16.9	100.0	21.2
PEToolLLaMA	97.5	43.1	99.4	20.6	98.8	20.6
APOLLO	98.1	49.4	99.4	21.2	97.5	24.5
PersonalAlign/HIM	96.9	39.4	95.6	40.6	95.6	40.0
SmartAgent	97.5	37.7	94.4	24.4	98.1	26.9
ODYSSE	96.2	43.8	99.4	19.4	99.4	24.4
PARPO	96.9	36.2	100.0	23.1	99.4	23.8
PAHF	96.2	41.2	96.9	45.0	95.6	43.8
CIPHER	96.2	42.1	95.0	45.6	96.2	39.4
TAPS	96.9	47.5	98.1	43.8	97.5	43.8

Table 17: Removing stored instructions while retaining numerical state. Task percentages use all 32 scheduled calls per cell; style percentages use observable judgments. Detailed preference tables give counts for each channel. Methods appear in a fixed order, not ranked by results. Frozen and hybrid share the same learning history; online starts with empty user state. Host default never updates. Memory mode controls instruction delivery. Exploration can change settings even when a method does not learn their scores.

	Offline frozen		Online		Hybrid	
Method	Task %	Style %	Task %	Style %	Task %	Style %
Host default	100.0	56.2	100.0	56.2	96.9	50.0
FABLE-Diag	100.0	25.0	100.0	34.4	100.0	31.2
FABLE	96.9	37.5	100.0	31.2	96.9	34.4
PUMA	100.0	53.1	100.0	34.4	100.0	28.1
PEToolLLaMA	100.0	50.0	100.0	21.9	100.0	28.1
APOLLO	100.0	59.4	100.0	25.0	100.0	37.5
PersonalAlign/HIM	100.0	53.1	100.0	50.0	100.0	56.2
SmartAgent	96.9	53.1	100.0	31.2	100.0	31.2
ODYSSE	100.0	34.4	100.0	25.0	96.9	32.3
PARPO	96.9	28.1	100.0	25.0	100.0	31.2
PAHF	100.0	56.2	100.0	40.6	100.0	53.1
CIPHER	100.0	50.0	100.0	53.1	100.0	53.1
TAPS	100.0	56.2	93.8	50.0	100.0	56.2

F.3 Preference changes, separation, and memory

The following tests change one user's style preference, switch between users, or request a specific memory operation. Each addresses a different reason why later behavior might change.

Preference changes. Recovery requires two consecutive style matches, with an unchanged-style control and a six-task horizon. Frozen sometimes meets the same criterion, so recovery alone does not show learning.

Table 18: Style recovery within the registered six-task horizon. Cells are users with two consecutive style matches/all scheduled users; an unobservable judgment breaks the sequence. Unchanged-profile streams provide a matched control for the same criterion. Non-recovery is censored at the horizon and is not assigned an invented recovery time. These are single-seed descriptive counts.

	Unchanged style			Changed style		
Method	F	O	H	F	O	H
Host default	2/4	2/4	4/4	1/4	2/4	1/4
FABLE-Diag	1/4	2/4	3/4	1/4	1/4	1/4
FABLE	2/4	2/4	1/4	1/4	2/4	1/4
PUMA	4/4	1/4	2/4	0/4	2/4	1/4
PEToolLLaMA	2/4	1/4	1/4	3/4	2/4	1/4
APOLLO	1/4	1/4	1/4	1/4	1/4	1/4
PersonalAlign/HIM	2/4	2/4	1/4	3/4	2/4	1/4
SmartAgent	1/4	1/4	2/4	2/4	1/4	2/4
ODYSSE	3/4	1/4	3/4	1/4	2/4	1/4
PARPO	1/4	1/4	2/4	1/4	1/4	1/4
PAHF	3/4	2/4	3/4	3/4	1/4	2/4
CIPHER	2/4	2/4	3/4	3/4	3/4	2/4
TAPS	3/4	3/4	4/4	3/4	3/4	2/4

User separation. Both switching orders contain 104 steps. Neither records changes to the inactive user's state or store, or instructions sourced from another user. The other user's marker appears in $0/102$ observable interleaved responses and $0/104$ isolated responses. These are local runtime checks, not adversarial privacy guarantees. The full counts and qualifications remain in the accompanying user-isolation records.

Memory operations. Explicit tasks test retention, recall, and other requested operations using tool traces and stored contents. They do not establish learning of an unstated memory preference.

Table 19: Cross-task memory capability on seven SDK entry points. Task success retains failed executions in the denominator; state correctness uses observed local validators. Readback tasks are split by their original requirement: retrieve a retained or updated value, or confirm absence after forgetting or non-retention. Correct abstention is not retrieval. Denied attempts and unauthorized writes are distinct. Oracle storage is privileged. These operations test runtime capability, not preference learning.

Control	Operation	Calls	Task	State	Denied	Unauth.
Memory enabled	Retain	28	28/28	28/28	0	0
Memory enabled	Recall stored	56	54/56	55/56	0	0
Memory enabled	Confirm absent	56	56/56	56/56	0	0
Memory enabled	Update	28	27/28	27/28	0	0
Memory enabled	Forget	28	28/28	28/28	0	0
Memory enabled	Do not retain	28	28/28	28/28	0	0
No memory	Retain	28	0/28	0/28	0	0
No memory	Recall stored	56	0/56	0/56	6	0
No memory	Confirm absent	56	56/56	56/56	5	0
No memory	Update	28	0/28	0/28	0	0
No memory	Forget	28	28/28	28/28	0	0
No memory	Do not retain	28	28/28	28/28	0	0
Oracle store	Retain	28	28/28	28/28	0	0
Oracle store	Recall stored	56	54/56	56/56	0	0
Oracle store	Confirm absent	56	56/56	56/56	0	0
Oracle store	Update	28	27/28	28/28	0	0
Oracle store	Forget	28	27/28	28/28	0	0
Oracle store	Do not retain	28	28/28	28/28	0	0

G Simulated Users and Feedback Extraction

This appendix details the simulated-user data (Section 4.1), extractor training (Section 4.2), and extraction and card results (Section 5.3). No personalization component controls the host during collection, and the extractor does not select host actions.

G.1 Profiles and collection checks

Each of 100 implementations runs the same 100 tasks under four fixed profiles and a task-only control, producing 50,000 conversations and 145,923 feedback examples. The host sees the task, not the profile, and its context resets each task. The simulator privately receives the profile and reacts to the visible response and reported tool activity. Hosts run separately by condition, so matching a task does not make the responses or histories identical. The simulator can accept, correct, continue, or stop without expressing a preference on every turn. The task-only control has no stable style, tool, memory, clarification, tone, expertise, or patience preference.

Table 20: The four fixed simulated users. Each profile also forbids sensitive memory retention and requires confirmation before destructive actions.

Profile	Answer and tone	Tools and questions	Memory
Concise/direct	Lead with the answer; neutral and direct	Use tools only when materially useful, omit routine narration, and ask only blocking questions	Retain stable work preferences but not task-sensitive details
Detailed/ analytical	Explain assumptions, steps, evidence, and implications	Verify proactively, identify evidence, and clarify consequential ambiguity	Retain stable preferences and non-sensitive project context
Structured/ operator	Use compact labeled bullets with actions and exceptions separated	Verify current state, summarize consequential evidence, and ask one focused question when needed	Retain workflow preferences and identify them when applied
Plain/cautious	Use accessible connected prose and a calm conversational tone	Confirm consequential external actions and clarify meaningful ambiguity	Do not retain information across tasks without permission

Table 21: Simulated-user collection without active personalization components. Each condition contains the same 10,000 base tasks. Duplicate rate uses normalized visible feedback. Consistency repair retries concise-user feedback after an unmistakably overlong, heavily structured host answer; it is not a general preference-labeling rule. Goal met is the simulator's final task assessment, not an independent environment score.

Condition	Turns/record	Chars/feedback	Duplicate (%)	Repair (%)	Goal met (%)
Concise/direct	3.549	188.9	5.37	19.23	68.17
Detailed/analytical	2.863	479.4	0.11	0.00	69.66
Structured/operator	2.825	309.4	0.19	0.00	69.60
Plain/cautious	2.933	295.8	0.36	0.00	69.33
Task-only	2.423	341.3	0.31	0.00	72.73

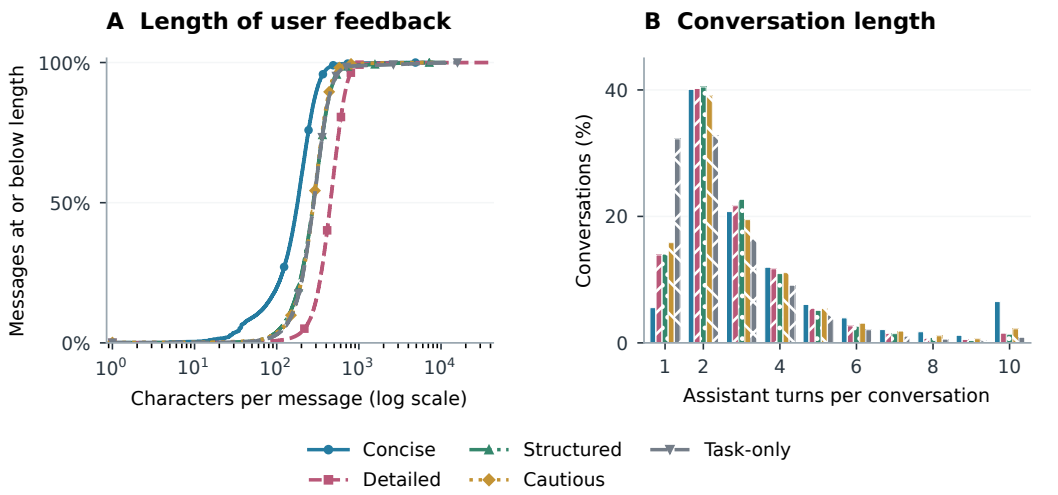

Figure 7: Observed feedback and conversation lengths for all five collection conditions. A shows the fraction of nonempty user messages at or below each character count, using a log horizontal scale and the full observed range. B shows assistant turns per conversation, including conversations that reach the ten-turn limit. Each condition contains 10,000 conversations; together they yield 145,923 turn-level windows. Curves and bars use exact frequency counts without smoothing. The conditions share base tasks, not necessarily responses or histories.

No collection events are missing or malformed. A classifier using only user messages identifies profile type with 0.846 balanced accuracy, versus 0.25 chance. Detailed-user messages are $2.54\times$ as long as concise-user messages. Figure 7 shows the full distributions of feedback length and conversation turns under each condition. The profiles therefore produce distinguishable simulated feedback. These measurements do not assess how well the comments represent real users. A targeted consistency check retries 19.23% of concise-user turns after unmistakably long, heavily structured answers. Other conditions have no such repair; the corpus is not unfiltered simulator output.

G.2 Extractor training and evaluation

We use these comments to train the extractor, not a host response policy or all the personalization methods. During execution, we keep the extractor fixed; each method decides how its predictions change the user's state.

The profile-blind teacher sees the request, response, reported tool names and statuses, one prior exchange, and newest user message. It labels feedback, polarity, Likert rating, target behavior, scope, persistent preference, supported instruction, exact evidence, and confidence. It never sees the hidden profile or private assessment. Evidence must be copied from the newest reaction; scope extends beyond the current turn only when the wording supports it.

The $8{,}000$ examples are selected before labeling and split by base scenario into $6{,}000/1{,}000/1{,}000$ train/development/test examples. No scenario crosses a split. Sonnet 4.5 produces $7{,}999$ labels; a logged profile-blind GPT fallback handles one refusal. Allocated labeling cost is $131.61 ($0.0165 per label). A separate test covers 20 unseen profiles using recorded responses.

Qwen3-0.6B LoRA models train for four epochs under five seeds and retain the checkpoint with lowest shared development loss. Five-condition training uses all profile conditions and task-only data. For task-only training, examples are repeated to match the $6{,}000$ training instances. This equalizes training size while leaving different numbers of unique examples.

To also match the scenarios, a second comparison uses one example from each of 578 shared scenarios per arm. The paired examples can still differ in conversation turn, response, and history, so they are not strict counterfactual versions of the same turn. Measures and Student-t intervals use

the five training seeds, with no weighted summary score. Figure 8 and Table 22 compare extraction performance under the two training conditions.

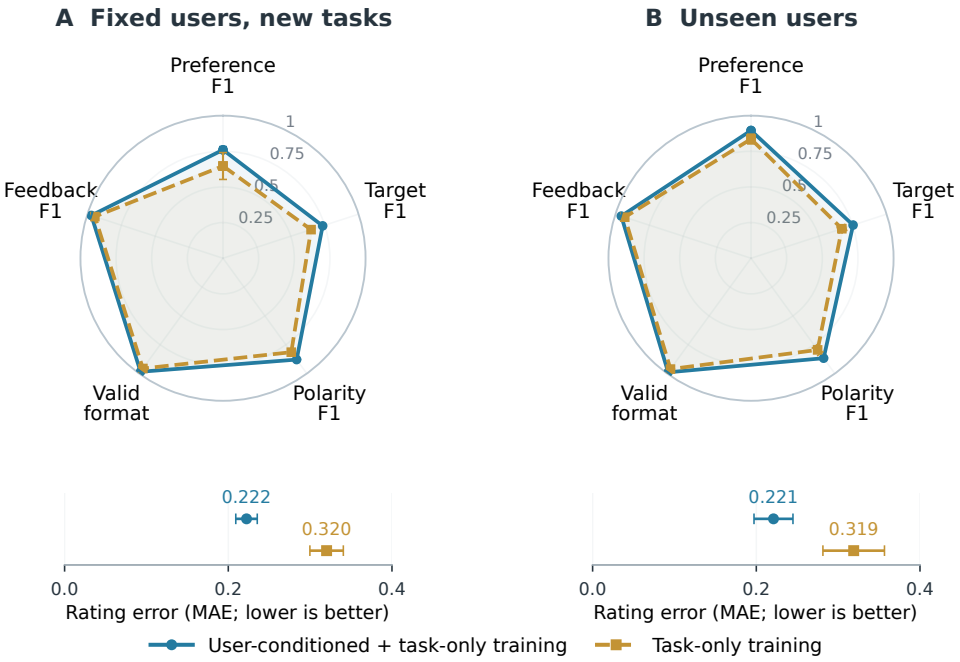

Figure 8: Feedback from the simulated profiles improves preference detection for fixed and unseen users. The two models use the same architecture and 6,000 training instances; task-only examples are repeated to match that budget. Each spoke uses a 0–1 scale. Rating error is shown separately because lower values are better. Whiskers are 95% Student-t intervals over five training seeds. Table 22 gives all six metrics and their intervals; the radar area is not an overall score.

Table 22: Profile-blind feedback extraction. Cells are means [95% Student-t intervals] over five training seeds. Both training conditions use the same model and 6,000 optimization instances; task-only examples are repeated to match that budget. Bold marks a better mean whose paired seed-difference interval excludes zero.

	Seen users, new tasks		20 unseen users	
Metric	**Five-condition**	**Task-only**	**Five-condition**	**Task-only**
Schema ↑	**0.984 [0.980, 0.988]**	0.956 [0.946, 0.966]	**0.989 [0.986, 0.992]**	0.961 [0.945, 0.977]
Feedback F1 ↑	**0.971 [0.969, 0.973]**	0.945 [0.939, 0.951]	**0.957 [0.952, 0.961]**	0.931 [0.916, 0.947]
Personalization F1 ↑	**0.760 [0.738, 0.783]**	0.647 [0.553, 0.741]	**0.896 [0.881, 0.911]**	0.836 [0.802, 0.870]
Polarity F1 ↑	**0.879 [0.865, 0.892]**	0.815 [0.796, 0.833]	**0.867 [0.856, 0.877]**	0.795 [0.762, 0.827]
Likert MAE ↓	**0.222 [0.209, 0.235]**	0.320 [0.300, 0.341]	**0.221 [0.197, 0.245]**	0.319 [0.281, 0.357]
Target F1 ↑	**0.735 [0.731, 0.739]**	0.649 [0.637, 0.662]	**0.753 [0.742, 0.764]**	0.670 [0.642, 0.698]

Table 23: Shared-scenario training ablation with 578 unique examples per arm. Cells give profile minus task-only differences with two-sided 95% Student-t intervals over five training seeds. Positive values are better except for Likert MAE. Bold marks intervals wholly in the favorable direction. Pairs share scenarios but may differ in responses or histories.

Metric	Fixed users, new tasks	20 unseen users
Schema	$-0.013\,[-0.019, -0.007]$	$-0.019\,[-0.041, +0.003]$
Feedback F1	$-0.007\,[-0.013, -0.002]$	$+0.016\,[-0.010, +0.042]$
Personalization F1	$\mathbf{+0.473\,[+0.287, +0.660]}$	$\mathbf{+0.653\,[+0.521, +0.785]}$
Polarity F1	$+0.020\,[-0.001, +0.040]$	$+0.009\,[-0.030, +0.048]$
Likert MAE	$-0.001\,[-0.024, +0.021]$	$-0.019\,[-0.110, +0.072]$
Target F1	$\mathbf{+0.031\,[+0.012, +0.050]}$	$+0.029\,[-0.000, +0.059]$

The shared-scenario comparison improves preference detection for both fixed and unseen users, but slightly reduces formatting validity and general-feedback F1 for fixed users. The task-only false-positive difference, -0.048 (95% CI $[-0.188, 0.092]$), does not establish a reduction in false preference detections.

Table 24 reports the underlying false-positive rates alongside training time and extraction latency.

Table 24: Extractor false positives and runtime. Personalization FPR is measured on held-out task-only reactions. Runtime values are means with two-sided 95% Student-t intervals over five seeds; training used one A100-40GB per run.

Training data	**Pers. FPR ↓**	**Train min. ↓**	**Inference ms/ex. ↓**
Five-condition	$0.117\,[0.091, 0.144]$	$48.473\,[48.401, 48.544]$	$294.014\,[278.194, 309.835]$
Task-only matched	$0.165\,[0.043, 0.287]$	$48.213\,[48.064, 48.362]$	$286.171\,[259.704, 312.639]$

G.3 Personal-card contents and separation

The card builder combines predicted persistent instructions per user. We fix the extractor seed at 2026 before inspecting test results. An independent GPT-5.6/Codex judge compares each card with the target profile one preference at a time, without identities or source conversations. It also compares each fixed user's card with the other three profiles. Table 26 reports the own-profile match rate minus the match rate against those other profiles. The teacher reference uses profile-blind labels, not the hidden profile as an instruction source.

Table 25: Matches between personal-card instructions and specified preferences. Cards use profile-blind predictions from the prespecified seed 2026. Cells count full matches for each preference among the designed profiles; partial matches are excluded. The fixed-user teacher oracle uses teacher records, not extractor predictions. These descriptive rates are not population estimates. Unseen-user memory scores check the card's stated preference, not demonstrated retention or recall.

Users	**Events used for card**	**Style**	**Tool**	**Memory**	**Clarification**	**Tone**
Fixed users	AgentBake-8K	3/4 (75%)	1/4 (25%)	0/4 (0%)	0/4 (0%)	4/4 (100%)
	Task-only matched	3/4 (75%)	2/4 (50%)	0/4 (0%)	0/4 (0%)	2/4 (50%)
	Teacher oracle	3/4 (75%)	3/4 (75%)	1/4 (25%)	0/4 (0%)	3/4 (75%)
20 unseen users	AgentBake-8K	4/20 (20%)	0/20 (0%)	0/20 (0%)	0/20 (0%)	0/20 (0%)
	Task-only matched	4/20 (20%)	0/20 (0%)	0/20 (0%)	0/20 (0%)	0/20 (0%)

Table 26: Fixed-user card isolation at seed 2026. Each cell is the channel match rate for the card's own profile minus its match rate against the other three profiles. Positive values indicate user-specific separation; channels remain separate.

Events used for card	Style	Tool	Memory	Clarification	Tone
AgentBake-8K	+0.667	+0.083	+0.000	+0.000	+0.583
Task-only matched	+0.667	+0.417	+0.000	+0.000	+0.417
Teacher oracle	+0.667	+0.667	+0.250	+0.000	+0.500

The cards do not fully represent all specified preferences. In the memory dimension, for example, the score assesses the recorded instruction about memory use; retention and recall require separate execution tests. Card-content scores therefore do not measure host compliance or real-user satisfaction.

G.4 Data and reproducibility

The source files contain prompt excerpts, with version hashes in `reporting/prompt_hashes_v6.json`. Full templates are retained in the versioned experiment records. The conversation data remains private because it contains hidden profiles and simulator assessments; training inputs omit these fields. The separate release copy has credentials removed, while original local data is unchanged.

Selections, splits, model manifests, source hashes, calls, failures, and judge-input mappings support reconstruction without new model calls. Environment snapshots record installed packages, not reconstructed dependency locks.

H Supporting Configuration-Choice Diagnostics

This appendix describes a selection study conducted separately from the host-execution comparisons. It contains 100 implementation records linked to 66 source agents; 921/991 selected scenarios have usable decision sequences (Table 28). The implementation inventory is supplied with the run metadata.

H.1 Learning to select configurations

The execution studies judge what a host actually does after receiving preferences. This separate study asks whether a method learns to select settings that match a written profile over longer sequences. Six scripted profiles describe brevity, thorough research, security, hands-off delegation, step-by-step learning, and pragmatic development; they differ from the four simulated profiles in the execution studies. Table 27 compares held-out choices, and Figure 9 follows learning and an unannounced profile change. Both use the configuration-choice score defined below, which judges selected settings rather than host responses.

The same selection problem also yields reward, regret, and cost. Reward combines fixed preference, task-adequacy, cost, and violation terms. Regret is the utility gap to the best permitted configuration under the evaluator without feedback noise. Cost is a normalized configuration score on a 0–1 scale; execution tokens and latency are measured separately. Among learning methods, FABLE-Diag has the highest mean configuration alignment, while FABLE has the highest mean reward and lowest mean regret. These rankings apply to the scripted selection problem.

Table 27: Held-out configuration-choice scores under the scripted evaluator (5 seeds, 100 implementations, 6 scripted user profiles; mean [95% cluster-bootstrap interval] over 66 normalized source agents and 2,000 resamples). Framework ports of the same source are resampled together. FABLE-Diag uses a diagonal posterior; other named methods are AgentBake adaptations, not original checkpoints. Adequacy rates whether a configuration suits the task, not whether the host completed it. Cost is a normalized configuration score, not measured execution spending. The best personalization method in each column is bold.

Method	Alignment ↑	Adequacy ↑	Reward ↑	Regret ↓	Cost ↓
FABLE-Diag	**0.967 [0.963, 0.970]**	0.750 [0.736, 0.763]	0.622 [0.599, 0.640]	0.055 [0.051, 0.058]	0.342 [0.326, 0.355]
FABLE	0.937 [0.933, 0.940]	0.798 [0.780, 0.814]	**0.648 [0.624, 0.669]**	**0.028 [0.024, 0.031]**	0.305 [0.288, 0.321]
PUMA	0.756 [0.751, 0.761]	0.763 [0.750, 0.774]	0.442 [0.422, 0.459]	0.235 [0.226, 0.242]	0.406 [0.386, 0.425]
PETool					
LLaMA	0.568 [0.559, 0.575]	0.860 [0.845, 0.874]	0.387 [0.365, 0.405]	0.290 [0.281, 0.298]	0.265 [0.240, 0.290]
APOLLO	0.566 [0.555, 0.575]	0.872 [0.855, 0.887]	0.399 [0.376, 0.418]	0.277 [0.269, 0.284]	**0.236 [0.211, 0.260]**
PersonalAlign/					
HIM	0.627 [0.620, 0.634]	**0.895 [0.872, 0.915]**	0.470 [0.442, 0.494]	0.206 [0.199, 0.212]	0.263 [0.237, 0.289]
SmartAgent	0.917 [0.910, 0.922]	0.789 [0.770, 0.805]	0.628 [0.605, 0.647]	0.048 [0.044, 0.052]	0.285 [0.268, 0.300]
ODYSSE	0.946 [0.941, 0.951]	0.761 [0.744, 0.777]	0.623 [0.601, 0.641]	0.053 [0.048, 0.059]	0.296 [0.276, 0.314]
PARPO	0.957 [0.950, 0.964]	0.761 [0.743, 0.776]	0.626 [0.602, 0.645]	0.050 [0.046, 0.054]	0.330 [0.312, 0.346]
PAHF	0.904 [0.896, 0.912]	0.709 [0.697, 0.720]	0.527 [0.504, 0.546]	0.149 [0.143, 0.154]	0.401 [0.393, 0.409]
CIPHER	0.859 [0.852, 0.865]	0.711 [0.700, 0.721]	0.482 [0.458, 0.502]	0.195 [0.188, 0.202]	0.454 [0.445, 0.462]
TAPS	0.869 [0.850, 0.884]	0.761 [0.747, 0.775]	0.567 [0.539, 0.589]	0.109 [0.101, 0.118]	0.279 [0.261, 0.296]
Host default	0.483 [0.477, 0.489]	0.944 [0.921, 0.962]	0.388 [0.361, 0.411]	0.288 [0.281, 0.294]	0.232 [0.206, 0.258]

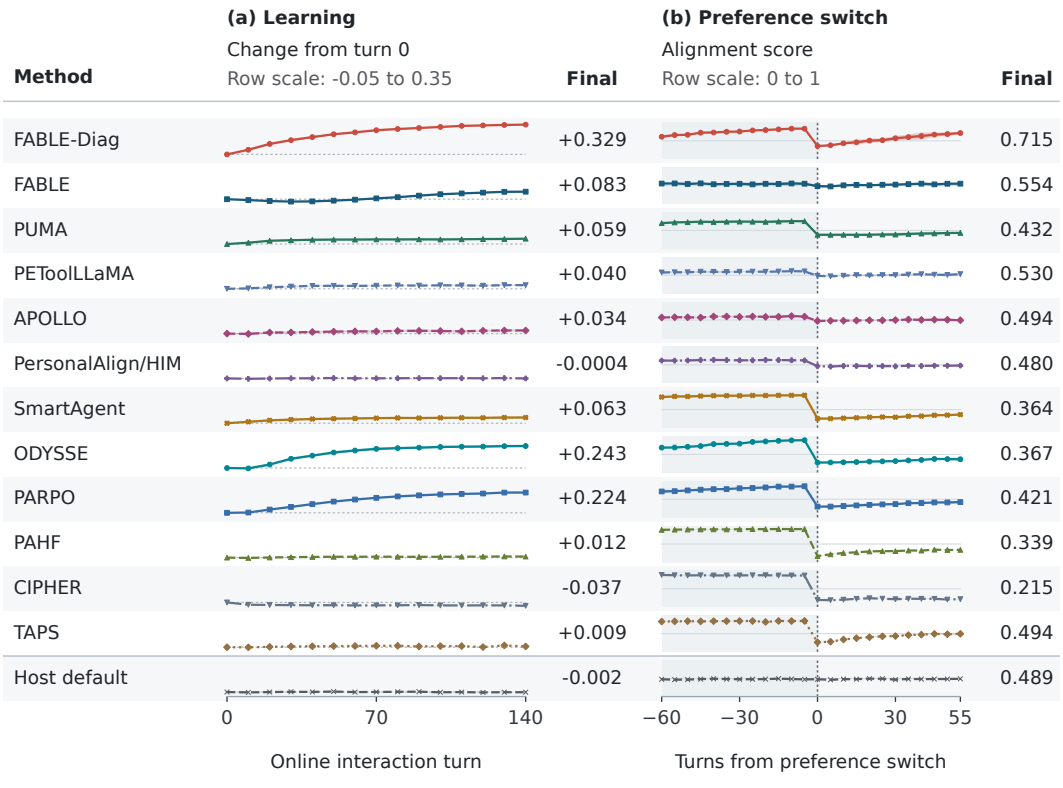

Figure 9: Configuration-choice alignment during learning and after a hidden profile switch. Each method has its own row; vertical scales are shared within each column. Panel (a) subtracts each method's turn-zero score. Panel (b) shows absolute alignment; dotted lines mark the switch. Points show means and bands show 95% intervals over five seeds in panel (a) and three seeds in panel (b). Final means appear at right. Both panels use Equation 3, not the style of executed host replies.

Table 28: Trace-corpus construction and coverage. Selected scenarios have selection metadata; usable scenarios contain a nonempty trace sequence. Decisions apply the benchmark's 200-turn cap separately to each implementation. Tool columns count implementations whose declared topology maps to the corresponding benchmark mode; they do not establish live control realization. A source agent may appear under multiple SDKs, so source counts are deduplicated in the total row.

SDK	Impl.	Sources	Selected	Usable	Decisions	Any ext.	Web	File	Code
autogen	16	16	160	158	2,440	14	14	4	3
crewai	14	13	131	119	1,836	10	8	3	4
langchain	15	14	150	131	1,875	14	13	2	2
langgraph	30	29	300	292	4,878	19	18	3	2
llamaindex	5	5	50	38	545	5	5	0	0
pydanticai	10	10	100	95	1,531	8	6	1	3
strands	10	10	100	88	972	7	7	1	0
Total	100	66	991	921	14,077	77	71	14	14

H.2 Protocol and score interpretation

Each component receives task type, risk, ambiguity, memory need, and tool need, then selects memory, tool, and style settings from the same capability-filtered candidates. The catalog has five memory, six tool, and six style modes. Six scripted user profiles cover brevity, thorough research, security, hands-off delegation, step-by-step learning, and pragmatic development. They are separate from the four simulated users listed in Table 20. Before online updates, each method receives preset numerical answers about brevity, tool use, memory use, clarification, or confirmation. These answers initialize the methods that support onboarding; they are not inferred from feedback. The evaluator's full profile scores and task-specific preference rules are not passed to the component. Each implementation is capped at 200 turns, with learning during the first 70% and frozen evaluation on the remaining 30%. The main comparison contains $5{,}490{,}030$ decisions over five seeds; the hidden-switch test has $385{,}164$ decisions over three seeds, fourteen implementations, and opposing profile pairs.

The configuration-choice score measures how the selected memory, tool, and style settings match a written profile. For recorded situation C_t and selected configuration a_t, it normalizes the profile score $P(C_t, a_t)$ within the available candidates:

$$\mathrm{Align}(C_t, a_t) = \frac{P(C_t, a_t) - \min_{a \in \mathcal{A}_t} P(C_t, a)}{\max_{a \in \mathcal{A}_t} P(C_t, a) - \min_{a \in \mathcal{A}_t} P(C_t, a)}. \tag{3}$$

When all candidates have the same profile score, alignment is defined as one. The same normalization is applied separately to memory, tools, and style.

This score is used for the configuration-choice table and curves in this appendix (Table 27; Figure 9). The execution comparisons instead use the independently judged response-level preference match rate from Section 4.3.

Task adequacy instead scores whether a setting suits the constructed task, independently of the user. It is not completed-task success. The noisy learning reward combines fixed preference, adequacy, cost, and rule-violation terms:

$$u_{\boldsymbol{\alpha}}(C, a) = \tanh(\alpha_P P(C, a) + \alpha_A(2A(C, a) - 1) - \alpha_C C(a) - \alpha_V V(C, a)), \tag{4}$$

$$y_t = \mathrm{clip}_{[-1,1]}(u_{\boldsymbol{\alpha}}(C_t, a_t) + \varepsilon_t). \tag{5}$$

The released evaluator fixes these weights across methods. Regret is the gap to the best candidate under the noiseless evaluator; cost and violations are reported separately. Recorded host outcomes remain metadata, not rewards for unexecuted alternative actions.

Candidate-set violations are zero in the main, switch, and sensitivity tests because all controls choose from the permitted list. This checks action selection, not general runtime security. Source-group uncertainty and the full SDK/structure breakdown remain in the source tables; SDK groups have different agents and capabilities and do not rank framework quality.

The archived preference-switch driver cannot import a helper from the current runner. The saved decisions and curves have been checked. The plots can be regenerated from those records, but the original driver has not been rerun.

H.3 Preference channels and sensitivity tests

Figure 10 reports memory, tool, and answer-style alignment alongside overall alignment and task adequacy. Figure 11 compares configuration-choice alignment as feedback noise, frequency, budget, and the cost weight vary.

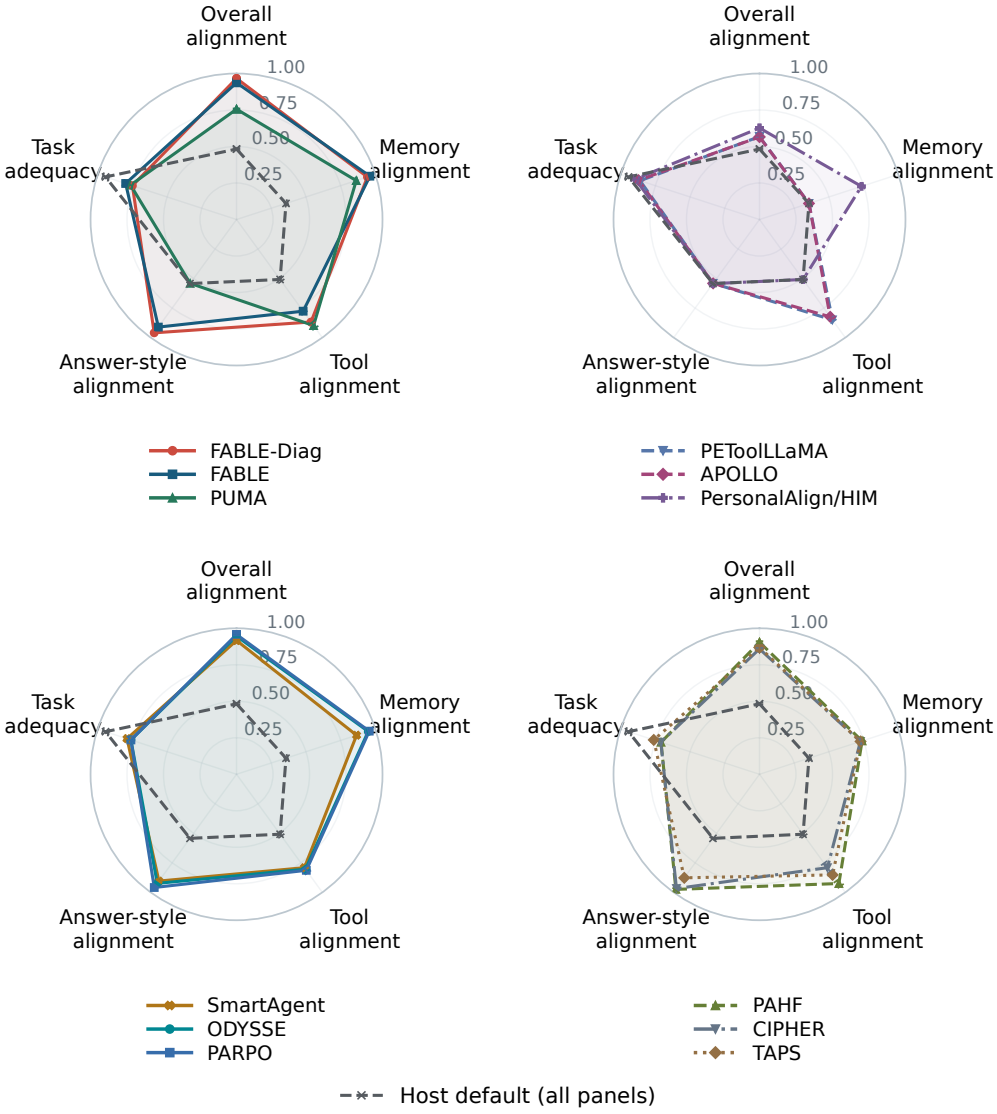

Figure 10: Five held-out configuration-choice scores on their original 0–1 scales. The four panels use the method order of Table 27; the non-learning host is repeated as a dashed reference. Vertices are means; radial whiskers are 95% cluster-bootstrap intervals over 66 normalized source agents and 2,000 resamples. Overall alignment and task adequacy accompany the three preference channels. No score is rescaled or combined into a radar-area ranking. Task adequacy rates the selected setting, not completed host work; memory alignment is not demonstrated recall. Specialists retain host choices outside their control scope. The exact channel table and both source CSVs accompany the figure.

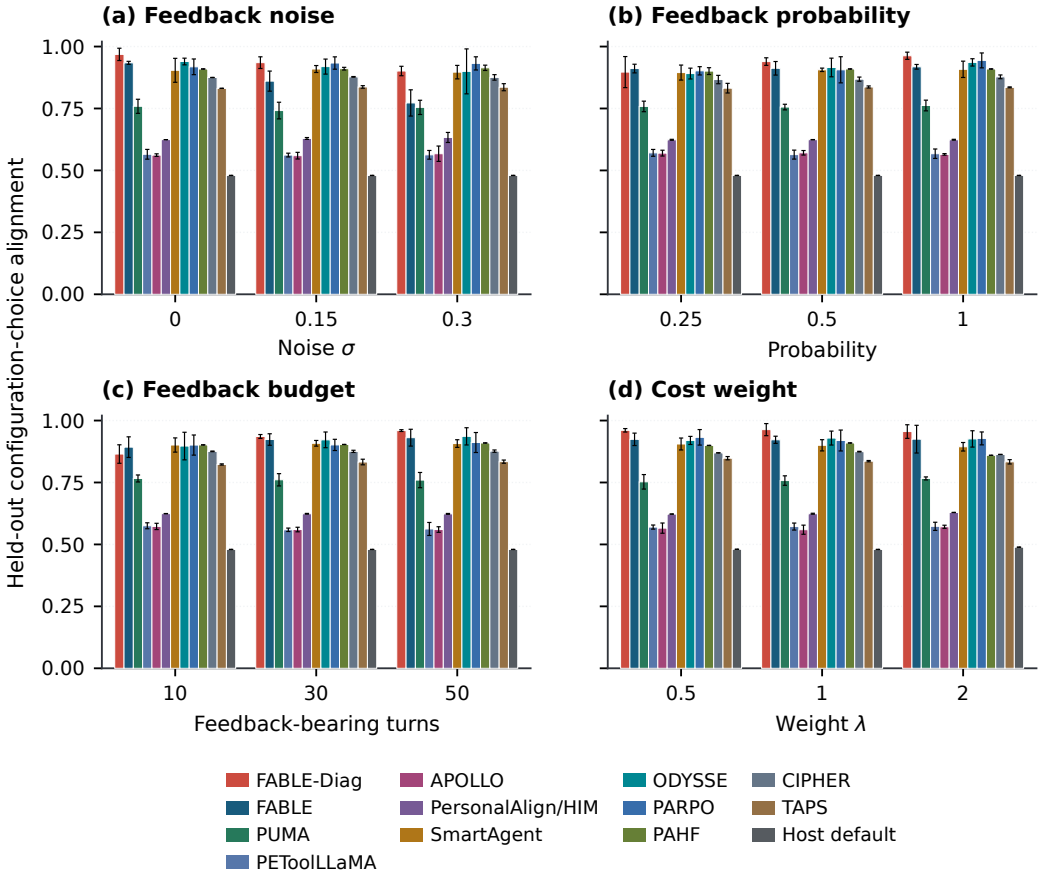

Figure 11: Configuration-choice alignment under noisier or less frequent feedback, different feedback budgets, and different cost weights. Each test uses one implementation per framework and three independent seeds. All thirteen controls are shown, with the same method colors as the learning curves. Bars show means; whiskers show 95% confidence intervals. Each vertical axis starts at zero. These are simulated choice scores, not fresh task-completion rates.

I ACCURATE INSTRUCTIONS AND CONTINUED LEARNING

These follow-ups use the main calculation tasks, not the restricted original applications: seven SDKs, four fixed users, eight held-out tasks, native tools, and the deterministic checker. Ten execution replicates per SDK/user combination sample new outputs, not new users or separately trained hosts.

Accurate instructions. Each matched task is executed afresh with the user's specified style instruction, no instruction, or the opposing user's instruction. Concise/direct opposes detailed/analytical; structured/operator opposes plain/cautious. One template renders the literal instruction and preferred and avoided styles. Accurate and opposite instructions share position and priority; the control omits the block. Correctness and permissions take priority. Task data reset before execution; a hash orders conditions within each matched task. The $7 \times 4 \times 10 \times 8 \times 3 = 6{,}720$ scheduled positions yield $6{,}622$ successful host executions and 98 retained failures. No simulator, extractor, or learner is used.

Each SDK request is recorded before forwarding. If the required instruction is absent from system, developer, and user messages, forwarding is blocked and the execution remains a failure. Tool output and metadata do not qualify. Delivery verifies presence in an attempted request, not a returned answer or style match.

CIPHER and PAHF learning. A simulated user accepts or edits the host answer. Released CIPHER-N completion and learning routines (Gao et al., 2024) infer preferences from original and edited answers and store task/preference pairs. The source BERT encoder retrieves three nearby tasks; their preferences are summarized before the next host call. Stored pairs need not contain newly inferred preferences.

PAHF retains retrieval, pre-action questioning, and post-feedback updates (Liang et al., 2026). During training, it may ask one style question; the simulator answers only questions actually asked. After execution, the simulator comments on the answer. PAHF checks for new information and adds or merges preferences in per-user SQLite memory, retrieving with source DRAGON encoders. The host receives the summary and any current question and answer. Neither learner sees hidden profiles or evaluation judgments, uses the shared extractor, or replaces edits or reactions with scalar configuration rewards. Different feedback formats prevent attributing cross-method differences to update rules alone.

Source checks. We pin CIPHER at `85c306186781` and PAHF at `7a11213360a8`. An indentation-only repair in an unused PAHF FAISS implementation permits import; its SQLite and DRAGON code are unchanged. Adapters change task-domain wording and transport and invoke native SDK hosts. Source-routine, encoder, retrieval, memory-write, and state-restoration checks validate these integrations, not reproduction of the source papers' original-domain results.

Final learning comparison. Each method/SDK/user/replicate has separate memory and a trajectory of 32 feedback opportunities: eight original training fixtures and 24 from the same deterministic generator, all disjoint from held-out tasks. Complete state is copied and frozen after eight opportunities. At the final checkpoint, updated32 and this frozen8 copy execute all eight held-out tasks afresh, without feedback, memory updates, or user questions. The primary contrast is updated32 minus frozen8 within each method, using these fresh executions rather than earlier checkpoint-eight answers.

All $40{,}320$ scheduled positions, including failures, are accounted for; $38{,}901$ reach host execution. Audits verify $18{,}480$ checkpoint states and unchanged state at all $22{,}400$ held-out positions. Style judging covers $8{,}960$ final updated32/frozen8 positions. Judgments for $13{,}440$ intermediate positions at 0, 8, and 16 opportunities are deferred; no intermediate learning curve is reported. CIPHER receives feedback on $8{,}755/8{,}960$ training opportunities and PAHF on $7{,}961/8{,}960$; PAHF also receives 378 answers to 381 questions. Scheduled opportunities need not yield feedback or updates.

Outcomes and judges. Preferred-style match measures the actual held-out answer: full match is one, and partial match or contradiction is zero. Inapplicability, unobservable behavior, host failure,

and unavailable judgments remain missing. Native task success requires successful host execution and a correct submitted result. The profile-blind task judge and local permission checks are separate. Task-and-style success requires native task success, an observed full style match, and no observed local permission violation; it is not the earlier five-preference joint success.

Host, learner, and simulator use separate Sonnet 4.6 calls. GPT-6 Astra is primary; secondary Sonnet shares their model family. Judges see no method or condition labels or private simulator reasoning; only preference judgments receive profiles. Astra passes all 36 engineered rubric checks; offline decoding of saved Sonnet calibration responses yields 30 passes, including one of three memory-evidence checks. These checks are not human validation, and same-family agreement does not establish accuracy.

For the instruction study, $758/10{,}405$ first-round Astra judgments succeed; failures of unknown cause leave overall style contrasts unestimable. Sonnet attempts all $10{,}405$ inputs: $9{,}935$ are parsed and 470 fail. For learning, Astra stops after 199 inputs (191 parsed, eight HTTP 429 failures), leaving $16{,}856$ unissued. Sonnet attempts all $17{,}055$: $15{,}599$ are parsed and $1{,}456$ fail. A logged continuation changes only the transport-error stop rule; prompts and the one-attempt-per-input rule remain unchanged.

Offline decoding recovers 437 instruction-study and $1{,}396$ learning-study task judgments from saved Sonnet text. It also flags four and eight originally accepted preference outputs, respectively, with duplicate JSON keys. Reported estimates retain the original parser; no judgment labels are replaced. Astra/Sonnet style agreement is 65.9% over 461 shared answers in the instruction study and 67.0% over 112 in learning (Cohen's $\kappa = 0.506$ and 0.467). This limited overlap does not validate agreement across the full studies.

Paired comparisons. Paired task differences are averaged within 28 equally weighted fixed SDK/user groups. We bootstrap the ten replicate trajectories within each group, keeping tasks and paired conditions together ($20{,}000$ draws; seed 81271). Each study's two primary style contrasts use nominal 97.5% individual intervals for a 95% Bonferroni family; other intervals are pointwise 95% and descriptive.

Observed-pair estimates accompany observable counts, confirmed matches over all scheduled positions, and bounds setting missing binary outcomes to zero or one. Any group without observed pairs makes the estimate unavailable, not reweighted. An interval containing zero establishes neither equivalence nor inability to learn. Inference is conditional on these profiles, tasks, SDKs, and models. Calls, tokens, latency, and missing usage are recorded by role; unknown usage is not zero.

User differences and missing outcomes. In the instruction study, no observable *concise/direct* answer fully matches its style under Sonnet; all observable *detailed/analytical* answers do. Both contrasts are zero for these users. Gains come from *structured/operator* and *plain/cautious*. Missing-outcome bounds are $[24.0, 28.1]$ points versus no instruction and $[23.9, 28.1]$ versus opposite instructions; for continued learning they are $[-2.54, 1.03]$ for CIPHER and $[4.91, 13.75]$ for PAHF. These are not confidence intervals. Tables 29 and 30 report paired coverage, other endpoints, and channel counts.

Table 29: Endpoint differences and paired coverage in the follow-ups. Differences are in percentage points. Style uses 97.5% intervals: each study's two primary contrasts form a nominal 95% family within each judge. Other intervals are pointwise 95%. Obs. is the number of pairs observed in both conditions, out of 2,240 scheduled pairs per contrast. Task + style requires those two outcomes and local permission compliance, not all five preferences. Missing-outcome bounds remain in the accompanying CSV/JSON. Native task and permission checks do not depend on the LLM judge.

Comparison / judge	Endpoint	Difference [interval]	Obs.	Total
Accurate vs. no instruction				
Astra	Style	*n/a* [*n/a*]	10	2240
Native	Task	-2.4 [-3.3, -1.5]	2240	2240
Astra	Task + style	*n/a* [*n/a*]	37	2240
Astra	Task judge	*n/a* [*n/a*]	16	2240
Sonnet 4.6	Style	+26.8 [25.4, 28.1]	2158	2240
Sonnet 4.6	Task + style	+26.2 [25.0, 27.5]	2213	2240

Comparison / judge	Endpoint	Difference [interval]	Obs.	Total
Sonnet 4.6	Task judge	-2.4 [-3.4, -1.3]	1866	2240
Accurate vs. opposite instruction				
Astra	Style	*n/a* [*n/a*]	13	2240
Native	Task	+0.0 [-1.0, 1.0]	2240	2240
Astra	Task + style	*n/a* [*n/a*]	43	2240
Astra	Task judge	*n/a* [*n/a*]	16	2240
Sonnet 4.6	Style	+26.9 [25.5, 28.2]	2154	2240
Sonnet 4.6	Task + style	+27.7 [26.4, 28.9]	2211	2240
Sonnet 4.6	Task judge	-0.7 [-1.8, 0.4]	1758	2240
CIPHER: 32 vs. frozen at 8				
Sonnet 4.6	Style	-0.4 [-1.3, 0.5]	2163	2240
Native	Task	+1.1 [-0.1, 2.3]	2240	2240
Sonnet 4.6	Task + style	+0.2 [-1.0, 1.3]	2206	2240
Sonnet 4.6	Task judge	+0.5 [-0.9, 1.9]	1641	2240
Native	Permissions	+0.0 [0.0, 0.0]	2231	2240
PAHF: 32 vs. frozen at 8				
Sonnet 4.6	Style	+9.7 [5.8, 13.6]	2096	2240
Native	Task	+9.9 [7.5, 12.1]	2240	2240
Sonnet 4.6	Task + style	+12.9 [9.6, 16.2]	2224	2240
Sonnet 4.6	Task judge	+8.0 [5.2, 10.8]	1466	2240
Native	Permissions	+0.0 [0.0, 0.0]	2133	2240

Table 30: Secondary Sonnet 4.6 judgments: full matches/observable responses. Each condition has 2,240 scheduled positions. CIPHER and PAHF compare state updated through 32 opportunities with state frozen after 8; both execute afresh at the final checkpoint. Unobservable preferences are not scored as failures; n/a means no observable responses. These are descriptive counts, not paired effect estimates or confidence intervals. Sonnet shares the host's model family.

	Direct instructions			CIPHER		PAHF	
Channel	Accurate	None	Opposite	updated32	frozen8	updated32	frozen8
Style	1129/2191	549/2198	549/2196	1114/2199	1132/2201	948/2145	735/2137
Tool	1873/2191	1655/2198	1621/2196	1815/2199	1815/2201	1783/2145	1668/2137
Memory	n/a	n/a	n/a	n/a	n/a	0/1	n/a
Clarification	24/43	0/5	9/34	19/42	19/50	68/111	64/129
Tone	1545/2191	628/2198	676/2196	1269/2199	1278/2201	1222/2145	946/2137

Table 31: Observed resource use in the follow-ups. Learning execution totals include training and unjudged intermediate checkpoints. Calls are model requests, with one application invocation per unique judge input within each judging round. Token totals are in millions; Unknown counts calls without usage data, not zero-cost calls. Seconds are medians per host execution, per completed learner/simulator call, or per successfully parsed judge call. These different units cannot be summed into end-to-end latency. Some wall-clock measurements can include a 30-second coordinator pause during a proxy cooldown. Stopped judging rounds count only issued calls, not unissued inputs; their latencies are not summarized. No complete provider bill was returned.

Study / role	Calls	In (M)	Out (M)	Unknown	Seconds
Delivery / Host	17,356	19.747	5.360	174	17.60
Delivery / Astra	10,405	7.091	0.227	9,647	7.34
Delivery / Sonnet 4.6	10,405	17.579	4.578	3	9.60
Learning / Host	103,085	123.954	29.465	767	16.32
Learning / Learner	76,022	24.244	17.374	1,276	1.81
Learning / Simulator	21,453	21.438	2.790	125	2.52
Learning / Sonnet 4.6	17,055	27.532	7.476	3	9.54
Learning / Astra (stopped)	199	1.804	0.059	8	*n/a*