GRASP: Gated Regression-Aware Skill Proposer for Self-Improving LLM Agents

Johannes Moll[1], **Jean-Philippe Corbeil**[2], **Jiazhen Pan**[1], **Martin Hadamitzky**[1], **Daniel Rueckert**[1], **Lisa Adams**[1,*], **Keno Bressem**[1,*]
[1]Technical University of Munich and TUM University Hospital, [2]Microsoft Healthcare & Life Sciences
Correspondence: johannes.moll@tum.de

github.com/jomoll/GRASP jomoll.github.io/GRASP

Abstract

LLM agents acting in structured environments fail in operational rather than conversational ways, and reliability depends on procedural knowledge of the environment. Prior self-improvement methods accumulate natural-language guidance without checking that each new item preserves previously correct behavior, so a note that fixes one trajectory can silently regress another. We introduce GRASP (Gated Regression-Aware Skill Proposer), which treats agent improvement as a sequence of edits to a bounded skill library, admitting each candidate only if it produces a net improvement on a balanced held-out probe under a hard regression budget. We evaluate GRASP across five base models (gpt-oss-120b, DeepSeek V4 Flash, Gemini 3.1 Flash Lite, GPT-4.1, GPT-5.4) on two FHIR-based clinical benchmarks. On MedAgentBench, GRASP lifts gpt-oss-120b from 40.6% to 88.8%, exceeds the strongest of five self-improvement baselines by 21.0 points, and improves every other base model by 17.2 to 40.3 points. Ablations attribute the gain to comparative proposal generation, the acceptance gate, and the hard regression budget rather than to skill writing itself, which without validation is no better than using no skills. The mechanism generalizes beyond the clinical domain, improving agents on three of four non-clinical environments and remaining flat only where the action space is open-ended. Frozen libraries transfer across models, where skills from a stronger model improve weaker executors beyond what they learn for themselves while the reverse does not, an asymmetry that no ungated baseline reproduces.

1 Introduction

Structured environments such as electronic health records, databases, and web interfaces impose strict operational rules on LLM agents. Violations of these rules produce recurring errors that often go uncorrected. A clinical assistant working with an EHR, for instance, must retrieve the right resources, apply temporal eligibility criteria, avoid duplicate medication orders, and verify that changes were written (Jiang et al., 2025; Moll et al., 2026). These failures reflect the structure of the environment rather than the particular case, so reliability depends on procedural knowledge of how to act in a specific environment, not on general language ability (Hsiao et al., 2025; Roig, 2025).

To acquire this knowledge, prior self-improvement methods accumulate natural-language guidance at inference, either as reflections on individual failures (Shinn et al., 2023) or as rules extracted across many (Zhao et al., 2024), admitting each item as soon as it is produced (Appendix A). Without a check against existing behavior, a note that fixes one trajectory can silently regress another, and over time monotonic accumulation dilutes task-relevant content and degrades the agent's context (Shi et al., 2023; Liu et al., 2024a). Avoiding both failures requires held-out validation of each candidate against previously correct behavior, together with a bounded library that supports modification and removal alongside addition.

We introduce GRASP (**G**ated **R**egression-**A**ware **S**kill **P**roposer), which treats agent self-improvement as a sequence of validated edits to a small library of behavioral instructions, or *skills*, injected into the agent's context at inference (Wang et al., 2023). After each batch of episodes, failed trajectories are grouped by failure mode, and candidate edits are proposed for the most frequent modes, each adding, modifying, or removing a skill. Each candidate is re-run on a balanced probe of previously failing and previously passing examples, and accepted only if it fixes more failures than it causes regressions, with no new regression beyond the existing baseline. The mechanism requires no parameter updates and the library remains small,

[*]Equal senior authorship.

versioned, and reversible.

We evaluate GRASP across five language models (gpt-oss-120b, DeepSeek V4 Flash, Gemini 3.1 Flash Lite, GPT-4.1, GPT-5.4) on two FHIR-based clinical agent benchmarks, MedAgentBench (Jiang et al., 2025) and MedAgentBench-v2 (Chen et al., 2025), with supporting evaluation on FHIR-AgentBench (Lee et al., 2025) and four non-clinical environments (Shridhar et al., 2021; Yao et al., 2022a; Liu et al., 2024b). Our main results are as follows.

- GRASP substantially outperforms a no-skills agent and five self-improvement baselines spanning memory, rule-extraction, and skill-library methods on MedAgentBench, an advantage that holds across all five base models.
- Ablations attribute the gain to comparative proposal generation, the acceptance gate, and the hard regression budget, showing that without them the skill-writing pipeline is no better than using no skills, which localizes the improvement to validated editing rather than skill writing.
- The mechanism generalizes beyond the clinical domain, helping in non-clinical environments where tasks recur with verifiable structure and remaining flat only where the action space is open-ended.
- Frozen skill libraries transfer across models and related benchmarks. A stronger writer improves weaker executors beyond what they learn for themselves while the reverse does not, an asymmetry that no ungated baseline reproduces, indicating that the libraries encode environment-specific procedural knowledge rather than model-specific patterns.

All code, learned skill libraries, and per-seed results are available at `https://github.com/jomoll/GRASP`.

2 GRASP: Gated Regression-Aware Skill Proposer

GRASP improves an LLM agent through repeated validated edits to a library of behavioral instructions, with no parameter updates. After each batch of training episodes, failed trajectories are grouped by failure mode, a skill-writing model proposes candidate edits for the most frequent modes, and each candidate is evaluated on a held-out probe before acceptance (Algorithm 1).

2.1 Skills

A *skill* is a structured behavioral instruction injected into the agent's context at inference (Wang et al., 2023), specifying a trigger condition, a behavioral rule, an optional verification step for actions with side effects, and a contrastive example. Skills are Markdown documents with YAML frontmatter, making them auditable and reversible. The library starts empty and is populated by GRASP from training trajectories (Appendix B.4 and B.5).

2.2 Failure-Driven Skill Proposal

After each batch, a skill-writing model observes the failed traces and proposes edits, building on the verbal-feedback paradigm (Shinn et al., 2023; Zhao et al., 2024) but operating on a structured library rather than a flat memory.

Mechanism-specific failure classification. An LLM classifier assigns each failing trace an open-vocabulary label from the action sequence, the reported answer, and the expected answer. Labels must be mechanism-specific, meaning two labels imply two different corrective actions. A trace omitting a required date filter is labeled `date_filter_omitted` rather than `wrong_answer`, since the former points to a corrective skill and the latter does not. Labels from prior epochs are presented as candidates, and new ones are minted when none fits (classifier dynamics in Appendix E.1). Failures are grouped by label with no minimum group size, so a single failure can prompt a proposal if its mechanism is distinct.

Grouped proposal generation. The framework generates K proposals per update by cycling over failure-mode groups from largest to smallest. Each call to the skill writer provides the failing traces from one group, a sample of passing traces, active skill summaries with provenance and effectiveness statistics, and the labels of other active failure modes to discourage regressions on unrelated tasks. The skill writer returns an `ADD`, `MODIFY`, or `REMOVE` edit. To limit the number of skills, `ADD` is blocked at capacity unless a paired `REMOVE` frees a slot.

2.3 Regression-Aware Selection

Each candidate is evaluated against a balanced probe before being committed, following the

held-out validation strategy familiar from prompt-optimization frameworks (Khattab et al., 2023). Probe examples are drawn exclusively from the development split, with validation and test examples never entering training (full per-batch loop in Appendix B.1).

Probe construction. The probe contains up to $N/2$ previously-failing and $N/2$ previously-passing samples from episodes completed earlier in the current epoch, stratified by task type, where *previously-failing* and *previously-passing* refer to per-sample correctness recorded under the library state in effect when each sample was originally run. For the first batch of each epoch, the previous epoch's samples serve as the baseline. The probe is always out-of-sample relative to the batch that generated the proposals.

Acceptance criterion. Probe labels from earlier batches reflect library states that may differ from the current library S, so we re-run S on the probe for a fresh causal baseline. Let $\mathcal{P}_{\text{fail}}$ and $\mathcal{P}_{\text{pass}}$ denote the previously-failing and previously-passing subsets. Running S yields baseline counts $F_0 = |\{x \in \mathcal{P}_{\text{fail}} : x \text{ passes under } S\}|$ and $R_0 = |\{x \in \mathcal{P}_{\text{pass}} : x \text{ fails under } S\}|$. For a candidate library S^c, the analogous counts $F(c)$ and $R(c)$ are computed on the same probe. A candidate is accepted only if

$$\big(F(c)-F_0\big)-\big(R(c)-R_0\big) > 0 \quad \text{and} \quad R(c) \le R_0. \tag{1}$$

The first condition requires a net improvement in fixes over regressions, the second is a hard regression budget. The highest-scoring candidate satisfying both is applied, otherwise the library is left unchanged. If the winning candidate introduces any regression, a contrastive revision step prompts the skill writer to narrow the trigger or add a guard clause, and the revision replaces the original only if its adjusted score is strictly higher and still satisfies the regression budget (Appendix B.2).

3 Experimental Setup

3.1 Benchmarks

Our primary evaluation uses three FHIR-based clinical agent benchmarks. **MedAgentBench** (Jiang et al., 2025) requires an agent to query a live FHIR server, reconcile medication orders, and verify writes through follow-up reads. **MedAgentBench-v2** (Chen et al., 2025) extends it with multi-step decision tasks, coordinated writes, and clinical safety protocols. **FHIR-AgentBench** (Lee et al., 2025) evaluates structured clinical QA and tool use on an independent FHIR environment. Each is split into disjoint development, validation, and held-out test sets (Table 8), with additional out-of-distribution (OOD) splits for the two MedAgentBench variants. We report exact-match accuracy against ground-truth answers and FHIR-server state (scoring details in Appendix D). For evidence of generality, we also evaluate on four AgentBench (Liu et al., 2024b) environments, ALFWorld (Shridhar et al., 2021), WebShop (Yao et al., 2022a), OS Interaction, and DBBench.

3.2 Models

We evaluate GRASP with five base models spanning open and proprietary families and a range of capability. The open-source models are gpt-oss-120b, an open model that fits on a single H200 GPU, and DeepSeek V4 Flash, a larger long-context alternative, both self-hosted. The proprietary models are Gemini 3.1 Flash Lite, GPT-5.4 (low reasoning effort), and GPT-4.1, three frontier models accessed via official provider APIs. Exact handles, sizes, precision settings, and access dates appear in Appendix C. Within a run, the same model serves as both executing agent and skill-writer.

3.3 Baselines

We compare GRASP against a no-skills baseline and five self-improvement methods, all using the same base agent, decoding settings, and training episodes. For every method, the same model serves as executing agent and as the LLM that writes notes, rules, or skills.

Sequential memory (Chen et al., 2025) appends a natural-language correction note to a flat memory block after each failing episode. Notes are task-specific conditional instructions, and the memory grows unbounded with no acceptance gate.

Batch memory is identical but updated once per batch, matching GRASP's cadence and isolating the acceptance gate from update frequency.

ExpeL (Zhao et al., 2024) extracts contrastive natural-language rules by pairing each failing trace with its nearest successful trace and prompting an LLM to propose AGREE/EDIT/REMOVE/ADD operations on a bounded rule list. Rules are scored by accumulated operation counts and capped at 20.

Evo-MedAgent (Shen et al., 2026) maintains episodic and semantic memory stores with top-k retrieval and per-rule utility tracking, but no probe-based acceptance gate.

SkillX (Wang et al., 2026) extracts skills from successful trajectories via LLM decomposition and retrieves them by token overlap, with no regression control.

All methods inject into the same field of the task prompt, isolating the learned content and update rule from placement effects (Appendix B.5).

3.4 Training and Evaluation Protocol

Training. Each method trains for 5 epochs over the dev split with a batch size of 48 episodes. After every epoch, validation accuracy is computed, the learned state is checkpointed when it improves, and the best checkpoint is restored after the final epoch. The test and OOD splits are evaluated once on this checkpoint and never participate in training, checkpointing, or hyperparameter selection. For GRASP, the probe contains 36 examples from episodes completed earlier in the epoch, and the skill writer generates 4 candidate edits per batch.

Cross-model transfer. For each transfer cell, the source model M_s runs a complete GRASP training cycle as both executing agent and skill-writer. The resulting library is selected by best-validation on M_s, frozen, and applied to the target model M_t at test time, with no further training, retrieval refitting, or library editing.

Compute and context budget. Methods differ in where they spend training compute and how much context they inject at inference. GRASP uses roughly $3\times$ more training-time LLM calls per batch than the simplest memory baselines, almost entirely from probe-based validation, since each batch evaluates $K + 1$ library variants on a 36-episode probe. At inference the picture inverts. GRASP libraries average 5 skills and 5.6k tokens, while the memory and skill-library baselines grow to 34–53k (Appendix Table 6). Only GRASP and ExpeL stay below 10k inference tokens.

Reporting and statistics. Headline numbers are test accuracy unless otherwise indicated, reported as mean $\pm$ standard deviation across seeds. Main results and benchmark transfer use five seeds for open-source models and three for proprietary, while ablations, sensitivity analyses, and model transfer experiments use three seeds. For every major claim in Section 4, Appendix E.4 reports the mean difference Δ with a 95% bootstrap CI from 10,000 resamples, Cohen's d, and permutation p-values.

3.5 Ablations

We ablate eight design choices of GRASP on gpt-oss-120b on MedAgentBench with three seeds. *No failure grouping* passes all failed traces to the skill-writer at once instead of grouping them by mechanism. *No regression budget* accepts edits on positive net fixes without enforcing $R(c) \leq R_0$. *Fixes-only* scores candidates on previously failing examples alone, omitting the passing examples that detect regressions. *Append-only* restricts the skill-writer to `ADD`, removing modification and deletion. Two *no acceptance gate* variants drop probe validation entirely and apply every edit unconditionally with FIFO eviction, at $K{=}4$ and $K{=}1$. Finally, two *matched-compute* variants separate the probe's validation signal from the cost of running it, scoring all $K{+}1$ candidates on the full probe as GRASP does but selecting the applied edit without the scores, one by the skill-writer's preference and one at random. One-at-a-time sweeps over B, N, K, and the invalid-action weight λ appear in Appendix E.2.

4 Results

4.1 Benchmark Results

Table 1 reports test accuracy on MedAgentBench and MedAgentBench-v2. On MedAgentBench, GRASP is the strongest method on every model and on both the in-domain and OOD splits. In-domain, it lifts gpt-oss-120b from 40.6% to 88.8%, a gain of 48.2 points over the no-skills baseline and 21.0 over the strongest baseline (Evo-MedAgent, 67.8%), and the pattern replicates across families, GPT-5.4 (low) from 45.1% to 85.4% (+40.3), GPT-4.1 from 45.5% to 84.9% (+39.4), DeepSeek V4 Flash from 47.7% to 70.0% (+22.3), and Gemini 3.1 Flash Lite from 54.2% to 71.4% (+17.2). On the OOD split the lead is larger, reaching 56.3% on gpt-oss-120b against 31.3% for the best baseline (a 25.0-point gain), with similarly large margins on DeepSeek (52.3% vs 20.0%), GPT-4.1 (27.2% vs 2.2%), and GPT-5.4 (80.6% vs 16.7%), and a narrower lead on Gemini (41.7% vs 38.3%). On MedAgentBench-v2 the picture is more mixed. GRASP leads the in-domain test split on three of the five models, improving gpt-oss-120b from 61.1% to 76.9% (+15.8) and exceeding the best baseline

Table 1: Main results on two FHIR clinical benchmarks. Val* is the best validation accuracy across epochs and Test is the held-out test accuracy using the best-validation checkpoint, OOD is held-out task types. Open-source results are across five seeds, proprietary results across three. Bold marks the best score in each column within a model.

		MedAgentBench			MedAgentBench-v2		
Model	Method	Val*	Test	OOD	Val*	Test	OOD
gpt-oss-120b	No skills	36.4 ±5.6	40.6 ±3.9	8.7 ±3.2	57.6 ±6.5	61.1 ±4.6	75.2 ±7.3
	Seq. Memory	38.5 ±12.4	41.2 ±5.5	16.7 ±15.6	54.2 ±10.2	53.8 ±11.6	82.3 ±4.5
	Batch Memory	36.8 ±5.3	34.4 ±8.0	13.0 ±8.1	62.2 ±6.6	59.4 ±4.9	78.8 ±12.7
	ExpeL	47.5 ±6.2	49.1 ±6.3	12.0 ±5.2	65.5 ±1.9	67.8 ±2.8	86.3 ±1.4
	Evo-MedAgent	62.5 ±17.0	67.8 ±14.1	31.3 ±21.7	65.3 ±3.5	66.9 ±2.6	86.7 ±0.0
	SkillX	51.5 ±4.1	53.1 ±4.6	21.3 ±3.6	67.5 ±3.7	64.1 ±4.2	**87.1 ±0.8**
	GRASP (ours)	**86.0 ±4.4**	**88.8 ±5.8**	**56.3 ±14.5**	**77.0 ±3.1**	**76.9 ±4.0**	86.0 ±1.5
DeepSeek V4 Flash	No skills	41.8 ±2.5	47.7 ±2.1	3.1 ±0.7	23.8 ±3.0	20.4 ±4.1	83.6 ±1.3
	Seq. Memory	33.0 ±17.5	35.0 ±16.5	8.7 ±11.9	17.8 ±4.2	13.8 ±7.8	65.3 ±13.7
	Batch Memory	36.5 ±6.7	37.8 ±6.4	3.3 ±4.7	17.8 ±5.6	16.6 ±5.3	57.3 ±28.9
	ExpeL	50.7 ±2.1	55.6 ±2.8	2.3 ±1.5	31.5 ±5.4	22.8 ±4.1	**85.0 ±1.2**
	Evo-MedAgent	52.5 ±5.2	51.9 ±6.5	20.0 ±25.2	25.5 ±4.3	25.3 ±3.9	42.7 ±9.2
	SkillX	55.0 ±0.9	55.9 ±2.8	6.3 ±2.7	19.5 ±4.6	14.7 ±3.9	28.0 ±11.0
	GRASP (ours)	**68.8 ±3.2**	**70.0 ±5.5**	**52.3 ±14.7**	**36.8 ±13.5**	**30.9 ±14.7**	67.0 ±22.5
Gemini 3.1 Flash Lite	No skills	49.0 ±1.5	54.2 ±1.5	18.1 ±3.5	47.6 ±2.5	44.6 ±2.6	76.3 ±3.0
	Seq. Memory	48.8 ±1.3	54.2 ±2.4	8.9 ±11.1	60.0 ±2.5	50.5 ±7.0	56.1 ±17.0
	Batch Memory	29.2 ±13.8	39.6 ±22.2	8.3 ±10.1	55.4 ±4.7	51.6 ±5.6	73.9 ±3.8
	ExpeL	51.2 ±0.0	53.6 ±1.8	17.2 ±2.5	58.3 ±4.4	55.7 ±6.3	77.2 ±1.0
	Evo-MedAgent	49.6 ±1.9	55.2 ±1.8	15.6 ±2.5	62.9 ±5.2	54.2 ±3.3	72.2 ±2.5
	SkillX	54.2 ±1.4	54.2 ±3.6	38.3 ±1.7	65.4 ±5.2	65.1 ±3.3	66.1 ±9.2
	GRASP (ours)	**66.7 ±2.6**	**71.4 ±1.8**	**41.7 ±20.9**	**68.8 ±5.7**	**67.2 ±2.7**	**80.0 ±3.3**
GPT-4.1	No skills	40.1 ±10.0	45.5 ±0.5	0.3 ±0.6	65.9 ±2.2	70.5 ±2.8	**84.9 ±0.7**
	Seq. Memory	42.5 ±0.0	45.3 ±0.0	0.6 ±1.0	68.8 ±1.2	67.7 ±0.9	83.3 ±1.7
	Batch Memory	46.2 ±8.8	47.4 ±9.5	2.2 ±2.5	68.3 ±5.1	73.4 ±1.6	83.9 ±1.0
	ExpeL	42.5 ±0.0	45.3 ±0.0	0.6 ±1.0	72.5 ±2.2	72.9 ±2.4	84.4 ±1.0
	Evo-MedAgent	42.5 ±0.0	45.3 ±0.0	0.6 ±1.0	66.7 ±2.6	69.3 ±3.9	82.8 ±3.8
	SkillX	33.8 ±12.1	38.5 ±13.1	0.0 ±0.0	72.1 ±2.6	**74.0 ±2.4**	80.6 ±1.0
	GRASP (ours)	**86.2 ±2.5**	**84.9 ±0.9**	**27.2 ±16.9**	**73.8 ±3.3**	70.3 ±1.6	82.8 ±1.0
GPT-5.4 (low)	No skills	40.6 ±10.2	45.1 ±1.1	1.7 ±1.6	63.5 ±3.1	59.9 ±2.0	82.7 ±3.4
	Seq. Memory	46.7 ±7.2	47.4 ±5.0	15.6 ±25.5	62.1 ±8.1	60.9 ±1.6	81.7 ±4.4
	Batch Memory	47.9 ±0.7	46.4 ±0.9	16.7 ±3.3	62.9 ±11.2	**63.5 ±1.8**	78.3 ±10.9
	ExpeL	45.0 ±1.3	43.8 ±1.6	11.7 ±3.3	70.4 ±0.7	60.4 ±2.4	86.1 ±1.0
	Evo-MedAgent	42.5 ±2.2	43.8 ±4.1	1.7 ±1.7	68.8 ±3.3	62.5 ±1.6	80.0 ±1.7
	SkillX	45.4 ±3.1	44.8 ±0.9	3.3 ±2.9	68.8 ±1.2	57.8 ±1.6	**86.7 ±1.7**
	GRASP (ours)	**84.6 ±10.5**	**85.4 ±10.4**	**80.6 ±6.7**	**71.2 ±1.8**	63.3 ±3.3	80.0 ±4.7

(ExpeL, 67.8%) by 9.1 points, with smaller leads on DeepSeek (30.9% vs 25.3%) and Gemini (67.2% vs 65.1%). On GPT-4.1 and GPT-5.4 it is within noise of the strongest baseline, where the dominant failure is exhausting the 8-action budget on two paginated-search task types and no method beats no-skills (Appendix E.6). On the OOD split the methods are closely matched and GRASP does not lead overall, scoring 86.0% on gpt-oss-120b against 87.1% for the best baseline (SkillX) with overlapping standard deviations. We make no superiority claim on v2 OOD.

4.2 Learning Stability

Figure 1 shows validation accuracy across training epochs on MedAgentBench for gpt-oss-120b. GRASP improves almost monotonically and stabilizes above 85% from epoch 3. Sequential and Batch memory regress below the no-skills baseline as memory accumulates, ExpeL and SkillX stagnate near baseline, and Evo-MedAgent reaches competitive validation accuracy on some epochs but oscillates by more than 20 points between adjacent checkpoints. Seed variance corroborates the dynamic, with Evo's standard deviation across five seeds of 14.1 against GRASP's 5.8 at a 21.0-point higher mean.

4.3 Cross-Model Transfer

Table 2 reports cross-model transfer on MedAgentBench. Each column fixes an executor model and each row identifies the source model that produced the frozen skill library applied at inference. Diagonal entries reproduce the in-domain results of

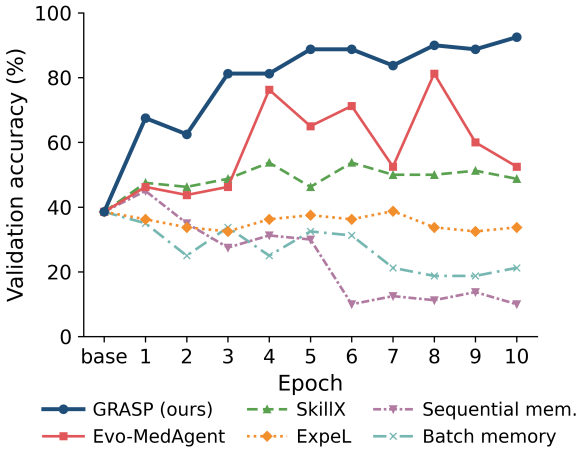

Figure 1: Validation accuracy across training epochs on MedAgentBench (gpt-oss-120b), one representative seed per method.

Table 2: Cross-model skill transfer on MedAgentBench across three seeds. Rows indicate the skill library's training model; columns indicate the executor at evaluation time. Diagonal entries (source equals executor) reproduce the GRASP rows of Table 1; off-diagonal entries apply frozen skills without further training.

		Executor model		
Skill source ↓	**Split**	**gpt-oss**	**Gemini 3.1**	**GPT-5.4**
None	Test	40.6 ±3.9	54.2 ±1.5	45.1 ±1.1
	OOD	8.7 ±3.2	18.1 ±3.5	1.7 ±1.6
gpt-oss	Test	**88.8 ±5.8**	**79.7 ±7.8**	72.4 ±15.8
	OOD	56.3 ±14.5	36.1 ±14.0	51.7 ±2.9
Gemini 3.1	Test	65.6 ±8.3	71.4 ±1.8	69.3 ±6.5
	OOD	57.8 ±31.9	41.7 ±20.9	58.9 ±37.5
GPT-5.4	Test	76.0 ±7.7	76.6 ±10.9	**85.4 ±10.4**
	OOD	**77.8 ±1.0**	**71.1 ±1.9**	**80.6 ±6.7**

Table 1, and off-diagonal entries apply a library to a different executor without further adaptation. The clearest effect is on OOD task types, where libraries written by GPT-5.4 (low) improve every executor over its own self-trained library. Applied to gpt-oss-120b, GPT-5.4 skills reach 77.8% OOD against 56.3% from gpt-oss-120b's own library, a 21.5-point gain, while trading 12.8 points in-domain (88.8% to 76.0%). Applied to Gemini 3.1 Flash Lite, they lift OOD from 41.7% to 71.1% (+29.4) and in-domain test from 71.4% to 76.6% (+5.2). GPT-5.4 is the strongest cross-model source on every transfer cell except Gemini in-domain test, where the gpt-oss library is marginally stronger. Libraries from weaker source models applied to stronger executors are consistently worse than self-training. Off-diagonal transfer with Gemini as the source is high-variance (OOD standard deviations above 30 points) and we draw no conclusions from those cells.

Table 3: Cross-benchmark skill transfer on gpt-oss-120b. Rows indicate the skill library's training benchmark, and columns indicate the evaluation target. Diagonal entries (source equals target) reproduce the gpt-oss-120b row of Table 1, off-diagonal entries apply frozen skills without further training. DBBench (AgentBench) is a non-clinical control and FHIR-AB has no OOD split.

		Target benchmark		
Skill source ↓	**Split**	**MAB**	**MAB-v2**	**FHIR-AB**
None (baseline)	Test	40.6 ±3.9	61.1 ±4.6	51.5 ±2.2
	OOD	8.7 ±3.2	75.2 ±7.3	—
MAB	Test	**88.8 ±5.8**	**78.1 ±7.8**	47.2 ±6.7
	OOD	**56.3 ±14.5**	**86.7 ±0.0**	—
MAB-v2	Test	51.6 ±6.8	76.9 ±4.0	45.7 ±9.4
	OOD	34.4 ±22.8	86.0 ±1.5	—
FHIR-AB	Test	45.3 ±4.4	67.2 ±8.8	**58.2 ±3.4**
	OOD	15.8 ±13.0	81.7 ±0.0	—
DBBench	Test	38.5 ±2.4	57.8 ±7.2	48.3 ±3.1
	OOD	1.1 ±1.0	75.0 ±10.4	—

4.4 Cross-Benchmark Transfer

Table 3 reports cross-benchmark transfer on gpt-oss-120b. Each row fixes a target benchmark and split, and each column applies a frozen library trained on a different source. Libraries trained on the DBBench environment (Liu et al., 2024b) serve as a non-clinical control.

Transfer between MedAgentBench and MedAgentBench-v2 is positive in both directions. MedAgentBench skills applied to MedAgentBench-v2 reach 78.1% test and 86.7% OOD, exceeding v2 self-training (76.9% and 86.0%). In reverse, v2 skills applied to MedAgentBench reach 51.6% test against a 40.6% baseline, with positive but high-variance OOD transfer (34.4% against an 8.7% baseline, ±22.8). On FHIR-AgentBench, self-training improves gpt-oss-120b from 51.5% to 58.2% across five seeds ($\Delta = +6.8$, 95% CI $[3.7, 10.0]$), while every library transferred into it degrades the baseline (MedAgentBench 47.2%, MedAgentBench-v2 45.7%, DBBench 48.3%). Transfer in the reverse direction, out of FHIR-AgentBench onto the MedAgentBench family, does not show this degradation. The DBBench control transfers negatively or neutrally to every clinical target.

Table 4: Ablation study on MedAgentBench with gpt-oss-120b across three seeds. *w/o regression budget* keeps comparative scoring of $K{=}4$ candidates but drops the hard budget $R(c) \leq R_0$. *w/o acceptance gate* drops probe validation entirely, applying edits unconditionally. *matched compute* runs the full probe as GRASP does but selects without the probe scores, by the skill-writer's preference or at random.

Method	Val*	Test
GRASP (full)	86.0 ±4.4	88.8 ±5.8
w/o failure grouping	82.1 ±5.2	84.4 ±3.1
w/o regression budget	81.2 ±6.5	81.8 ±1.8
fixes-only selection	78.8 ±14.4	80.2 ±13.7
append only	73.3 ±8.3	80.2 ±9.4
w/o acceptance gate ($K{=}4$)	65.4 ±9.7	63.5 ±3.9
w/o acceptance gate ($K{=}1$)	38.8 ±8.2	40.1 ±11.3
matched compute (proposer)	67.1 ±14.8	70.8 ±14.0
matched compute (random)	62.1 ±7.5	67.2 ±10.2

4.5 Ablations

Table 4 ablates the GRASP design choices defined in Section 3.5 on MedAgentBench with gpt-oss-120b. The results separate into three tiers. The full method reaches 88.8% test accuracy. Removing any single component while keeping the acceptance gate stays above 80%, above every comparator in Table 1. Removing failure grouping drops to 84.4% (-4.4), the regression budget to 81.8% (-7.0), and fixes-only and append-only both reach 80.2% (-8.6), with fixes-only the least stable at ± 13.7.

Removing the acceptance gate entirely is more damaging. Applying every proposed edit unconditionally falls to 63.5% (-25.3) at $K{=}4$ and to 40.1% (-48.7) at $K{=}1$, the latter matching the no-skills baseline (40.6%) within seed noise. The matched-compute variants spend GRASP's full per-batch probe budget but select the applied edit without the probe scores, reaching 70.8% (proposer) and 67.2% (random), within the variance of the no-gate $K{=}4$ variant (63.5%).

4.6 Non-Medical Environments

Table 5 reports test accuracy on four non-clinical AgentBench environments using gpt-oss-120b across three seeds. GRASP produces a large gain on ALFWorld (+28.4) and a substantial gain on WebShop (+20.6), where tasks recur with verifiable structure, a smaller gain on DBBench (+5.0), and no gain beyond seed noise on OS Interaction (+0.9), where the action space is open-ended.

Table 5: Non-medical results on four AgentBench (Liu et al., 2024b) environments using gpt-oss-120b across three seeds.

Benchmark	**No skills**	GRASP	**Gain**
ALFWorld	23.3 ±2.9	51.7 ±7.6	+28.4
WebShop	20.7 ±1.2	41.3 ±8.1	+20.6
DBBench	65.6 ±3.9	70.6 ±1.0	+5.0
OS Interaction	48.6 ±2.9	49.5 ±1.7	+0.9

5 Discussion

GRASP treats agent self-improvement as a sequence of validated edits to a bounded library of behavioral skills, each candidate gated by a regression-aware probe of previously failing and passing examples. The design responds to a specific failure mode of prior verbal-feedback methods, which accumulate guidance monotonically without checking that each item preserves existing behavior. Our results indicate that the gain over those methods comes from validated skill *editing* rather than from better skill *writing*. The skill-writer without validation at $K = 1$ reaches 40.1%, matching the no-skills baseline, so skill writing alone is no better than no skills. Generating $K = 4$ candidates without validation reaches 63.5%, so comparative generation contributes substantially even with no gate, and adding the acceptance gate brings the full method to 88.8%. This gain is not an artifact of the probe's compute, since spending the full probe budget but discarding the verdict stays within the variance of the no-gate variant that never runs the probe. Sequential and Batch memory update more often than GRASP yet regress below the no-skills baseline as context accumulates, illustrating that prompt-based agents forget through dilution rather than parameter drift, which a held-out probe naturally detects.

Cross-model and cross-benchmark transfer together indicate that the learned skills encode environment-specific procedural knowledge rather than model-specific in-context patterns or general clinical competence. The clearest signal is on held-out task types, where skills written by GPT-5.4 (low) improve every executor's OOD accuracy over what it learns for itself, while the in-domain effect is smaller and can cost a few points on the strongest executors. The direction is asymmetric. A stronger writer improves a weaker executor, but libraries from weaker writers applied to stronger executors are consistently worse than self-training. This asymmetry is specific to GRASP. The same

protocol applied to all five baselines leaves every one near or below 35% OOD after transfer (Appendix E.5), indicating that the gate, not writer strength, makes a library transferable. A natural reading is that a stronger writer articulates operational facts about the environment that a weaker executor can follow even when it cannot generate them, a distillation-like effect without parameter updates, and the gain concentrating on held-out task types fits this account. Transfer across benchmarks bounds the same knowledge from the other side. Skills move between MedAgentBench and MedAgentBench-v2, which share a tool-calling convention, but degrade FHIR-AgentBench below its own baseline, because a library written against one convention emits malformed calls under FHIR-AgentBench's Python-tool and free-text interface. What transfers is procedural knowledge tied to an environment and its interface, not clinical FHIR competence in general.

The non-medical results delineate where the mechanism applies. GRASP helps most where tasks recur with predictable structure and side effects are directly verifiable (ALFWorld, WebShop), and least where the action space is open-ended and consequences are diffuse, as on OS Interaction, where the gain does not exceed seed noise. DBBench falls between, since only a subset of its query types exhibits the recurring, checkable patterns a skill can target. A separate boundary is set by resource limits, not task structure. On MedAgentBench-v2 the proprietary models fail most often by exhausting the fixed 8-action budget on two paginated-search task types, which no method learned to avoid (Appendix E.6). The common factor across domains is not subject matter but task structure, recurring failure modes paired with a verifiable signal the probe can use to admit or reject an edit.

Two properties make the mechanism practical to deploy. First, bounded libraries outperform unbounded memories at a fraction of the inference cost. GRASP reaches higher test accuracy than every memory baseline while injecting roughly an order of magnitude fewer tokens (Section 3.4), consistent with prior findings that LLMs degrade under irrelevant context (Shi et al., 2023; Liu et al., 2024a). Because the library stays capacity-bounded, this overhead does not grow with training length. Second, the library is auditable by design. Each skill is a versioned Markdown document recording which failure mode prompted it, which probe score justified its acceptance, and which earlier skill it replaced (Appendix F), so a reviewer can trace why an agent behaves as it does and revert a specific behavior without retraining. Both properties matter most where an agent must be inspected and corrected after deployment rather than retrained, as in a clinical setting.

6 Conclusion

GRASP converts agent failures into validated edits to a bounded library of behavioral skills, each candidate gated by a regression-aware probe of previously failing and previously passing examples. Against the strongest of five memory-based baselines, GRASP adds 21.0 points on MedAgentBench, and the gain holds across five base models on two FHIR clinical benchmarks. Ablations localize it to the validation gate rather than the skill-writing process, which alone closes most of the gap against the no-skills baseline. Frozen libraries written by a stronger model improve weaker executors beyond what they learn for themselves while the reverse does not, an asymmetry that no ungated baseline reproduces and that, with the way skills break across interface boundaries, indicates the libraries encode environment-procedural knowledge rather than model-specific patterns. The same mechanism helps across clinical and non-clinical environments where tasks recur with verifiable structure, and is flat only where the action space is open-ended. Treating agent self-improvement as gated editing of a bounded library, rather than monotonic accumulation of guidance, offers a path toward iterative refinement of LLM agents in environments where procedural reliability matters.

Code and Data Availability. We release a unified codebase that implements GRASP and all five comparators against a shared agent and evaluation interface, with benchmark adapters for seven structured-agent environments spanning clinical FHIR and AgentBench tasks (`https://github.com/jomoll/GRASP`). A new self-improvement method can be added through a single update interface and run on any of these benchmarks without further integration. To make every reported result inspectable and reproducible, the repository also includes the learned skill libraries, the frozen transfer libraries, all prompts, and per-seed validation, test, and OOD accuracies for every cell of Tables 1–5.

Acknowledgments. This work was supported by Bayern Innovativ (Bavarian State Ministry of Eco-

nomics) under Grant KK6052501. We also thank François Beaulieu and Paul Vozila of Microsoft Healthcare & Life Sciences for their support of this project.

Limitations

GRASP is evaluated on benchmarks that score procedural reliability against ground-truth FHIR state rather than clinical correctness or patient outcomes. MedAgentBench and its variants use curated patient records, a containerized FHIR server, and exact-match scoring against pre-specified answers. Real clinical environments might differ in ways the benchmarks do not capture, including incomplete and contradictory records, large free-text note volumes, non-standard local codes, multi-system integration, and workflows that depend on context outside the EHR. All clinical evaluation is on English-language FHIR data, and generalization to non-English content and to proprietary EHR systems that do not expose FHIR is open. The skills GRASP learns are policies tuned to a benchmark rather than medical expertise, and translating the mechanism to a clinical setting would require clinician review of each skill, prospective comparison against current practice, and monitoring under live workflow conditions, none of which are reported here.

The validation step is the source of GRASP's stability but also its dominant cost. Each batch evaluates $K + 1$ candidate libraries on a 36-episode probe, contributing roughly 440 agent calls per batch and accounting for the majority of training-time compute. The "no parameter updates" framing should not be read as "no cost". The cost has moved from gradient updates to LLM inference. In settings where dev-split episodes are expensive, for example live clinical environments or slow tool interfaces, the probe size that delivered our reported results may need to be reduced, with the trade-off between probe size and gate reliability characterized in Appendix E.2. GRASP's probe compute is not budget-matched to the baselines, so its advantage over them could in principle reflect the extra compute the gate consumes rather than the gate's decisions themselves. We separate the two within GRASP itself. Our matched-compute ablations (Section 4.5) hold the compute fixed and vary only the gate, spending the full per-batch probe budget but discarding its verdict when selecting an edit. Accuracy stays at the no-gate level, attributing the gain to the validation decision rather than to the probe compute. Whether the gate would also help the baselines if added to them is a separate question, one of generality across methods rather than of GRASP's own mechanism, and we do not

address it here.

The cross-model transfer result is asymmetric. Stronger writers produce libraries that improve weaker executors beyond self-training, but the reverse direction is consistently worse than self-training. A frozen library is therefore not fully model-agnostic in practice, and deploying a library written by one model on a different one requires verifying that the new executor is not weaker on the procedural axes the writer assumed. We do not study how to detect such mismatches automatically, and the present results give no guidance on choosing a writer model for a given target executor beyond "use the strongest available". The asymmetry itself is specific to GRASP. We ran the cross-writer protocol on all five baselines in the two configurations where GRASP shows its strongest effect (GPT-5.4 to Gemini and to gpt-oss-120b), and none reproduces it (Appendix E.5). We do not extend this comparison to the full writer-executor matrix, which we compute only for GRASP.

Three of the five base models we evaluate (GPT-5.4, GPT-4.1, Gemini 3.1 Flash Lite) are accessed through provider APIs that may change without notice. Exact model handles and access dates are reported in Appendix C, and the proprietary rows of our results tables reflect access conditions specified there.

References

Marc-Antoine Allard, Arnaud Teinturier, Victor Xing, and Gautier Viaud. 2026. Experiential reflective learning for self-improving llm agents. *arXiv preprint arXiv:2603.24639*.

Eric Chen, Sam Postelnik, Kameron Black, Yixing Jiang, and Jonathan H Chen. 2025. Medagentbench v2: Improving medical llm agent design. In *Biocomputing 2026: Proceedings of the Pacific Symposium*, pages 354–371. World Scientific.

Xiang Deng, Yu Gu, Boyuan Zheng, Shijie Chen, Sam Stevens, Boshi Wang, Huan Sun, and Yu Su. 2023. Mind2web: Towards a generalist agent for the web. *Advances in Neural Information Processing Systems*, 36:28091–28114.

Yao Fu, Dong-Ki Kim, Jaekyeom Kim, Sungryull Sohn, Lajanugen Logeswaran, Kyunghoon Bae, and Honglak Lee. 2024. Autoguide: Automated generation and selection of context-aware guidelines for large language model agents. *Advances in Neural Information Processing Systems*, 37:119919–119948.

Vincent Hsiao, Mark Roberts, and Leslie Smith. 2025. Procedural knowledge improves agentic llm workflows. *arXiv preprint arXiv:2511.07568*.

Michael Y Hu, Benjamin Van Durme, Jacob Andreas, and Harsh Jhamtani. 2025. Sample-efficient online learning in lm agents via hindsight trajectory rewriting. *arXiv preprint arXiv:2510.10304*.

Yixing Jiang, Kameron C Black, Gloria Geng, Danny Park, James Zou, Andrew Y Ng, and Jonathan H Chen. 2025. Medagentbench: a virtual ehr environment to benchmark medical llm agents. *Nejm Ai*, 2(9):AIdbp2500144.

Carlos E Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik Narasimhan. 2024. Swe-bench: Can language models resolve real-world github issues? In *International Conference on Learning Representations*, volume 2024, pages 54107–54157.

Ryo Kamoi, Yusen Zhang, Nan Zhang, Jiawei Han, and Rui Zhang. 2024. When can llms actually correct their own mistakes? a critical survey of self-correction of llms. *Transactions of the Association for Computational Linguistics*, 12:1417–1440.

Omar Khattab, Arnav Singhvi, Paridhi Maheshwari, Zhiyuan Zhang, Keshav Santhanam, Sri Vardhamanan, Saiful Haq, Ashutosh Sharma, Thomas T. Joshi, Hanna Moazam, Heather Miller, Matei Zaharia, and Christopher Potts. 2023. Dspy: Compiling declarative language model calls into self-improving pipelines. *arXiv preprint arXiv:2310.03714*.

James Kirkpatrick, Razvan Pascanu, Neil Rabinowitz, Joel Veness, Guillaume Desjardins, Andrei A. Rusu, Kieran Milan, John Quan, Tiago Ramalho, Agnieszka Grabska-Barwinska, Demis Hassabis, Claudia Clopath, Dharshan Kumaran, and Raia Hadsell. 2017. Overcoming catastrophic forgetting in neural networks. *Proceedings of the national academy of sciences*, 114(13):3521–3526.

Gyubok Lee, Elea Bach, Eric Yang, Tom Pollard, Alistair Johnson, Edward Choi, Yugang jia, and Jong Ha Lee. 2025. Fhir-agentbench: Benchmarking llm agents for realistic interoperable ehr question answering. *arXiv preprint arXiv:2509.19319*.

Nelson F Liu, Kevin Lin, John Hewitt, Ashwin Paranjape, Michele Bevilacqua, Fabio Petroni, and Percy Liang. 2024a. Lost in the middle: How language models use long contexts. *Transactions of the association for computational linguistics*, 12:157–173.

Xiao Liu, Hao Yu, Hanchen Zhang, Yifan Xu, Xuanyu Lei, Hanyu Lai, Yu Gu, Hangliang Ding, Kaiwen Men, Kejuan Yang, Shudan Zhang, Xiang Deng, Aohan Zeng, Zhengxiao Du, Chenhui Zhang, Sheng Shen, Tianjun Zhang, Yu Su, Huan Sun, and 3 others. 2024b. Agentbench: Evaluating llms as agents. In *International Conference on Learning Representations*, volume 2024, pages 52989–53046.

Johannes Moll, Jannik Lübberstedt, Christoph Nuernbergk, Jacob Stroh, Luisa Mertens, Anna Purcarea, Christopher Zirn, Zeineb Benchaaben, Fabian Drexel,

Hartmut Häntze, Anirudh Narayanan, Friedrich Puttkammer, Andrei Zhukov, Jacqueline Lammert, Sebastian Ziegelmayer, Markus Graf, Marion Högner, Marcus Makowski, Florian Bassermann, and 5 others. 2026. Agentic clinical reasoning over longitudinal myeloma records: a retrospective evaluation against expert consensus. *arXiv preprint arXiv:2604.24473*.

Charles Packer, Sarah Wooders, Kevin Lin, Vivian Fang, Shishir G Patil, Ion Stoica, and Joseph E Gonzalez. 2023. Memgpt: Towards llms as operating systems. *arXiv preprint arXiv:2310.08560*.

JV Roig. 2025. How do llms fail in agentic scenarios? a qualitative analysis of success and failure scenarios of various llms in agentic simulations. *arXiv preprint arXiv:2512.07497*.

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

Weixiang Shen, Bailiang Jian, Jun Li, Che Liu, Johannes Moll, Xiaobin Hu, Daniel Rueckert, Hongwei Bran Li, and Jiazhen Pan. 2026. Evo-medagent: Beyond one-shot diagnosis with agents that remember, reflect, and improve. *arXiv preprint arXiv:2604.14475*.

Freda Shi, Xinyun Chen, Kanishka Misra, Nathan Scales, David Dohan, Ed H Chi, Nathanael Schärli, and Denny Zhou. 2023. Large language models can be easily distracted by irrelevant context. In *International Conference on Machine Learning*, pages 31210–31227. PMLR.

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

Mohit Shridhar, Xingdi Yuan, Marc-Alexandre Côté, Yonatan Bisk, Adam Trischler, and Matthew Hausknecht. 2021. ALFWorld: Aligning Text and Embodied Environments for Interactive Learning. In *Proceedings of the International Conference on Learning Representations (ICLR)*.

Karan Singhal, Shekoofeh Azizi, Tao Tu, S. Sara Mahdavi, Jason Wei, Hyung Won Chung, Nathan Scales, Ajay Tanwani, Heather Cole-Lewis, Stephen Pfohl, Perry Payne, Martin Seneviratne, Paul Gamble, Chris Kelly, Nathaneal Scharli, Aakanksha Chowdhery, Philip Mansfield, Blaise Aguera y Arcas, Dale Webster, and 11 others. 2023. Large language models encode clinical knowledge. *Nature*, 620(7972):172–180.

Chenxi Wang, Zhuoyun Yu, Xin Xie, Wuguannan Yao, Runnan Fang, Shuofei Qiao, Kexin Cao, Guozhou Zheng, Xiang Qi, Peng Zhang, and Shumin Deng. 2026. Skillx: Automatically constructing skill knowledge bases for agents. *arXiv preprint arXiv:2604.04804*.

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

Jiongxiao Wang, Qiaojing Yan, Yawei Wang, Yijun Tian, Soumya Smruti Mishra, Zhichao Xu, Megha Gandhi, Panpan Xu, and Lin Lee Cheong. 2025. Reinforcement learning for self-improving agent with skill library. *arXiv preprint arXiv:2512.17102*.

Wujiang Xu, Zujie Liang, Kai Mei, Hang Gao, Juntao Tan, and Yongfeng Zhang. 2026. A-mem: Agentic memory for llm agents. *Advances in Neural Information Processing Systems*, 38:17577–17604.

Chengrun Yang, Xuezhi Wang, Yifeng Lu, Hanxiao Liu, Quoc V Le, Denny Zhou, and Xinyun Chen. 2024. Large language models as optimizers. In *International Conference on Learning Representations*, volume 2024, pages 12028–12068.

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

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

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

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. 2024. Webarena: A realistic web environment for building autonomous agents. In *International Conference on Learning Representations*, volume 2024, pages 15585–15606.

Yongchao Zhou, Andrei Ioan Muresanu, Ziwen Han, Keiran Paster, Silviu Pitis, Harris Chan, and Jimmy Ba. 2022. Large language models are human-level prompt engineers. In *The eleventh international conference on learning representations*.

Kunlun Zhu, Zijia Liu, Bingxuan Li, Muxin Tian, Yingxuan Yang, Jiaxun Zhang, Pengrui Han, Qipeng Xie, Fuyang Cui, Weijia Zhang, Xiaoteng Ma, Xiaodong Yu, Gowtham Ramesh, Jialian Wu, Zicheng Liu, Pan Lu, James Zou, and Jiaxuan You. 2025. Where llm agents fail and how they can learn from failures. *arXiv preprint arXiv:2509.25370*.

A Related Work

Verbal-feedback self-improvement. Reflexion (Shinn et al., 2023) introduced the paradigm in which an agent writes natural-language critiques of its own failed trajectories and conditions on those critiques on subsequent attempts. ExpeL (Zhao et al., 2024) extends this to cross-trajectory rule extraction with bounded AGREE, EDIT, REMOVE, or ADD operations. A critical survey (Kamoi et al., 2024) documents that LLMs often cannot reliably correct their own reasoning without external feedback. These approaches accumulate or revise guidance monotonically with no held-out check that a new piece of feedback preserves previously correct behavior. GRASP retains the verbal-feedback paradigm but gates every candidate edit on a regression-aware probe of previously failing and passing examples.

Learning from agent failures. AutoGuide (Fu et al., 2024) creates context-aware guidelines from paired trajectories with different outcomes and retrieves them at inference. AgentDebug (Zhu et al., 2025) introduces a modular taxonomy of agent failures and a debugging framework that traces failures to their root cause. ECHO (Hu et al., 2025) adapts hindsight experience replay to LM agents, generating counterfactual successful trajectories from failed ones. Experiential Reflective Learning (Allard et al., 2026) extracts heuristics from single-attempt trajectories and retrieves them at test time. GRASP shares the failure-driven orientation but differs on three axes. The unit of learning is a structured behavioral skill rather than a retrieved trajectory or heuristic, every edit passes through an explicit regression-aware gate rather than relying on accumulation or retrieval, and modification and removal are first-class operations, so the library evolves rather than grows.

Skill-library agents. Voyager (Wang et al., 2023) established skill libraries as accumulated behavioral primitives conditioned on at inference. SkillX (Wang et al., 2026) extends this to general agent tasks by decomposing successful trajectories into named skills retrieved by token overlap, and SAGE (Wang et al., 2025) integrates a skill library into RL-based agent training. These methods grow their library without validating that each addition preserves prior behavior, and draw skills only from successful trajectories. GRASP retains the structured-library design but is failure-driven, supports modification and removal, gates every edit on probe-based net improvement, and requires no parameter updates.

Memory-augmented LLM agents. MemGPT (Packer et al., 2023) introduced managed long-term memory with explicit page-in and page-out operations, and A-Mem (Xu et al., 2026) organizes agent memories into dynamic interconnected knowledge networks with agent-driven indexing decisions. In our experimental comparison, sequential and batch memory (Chen et al., 2025) append correction notes after each failing episode or batch, and Evo-MedAgent (Shen et al., 2026) maintains episodic and semantic memories with top-k retrieval. None of these methods evaluate updates against a held-out probe and the memory grows monotonically over training. In our experiments this leads memory baselines to either regress below the no-skills baseline as context dilutes (Sequential, Batch) or oscillate sharply between epochs (Evo-MedAgent), confirming that update frequency and retrieval are not substitutes for an explicit acceptance criterion.

Prompt and program optimization. DSPy (Khattab et al., 2023) treats the natural-language portion of an LLM pipeline as parameters optimized against a development set, using held-out validation to compare candidates. Broader prompt-optimization methods (Zhou et al., 2022; Yang et al., 2024) cast prompt design as search guided by held-out performance. GRASP shares the held-out validation idea but applies it at a different granularity. The optimized object is a structured library of behavioral skills with add, modify, and remove operations rather than a flat prompt or fixed program template, and the acceptance criterion explicitly bounds regressions on previously correct behavior. This connects GRASP to the broader continual-learning literature on catastrophic forgetting (Kirkpatrick et al., 2017).

Clinical agent benchmarks. LLM evaluation in clinical settings has historically focused on question answering. Med-PaLM (Singhal et al., 2023) established MultiMedQA as a benchmark combining medical licensing exam questions, biomedical literature, and consumer health queries. These benchmarks test medical knowledge but not the procedural reliability required to operate in a clinical environment. MedAgentBench (Jiang et al., 2025) introduced FHIR-based tasks requiring resource retrieval, medication reconciliation, and verified

writes. MedAgentBench-v2 (Chen et al., 2025) extends the original with multi-step decision tasks, coordinated writes, and clinical safety protocols. FHIR-AgentBench (Lee et al., 2025) evaluates structured clinical question answering and tool use on an independent FHIR environment.

Structured-environment agents beyond clinical tasks. ReAct (Yao et al., 2022b) introduced the thought-action-observation loop that underlies most modern tool-using agents, and Toolformer (Schick et al., 2023) demonstrated that LLMs can learn to invoke external APIs without explicit supervision. AgentBench (Liu et al., 2024b) provides a standard suite for operating systems, databases, and web interfaces, while Mind2Web (Deng et al., 2023) and WebArena (Zhou et al., 2024) target the web and SWE-bench (Jimenez et al., 2024) targets software engineering. The non-medical experiments in Section 4.6 draw from the AgentBench environments.

Context degradation under long inputs. A consequence of monotonic memory accumulation is that task-relevant guidance is diluted by outdated or redundant entries. Shi et al. (2023) showed that LLMs are sensitive to irrelevant context inserted into their inputs, and Liu et al. (2024a) documented systematic position-based degradation in long-context retrieval. These findings motivate the bounded-library design of GRASP, which keeps the active skill set small by allowing removal and modification rather than only addition.

B Method Details

B.1 Per-Batch Algorithm

Algorithm 1 gives the full per-batch GRASP update referenced in Section 2.3.

B.2 Contrastive Revision

When the highest-scoring candidate $c^\star$ satisfies Equation 1 but causes at least one regression on the probe, the skill-writer is invoked a second time with the original proposal and the regressing traces, and asked to produce a narrower version that preserves the original fixes while exempting the regression patterns. The revision is evaluated against the same probe and the same $F_0, R_0, \mathcal{E}_0$ as the original candidate. It replaces $c^\star$ only if its adjusted score $(F - F_0) - (R - R_0)$ is strictly higher and it satisfies $R \leq R_0$. If the revision fails either condition, the original candidate is applied unchanged.

B.3 Invalid-Action Regression Weighting

Regressions from invalid agent actions (malformed tool calls, parse failures, or other agent-side errors the environment rejects) are weighted by $\lambda = 2$ in the score, while the hard budget $R(c) \leq R_0$ uses the unweighted count. The asymmetry reflects cost, not tuning. An invalid action fails the episode at the interface level, and under transfer a single rejected call ends the trajectory, so trading a wrong answer for a malformed call is a worse regression than the raw count indicates. The score penalizes these edits more heavily to discourage them without forbidding them, while the budget stays an unweighted count to keep the constraint interpretable. The result is insensitive to λ (Appendix E.2), since invalid-action regressions are rare on the probe.

B.4 Skill Template

GRASP-authored skills follow the structure below, mirroring the Agent Skills SKILL.md convention. Selected examples appear in Appendix F.

```
---
name: skill_name
description: one-line description
tags: [tag1, tag2]
version: 1
provenance:
  epoch: 0
  update_cycle: 2
  action: ADD
  probe_score: 4
---

## Trigger
When [specific condition]...

## Rule
You must [behavioral instruction]...

## Verification
After [action with side effect], confirm...

## Example
Failing: [failing trajectory]
Corrected: [corrected trajectory]
```

B.5 Skill and Memory Injection

Methods differ in how learned content reaches the agent at inference, in where it is placed, how much is injected, and how often it is refreshed. We hold the placement fixed across methods so

Algorithm 1: GRASP: per-batch skill update

Input: Library S, batch B of dev episodes, prior dev runs $\mathcal{H}$, max proposals K, probe size N
Output: Updated library S

// Step 1: Run batch and record outcomes
$\mathcal{T}_B \leftarrow$ run agent with library S on each episode in B ;
record per-episode correctness in $\mathcal{H}$;
$\mathcal{F}_B \leftarrow$ failed traces in $\mathcal{T}_B$;

// Step 2: Group failures and propose
$L \leftarrow$ CLASSIFYBYFAILUREMODE($\mathcal{F}_B$) ; // LLM call
$\{\mathcal{C}_k\}_{k=1}^{K} \leftarrow$ PROPOSE($\mathcal{F}_B$, L, S) ; // K candidate edits

// Step 3: Construct probe from earlier dev runs (never touches val/test)
$\mathcal{P}_{\text{fail}}, \mathcal{P}_{\text{pass}} \leftarrow$ STRATIFIEDSAMPLE($\mathcal{H} \setminus B$, $N/2$, $N/2$) ;

// Step 4: Baseline run — re-evaluate current library S on the probe
$F_0 \leftarrow |\{x \in \mathcal{P}_{\text{fail}} : x \text{ passes under } S\}|$;
$R_0 \leftarrow |\{x \in \mathcal{P}_{\text{pass}} : x \text{ fails under } S\}|$;
$\mathcal{E}_0 \leftarrow$ samples that errored during baseline run ;

// Step 5: Score each candidate on the same probe
for $c \in \{\mathcal{C}_1, \ldots, \mathcal{C}_K\}$ **do**
  $S^c \leftarrow$ apply c to a fork of S ;
  $F(c) \leftarrow |\{x \in \mathcal{P}_{\text{fail}} \setminus \mathcal{E}_0 : x \text{ passes under } S^c\}|$;
  $R(c) \leftarrow |\{x \in \mathcal{P}_{\text{pass}} \setminus \mathcal{E}_0 : x \text{ fails under } S^c\}|$;
  $\text{score}(c) \leftarrow (F(c) - F_0) - (R(c) - R_0)$;

// Step 6: Accept the best candidate that satisfies the regression budget
$\mathcal{C}^\star \leftarrow \{c : \text{score}(c) > 0 \text{ and } R(c) \leq R_0\}$;
if $\mathcal{C}^\star \neq \emptyset$ **then**
  $c^\star \leftarrow \arg\max_{c \in \mathcal{C}^\star} \text{score}(c)$;
  if $R(c^\star) > 0$ **then**
    $c^\star \leftarrow$ CONTRASTIVEREVISION($c^\star$, regressing samples) ; // if revision improves score
  $S \leftarrow$ apply $c^\star$ to S ;
return S

that it cannot account for the accuracy differences, and report the remaining differences in injected volume in Appendix Table 6. On the first decision turn, all methods insert their learned content into the same structured field of the task prompt, the `behavioral_skills` field that GRASP uses, falling back to a prepend before the task description only when the prompt is not structured as JSON. GRASP injects its skill library, Sequential and Batch memory inject their accumulated correction-note block, Evo-MedAgent injects its retrieved episodic and semantic memory, ExpeL injects its numbered rule list, and SkillX injects its top-k retrieved skills. Each method keeps its own internal formatting of that content, but the injection point is common to all. On turns after the first, the memory and retrieval methods refresh their content per query in their native manner, appending the current block to the running context, while GRASP's library is fixed for the episode and is not re-injected.

Holding placement fixed isolates two things that still vary across methods, the learned content itself and the update or selection rule that produces it, which is what the comparison is meant to test. The differences that remain do not favor GRASP. End-of-training injected-token counts in Appendix Table 6 show that GRASP injects roughly an order of magnitude fewer tokens than the unbounded memory baselines and than SkillX, so the gain can-

Table 6: LLM call accounting per training batch and library size at end of training, on MedAgentBench with gpt-oss-120b. *Update* calls come from the skill-writer, *probe* calls come from re-running the agent on development-split probe episodes (N=36 episodes $\times$ K+1=5 library variants).

Method	Training (per batch)			Inference state		Inference context
	Update calls	Probe calls	Total calls	Items	Tokens	
No skills	0	0	117	0	0	none
Sequential mem.	28	0	145	332	34,500	unbounded
Batch mem.	28	0	145	331	36,500	unbounded
ExpeL	14	0	131	20	1,200	bounded (cap 20 rules)
Evo-MedAgent	96	0	213	209	53,100	retrieval over growing stores
SkillX	90	0	207	5	51,600	retrieval (k=5, code skills)
GRASP (ours)	6	440	560	5	5,600	capacity-bounded ($\leq$10)

Table 7: Base models used in this work. Open-source models are self-hosted, proprietary models are accessed via the listed provider's API. Decoding settings are identical across models within each role (agent, skill-writer), see Section 3.4.

Model	Provider	Access	Version / handle	Size	Precision	Access date
gpt-oss-120b	OpenAI	self-hosted	openai/gpt-oss-120b	120B	MXFP4	05/2026
DeepSeek V4 Flash	DeepSeek	self-hosted	deepseek-ai/DeepSeek-V4-Flash	284B (13B active)	FP8	05/2026
Gemini 3.1 Flash Lite	Google	API	gemini-3.1-flash-lite-preview	–	–	05/2026
GPT-5.4 (low effort)	OpenAI	API	gpt-5.4-2026-03-05	–	–	05/2026
GPT-4.1	OpenAI	API	gpt-4.1-2025-04-14	–	–	05/2026

not come from injecting more text. Batch memory shares GRASP's injection field and its per-batch update cadence yet regresses below the no-skills baseline (34.4 versus 40.6 on gpt-oss-120b), indicating that injecting content into this field without an acceptance check is not by itself helpful. The no-gate ablation in Section 4.5 removes only the acceptance gate while leaving GRASP's injection and proposal machinery unchanged, and test accuracy drops by 25.3 points, attributing the gain to the gate rather than to placement.

C Model Specifications

Table 7 lists the exact model identifiers, providers, and access details for the five base models used throughout this work. Open-source models (gpt-oss-120b, DeepSeek V4 Flash) are self-hosted using vLLM on an NVIDIA H200 GPU. Proprietary models are accessed via official provider APIs. All agent execution uses temperature 0.0 with top-$p = 1.0$. The skill-writer uses temperature 0.7 with top-$p = 1.0$ for proposal diversity (see Section 3.2). Output is capped at 32,768 tokens per call across all methods and models, input prompts use each model's default context window without manual truncation.

Table 8: Benchmarks and split sizes used in this work. MedAgentBench and MedAgentBench-v2 include an out-of-distribution (OOD) test split formed from held-out task types; the remaining benchmarks do not.

Benchmark	Dev	Val	Test	OOD
Clinical (FHIR)				
MedAgentBench	96	80	64	60
MedAgentBench-v2	96	80	64	60
FHIR-AgentBench	120	80	80	–
AgentBench				
DBBench	240	124	60	–
OS Interaction	79	56	35	–
ALFWorld	26	24	20	–
WebShop	100	50	50	–

D Benchmark and Split Details

Table 8 reports split sizes for all seven benchmarks. The three clinical benchmarks share a common training and evaluation protocol. The four AgentBench environments are used as exploratory non-clinical probes.

MedAgentBench. Ten clinical task types (300 samples, 30 per type) executed against a live FHIR server hosted in Docker (Jiang et al., 2025). Tasks include resource retrieval, medication reconciliation, and verified writes. Two task types (tasks 6 and 7) are held out as the OOD split. The remaining eight task types are partitioned per task

type into dev (12), val (10), and in-domain test (8) with a fixed seed. Scoring is exact-match against ground-truth answers and FHIR-server state after the agent completes its trajectory.

MedAgentBench-v2. A redesigned variant with ten new task types covering imaging follow-up, anticoagulation reconciliation, vital-sign aggregation, catheter-removal protocols, oncology referrals, hypothyroidism management, QTc-prolongation protocols, opioid and naloxone pairing, and vaccination refresh (Chen et al., 2025). The task types incorporate multi-step decision logic and clinical safety protocols absent from v1. OOD tasks (5 and 7) differ from v1, so the OOD split exercises an independent set of held-out clinical workflows. Split sizes, scoring, and FHIR backend match v1.

FHIR-AgentBench. A read-only structured clinical question-answering benchmark on MIMIC-IV FHIR data hosted on the Google Cloud Healthcare API (Lee et al., 2025). Tasks require retrieval and reasoning over FHIR resources but do not write. Evaluation differs from the MedAgentBench variants. Answer correctness is assessed by an LLM judge against a ground-truth answer with prompt-level normalisation, and retrieval is scored by precision and recall against ground-truth resource IDs. The single headline number reported in this paper is LLM-judged answer accuracy. The benchmark is partitioned into dev (120), val (80), and test (80) with no OOD split.

AgentBench environments. Four environments from the AgentBench suite (Liu et al., 2024b) are evaluated as exploratory non-clinical probes. These are Database (DBBench), Operating System (OS Interaction), ALFWorld (Shridhar et al., 2021), and WebShop (Yao et al., 2022a). Split sizes are listed in Table 8. DBBench combines real queries from the standard split (partitioned 60/40 by query type) with synthetic aggregation queries. OS Interaction uses worlds 1 to 5 and 7 split 60/40 stratified per world, with world 6 held out. ALFWorld partitions its standard split 60/40 stratified by task type. WebShop uses a round-robin partition of the standard `webshop-std` index range (instances 0–199) into dev (100), validation (50), and test (50). Scoring uses each environment's native success criterion (task completion, query correctness, or attribute-matching reward).

E Extended Results

E.1 Failure-Mode Classifier: Within-Run Vocabulary Dynamics

The failure-mode classifier (Section 2.2) is open-vocabulary. At each batch it assigns each failing trace a label, preferring prior labels and minting new ones only when no prior label fits. We characterize two within-run properties of the resulting vocabulary on MedAgentBench and MedAgentBench-v2 with `gpt-oss-120b`, across the five seeds used in the main results.

Table 9: Cumulative label vocabulary size by batch index. Mean ± std across 5 seeds.

Batch idx	MedAgentBench	MedAgentBench-v2
0	11.6 ±1.4	13.0 ±4.9
2	16.4 ±1.4	21.2 ±6.1
4	18.8 ±2.6	24.8 ±7.8
6	19.6 ±2.9	27.8 ±7.8
8	21.0 ±4.1	29.4 ±8.0
9	22.2 ±4.5	30.6 ±8.2

Table 9 reports the mean cumulative number of distinct labels seen by batch index. The first batch discovers 12 to 13 labels, and the marginal addition per batch drops to 1 or 2 by mid-training. The vocabulary converges rather than expanding indefinitely.

Table 10: Per-batch trace-level label reuse rate (%). Fraction of failing traces in batch b whose label appeared in some batch $b' < b$ in the same seed. Mean ± std across 5 seeds.

Batch idx	MedAgentBench	MedAgentBench-v2
1	63.9 ±27.1	46.3 ±20.2
3	78.8 ±18.3	68.2 ±17.3
5	80.3 ±21.1	84.3 ±18.9
7	97.5 ±5.0	92.6 ±4.6
9	80.3 ±20.9	88.0 ±11.0

Table 10 reports the fraction of failing traces per batch whose label was already present in this seed's vocabulary. Reuse climbs from 0% at batch 0 (by construction) to 80–90% by batch 4 or 5 and above 90% by batch 7. After the first epoch, the classifier overwhelmingly reuses prior labels rather than minting new ones.

Together, these two properties show that the classifier converges on a stable label set within a single training run. This is the property the skill-writer needs. By mid-training, failures within a batch are grouped under labels the skill-writer has already

seen, allowing edits to target recurring mechanisms rather than novel ones.

E.2 Hyperparameter Sensitivity

Table 11: One-at-a-time sensitivity sweep on MedAgentBench (gpt-oss-120b) around the default configuration B=48, N=36, K=4, λ=2 (marked †). The default row is the same run reused in every block. Three seeds.

	Setting	Val*	Test
B (batch size)	24	87.1 $\pm$7.3	89.1 $\pm$7.2
	48†	86.0 $\pm$4.4	88.8 $\pm$5.8
	96	83.8 $\pm$0.0	87.0 $\pm$2.4
N (probe size)	16	84.6 $\pm$5.1	82.3 $\pm$3.3
	36†	86.0 $\pm$4.4	88.8 $\pm$5.8
	72	86.2 $\pm$8.7	86.5 $\pm$10.6
K (candidates)	1	70.8 $\pm$25.0	73.4 $\pm$23.1
	4†	86.0 $\pm$4.4	88.8 $\pm$5.8
	8	80.0 $\pm$7.6	84.4 $\pm$8.3
λ (invalid-action weight)	1	81.7 $\pm$8.9	83.9 $\pm$10.0
	2†	86.0 $\pm$4.4	88.8 $\pm$5.8
	4	87.1 $\pm$6.4	87.0 $\pm$7.4

Table 11 reports a one-at-a-time sensitivity sweep over the parameters governing the per-batch update loop. These are the batch size B (dev episodes per update), the probe size N (episodes used to evaluate candidate libraries), the number of candidate edits K per batch, and the invalid-action regression weight λ (Appendix B.3). Each block varies one parameter around the defaults B=48, N=36, K=4, λ=2 used throughout the paper, with the default run reused as the reference point in every block.

GRASP is robust to B, N, and λ but sensitive to K. Varying B across $\{24, 48, 96\}$ moves test accuracy by at most 2.1 points (87.0 to 89.1), all within one standard deviation of each other. Probe size N has a floor at 16 (82.3 test) and reaches its full value by 36 (88.8), with no further gain at 72, which the gate-selectivity analysis (Appendix E.3) attributes to the gate already resolving its decisions at N=36 rather than to a probe too small to discriminate. Varying λ across $\{1, 2, 4\}$ leaves in-domain test accuracy in an 84 to 89 band, with all three settings overlapping in standard deviation. K is the exception. At K=1 test accuracy falls to 73.4 ($\pm$23.1), since a single candidate leaves the gate with a binary accept-or-reject decision and removes the comparative selection that sets GRASP apart from naive verbal-feedback methods, and K=8 trails K=4 (84.4 vs 88.8) as more candidates dilute the fixed probe budget. We set K=4 to give the gate real comparative work without thinning per-candidate evaluation.

E.3 Gate Selectivity

Across five seeds on MedAgentBench with gpt-oss-120b, GRASP applies an edit in 64% of batches and rejects all four candidates in the other 36%. Across all candidates it admits 16%, and on average 1.3 of the 4 per batch clear the regression budget. The accepted candidate's score is a median of 3 and a mean of 4 fixed examples above the acceptance threshold, with a tail reaching 15. These margins are several examples wide on an 18-positive probe, so the gate resolves its decisions well above the one-example floor, which is why a larger probe does not improve accuracy (Table 11).

E.4 Statistical Significance

Table 12 reports per-comparison statistics for the primary results. For each comparison we report the mean difference Δ, a percentile bootstrap 95% confidence interval on Δ from 10,000 resamples, a two-sided unpaired permutation test (exact when $\binom{n_a+n_b}{n_a} \leq 50{,}000$, else Monte Carlo with 10^5 permutations), and Cohen's d with pooled standard deviation. We treat Δ and its bootstrap CI as the primary statistical evidence and report permutation p-values as supporting detail. Three-seed rows are reported as effect-size evidence rather than as significance claims, and we do not apply multiple-comparisons correction.

The seed counts in the main tables, five for open-source models and three for proprietary models and all transfer experiments, imply a minimum achievable two-sided permutation p of 0.008 at $n_a = n_b = 5$ and 0.10 at $n_a = n_b = 3$. The latter floor limits the power of permutation testing on three-seed rows but not the interpretation of Δ and its CI computed on the same data.

The primary comparisons show large and reliably positive differences in every cell. On the five-seed open-source comparisons, gpt-oss-120b reaches $\Delta = +21.0$ with 95% CI $[10.3, 33.8]$ and DeepSeek V4 Flash $\Delta = +14.1$ with CI $[9.4, 18.8]$, both at $p = 0.008$. The three-seed comparisons are of similar or larger magnitude with CIs that exclude zero. Gemini 3.1 Flash Lite reaches $\Delta = +16.2$ with CI $[14.1, 18.2]$, GPT-4.1 $\Delta = +37.5$ with CI $[27.1, 45.8]$, and GPT-5.4 (low) $\Delta = +38.0$ with CI $[26.6, 47.4]$, each with the permutation p capped at 0.10 by the three-seed protocol. The corresponding Cohen's d val-

Table 12: Per-comparison statistics. Δ is the mean difference (method A minus method B). 95% CI is a 10,000-resample percentile bootstrap on Δ. p is a two-sided unpaired permutation test. d is Cohen's d with pooled SD. Stars mark $p < 0.05$. Daggers (†) mark rows where the sample size makes $p < 0.05$ unreachable by construction, the observed p for those rows is the minimum achievable two-sided p at the given n.

Benchmark	Split	Model	Comparison	A	B	Δ	95% CI	p	d
MAB	Test	gpt-oss-120b	GRASP vs. Evo-MedAgent	88.8	67.8	$+21.0^{*}$	[10.3, 33.8]	0.008	1.94
MAB	Test	DeepSeek V4 Flash	GRASP vs. SkillX	70.0	55.9	$+14.1^{*}$	[9.4, 18.8]	0.008	3.24
MAB	Test	Gemini 3.1 Flash Lite	GRASP vs. Evo-MedAgent	71.4	55.2	+16.2	[14.1, 18.2]	$0.100^{\dagger}$	8.95
MAB	Test	GPT-4.1	GRASP vs. Batch Memory	84.9	47.4	+37.5	[27.1, 45.8]	$0.100^{\dagger}$	5.53
MAB	Test	GPT-5.4 (low)	GRASP vs. Seq. Memory	85.4	47.4	+38.0	[26.6, 47.4]	$0.100^{\dagger}$	4.65
MAB	OOD	gpt-oss-120b	GRASP vs. Evo-MedAgent	56.3	31.3	+25.0	[3.7, 44.0]	0.087	1.36
MAB-v2	Test	gpt-oss-120b	GRASP vs. ExpeL	76.9	67.8	$+9.1^{*}$	[5.3, 13.1]	0.008	2.59
MAB-v2	OOD	gpt-oss-120b	GRASP vs. SkillX	86.0	87.1	−1.1	[−2.6, 0.0]	0.556	−0.91
FHIR-AB	Test	gpt-oss-120b	GRASP vs. no skills	58.2	51.5	$+6.8^{*}$	[+3.7, 10.0]	0.008	2.36

ues appear in Table 12, where the Gemini-cell d of 8.95 is inflated by an unusually low pooled standard deviation in that row and should not be read as an estimate of typical effect magnitude. The MedAgentBench-v2 in-domain comparison on gpt-oss-120b is also strong, $\Delta = +9.1$ with CI $[5.3, 13.1]$ at $p = 0.008$.

The MedAgentBench OOD comparison on gpt-oss-120b shows $\Delta = +25.0$ with a bootstrap 95% CI of $[3.7, 44.0]$ that excludes zero. Evo-MedAgent's seed-to-seed standard deviation of 21.7 points on this split widens the permutation null to $p = 0.087$ but does not shift the underlying gap.

On MedAgentBench-v2 OOD, GRASP scores 1.1 points below SkillX with a bootstrap CI that includes zero, and we make no claim of superiority on this split. The cross-model transfer experiments use three seeds per cell, so we do not assign p-values to individual cells and interpret those effects through Δ and the standard deviations reported in the main tables. Cross-benchmark transfer and the FHIR-AgentBench in-domain comparison use five seeds per cell, with bootstrap CIs reported alongside the point estimates.

E.5 Cross-Writer Transfer Across Methods

Table 13 tests whether the cross-model asymmetry in Section 4.3 reflects GRASP's gate or any structured guidance from a stronger writer, by running the cross-writer protocol on all five baselines. For each method, GPT-5.4 (low) acts as both agent and writer during training, the resulting library is frozen, and a weaker executor (Gemini 3.1 Flash Lite or gpt-oss-120b) runs at test time.

The asymmetry does not replicate on any baseline. GRASP is the only method whose GPT-5.4 library generalizes OOD when transferred, reaching 71.1 on Gemini and 77.8 on gpt-oss, while no baseline's transferred library exceeds 35.0 (SkillX→Gemini) and most stay below 20 on both executors. The same frozen GPT-5.4 li-

Table 13: Cross-writer transfer on MedAgentBench across three seeds. GPT-5.4 → Gemini applies a GPT-5.4-trained library to Gemini 3.1 Flash Lite at test time. GPT-5.4 → gpt-oss applies the same library to gpt-oss-120b.

Method	Setting	ID Test	OOD
GRASP	GPT-5.4 self-train	85.4 ±10.4	80.6 ±6.7
	Gemini self-train	71.4 ±1.8	41.7 ±20.9
	GPT-5.4 → Gemini	**76.6 ±10.9**	**71.1 ±1.9**
	gpt-oss self-train	88.8 ±5.8	56.3 ±14.5
	GPT-5.4 → gpt-oss	**76.0 ±7.7**	**77.8 ±1.0**
SkillX	GPT-5.4 self-train	44.8 ±0.9	3.3 ±2.9
	Gemini self-train	54.2 ±3.6	38.3 ±1.7
	GPT-5.4 → Gemini	56.8 ±0.9	35.0 ±1.7
	gpt-oss self-train	53.1 ±4.6	21.3 ±3.6
	GPT-5.4 → gpt-oss	46.4 ±8.6	17.2 ±3.8
Evo-MedAgent	GPT-5.4 self-train	43.8 ±4.1	1.7 ±1.7
	Gemini self-train	55.2 ±1.8	15.6 ±2.5
	GPT-5.4 → Gemini	54.2 ±1.8	18.3 ±6.0
	gpt-oss self-train	67.8 ±14.1	31.3 ±21.7
	GPT-5.4 → gpt-oss	44.3 ±3.9	8.3 ±3.3
Seq. Memory	GPT-5.4 self-train	47.4 ±5.0	15.6 ±25.5
	Gemini self-train	53.9 ±3.3	2.5 ±1.2
	GPT-5.4 → Gemini	54.2 ±3.3	11.1 ±7.5
	gpt-oss self-train	41.2 ±5.5	16.7 ±15.6
	GPT-5.4 → gpt-oss	44.8 ±11.9	13.3 ±10.4
Batch Memory	GPT-5.4 self-train	46.4 ±0.9	16.7 ±3.3
	Gemini self-train	32.0 ±25.4	2.5 ±1.2
	GPT-5.4 → Gemini	49.5 ±5.9	12.2 ±7.9
	gpt-oss self-train	34.4 ±8.0	13.0 ±8.1
	GPT-5.4 → gpt-oss	37.0 ±11.7	14.4 ±8.6
ExpeL	GPT-5.4 self-train	43.8 ±1.6	11.7 ±3.3
	Gemini self-train	53.6 ±1.8	17.2 ±2.5
	GPT-5.4 → Gemini	52.6 ±0.9	18.9 ±4.2
	gpt-oss self-train	49.1 ±6.3	12.0 ±5.2
	GPT-5.4 → gpt-oss	39.6 ±3.9	12.2 ±3.5

brary reaches comparable OOD on two different executors, and in both cases the transfer exceeds the target's own self-trained library (41.7 for Gemini, 56.3 for gpt-oss), so the gains track the source library rather than the executor. Writer strength alone does not explain this, since the baselines' GPT-5.4 libraries are weak even at the source (1.7 to 16.7 OOD under self-training), leaving little procedural knowledge for transfer to carry. gpt-oss is a capable executor, yet baseline transfers to it stay near or below its self-train OOD rather than benefiting from the GPT-5.4 source. What survives transfer to a weaker executor is the procedural knowledge the gate selects for, not writer strength alone.

E.6 Action-Budget Failures on MedAgentBench-v2

Table 14: Action-budget exhaustion on MedAgentBench-v2 for GPT-5.4 (low), as averaged limit-hit episodes per run out of 64 samples. *Base* is the bare model under each method's evaluation conditions, *Adapted* is the method's best-validation checkpoint.

Method	Base	Adapted	Δ
Evo-MedAgent	5.7	5.0	−0.7
Seq. Memory	7.0	7.0	0.0
ExpeL	7.0	7.7	+0.7
Batch Memory	7.0	9.0	+2.0
SkillX	6.0	9.3	+3.3
GRASP	6.7	8.7	+2.0

On MedAgentBench-v2, the proprietary models GPT-5.4 (low) and GPT-4.1 most often fail by exhausting the fixed 8-action budget rather than by producing an incorrect answer. The effect concentrates on two task types involving paginated FHIR search (tasks 8 and 10), where the agent spends actions paging through search results and reaches the budget before completing the task. These two tasks reach the action limit on roughly 37–40% of attempts for GPT-5.4 (low), independent of the adaptation method.

Table 14 reports the average number of limit-hit episodes per run on MedAgentBench-v2 for GPT-5.4 (low), out of 64 evaluation samples, comparing the bare base model under each method's evaluation conditions against the method's best-validation checkpoint. The base-model column varies only by stochastic decoding, and its spread (5.7–7.0) is within Poisson noise for events at this rate. After adaptation, three methods stay at the base rate within noise and three sit modestly above it, but this split does not correspond to whether a method injects skills. Batch Memory, which injects only a flat memory block, shows the same increase as the skill-based methods, while Sequential Memory, which injects the same kind of memory, does not. With two to three runs per method these differences of one to two additional limit hits are directional rather than statistically resolved. None of the methods reduces the underlying failure, which is set by the base model's behavior on paginated search rather than by any adaptation.

F Selected Skills Learned by GRASP

The four skills below were selected by the authors, not sampled at random, to illustrate the format of GRASP-authored skills across a range of conditions. They span four benchmarks (ALFWorld, DBBench, MedAgentBench, MedAgentBench-v2), two authoring models (`gpt-oss-120b` and `GPT-5.4` low effort), and two edit types (ADD and MODIFY). The selection is intended to show what GRASP can produce, not what it typically produces. We do not claim these four are representative of average skill quality, length, or specificity across runs. We include these particular four because they make the structural elements of a skill (trigger, rule, verification, contrastive example, provenance) inspectable across the procedural domains the paper evaluates.

The clinical skills (F.3 and F.4) encode procedures for operating MedAgentBench and MedAgentBench-v2 as software systems. They contain medical-domain vocabulary (drug names, RxNorm-style identifiers, FHIR resource types) because the benchmarks do, but the operational content is about the benchmark's API, identifier scheme, and scoring criteria, not about real-world clinical care. The RxNorm codes shown in F.3, for example, are the codes used by MedAgentBench-v2's FHIR fixture, the skill captures the benchmark's coding scheme rather than the production RxNorm vocabulary. Reading these skills as drafts of clinical decision rules would misidentify what GRASP has learned. They are policies for completing benchmark episodes under exact-match scoring. The clinical scope of these experiments is discussed further in the Limitations section.

Enforce Cleaning Before Placement

(non-clinical, recurring workflow constraint; `gpt-oss-120b`, AgentBench / ALFWorld; ADD)

```
---
name: enforce_clean_before_placement
description: Clean target object before placing into final receptacle
tags: [alfworld, workflow-order]
version: 1
provenance:
  epoch: 3
  action: ADD
---

## Pattern Description

The target object must never be placed into its final receptacle before
it has been cleaned. The constraint applies to any task whose
instruction mentions "clean" (e.g. "pick_clean_then_place", "clean the
X before placing") or whose query type is pick_clean_then_place.
Enforcing a cleaning action immediately after pickup and before any
placement guarantees that the environment state satisfies the
cleanliness predicate; otherwise the episode silently fails.

## Triggers

- Instruction contains the word "clean" (case-insensitive) or the
  phrase "clean the <object>".
- Query type is pick_clean_then_place.
- A `take <obj> from <loc>` action has just executed and no prior
  `clean` observation exists for the carried object.

## Example

### Wrong trajectory

Think: The task is "pick_clean_then_place-Spatula-None-Drawer-10".
Act:   take spatula 1 from drawer 2
Obs:   You are now carrying spatula 1.
Think: I have the spatula, place it.
Act:   put spatula 1 in/on drawer 1
Obs:   Spatula 1 is now in drawer 1.
       [Episode ends; goal NOT satisfied.]
```

```
### Correct trajectory

Think: Instruction mentions "clean" -- enforce cleaning before
       placement.
Act:   take spatula 1 from drawer 2
Obs:   You are now carrying spatula 1.
Think: Before placing, I must clean the spatula.
Act:   clean spatula 1 with sinkbasin 1
Obs:   Spatula 1 is now clean.
Act:   put spatula 1 in/on drawer 1
Obs:   Spatula 1 is now in drawer 1.
       [Goal satisfied.]

## Pitfalls

- Do not clean after placement.
- If the object is already clean, skip the redundant action.
- Choose a valid cleaning device (sinkbasin, dishwasher); other
  receptacles raise an invalid-action error.
- Do not trigger on tasks that do not mention cleaning; this wastes
  the step budget.
```

Parse Percentage Strings for Aggregation
(non-clinical, low-gain benchmark; gpt-oss-120b, AgentBench / DBBench; ADD)

```
---
name: parse_percentage_strings_for_aggregation
description: Strip percent signs and normalise commas before numeric ops
tags: [dbbench, sql, locale-aware]
version: 1
provenance:
  epoch: 2
  action: ADD
---

## Pattern Description

When a task asks for a numeric aggregation (SUM, AVG, MIN, MAX) and
the table stores percentages as text with a comma decimal separator
and a trailing % sign (e.g. "23,3%"), the raw string cannot be
compared or summed. The agent must strip the % character and replace
the comma with a dot before casting to a numeric type. The same rule
applies whenever a WHERE clause filters on such a column.

## Triggers

- Instruction contains a percentage written with a comma and trailing
  % (e.g. "23,3%").
- Schema includes a column whose name contains % (e.g.
  `Foreign nationals in %`).
- Query type is an aggregation and the task mentions a percentage
  filter.

## Recommended Patterns

### Filter on a percentage column

-- WRONG
SELECT SUM(pop) FROM tbl
WHERE `Foreign nationals in %` = '23,3%';

-- CORRECT
SELECT SUM(pop) AS total
FROM   tbl
WHERE  CAST(REPLACE(REPLACE(`Foreign nationals in %`,
                            '%', ''), ',', '.')
            AS DECIMAL(10,2)) = 23.3;
```

```
### Aggregate a numeric column that contains commas

-- WRONG (keeps commas)
SELECT SUM(`Population`) FROM tbl;

-- CORRECT
SELECT SUM(CAST(REPLACE(`Population`, ',', '') AS UNSIGNED))
       AS total_population
FROM   tbl;

## Failure Indicators

- Observation shows 0 rows or NULL when a non-zero result is expected.
- SQL error: "Incorrect decimal value: '23,3%'".
- Final answer is a string that still includes the % sign.
```

Opioid–Naloxone Verification

(benchmark-procedural with medical vocabulary; gpt-oss-120b, MedAgentBench-v2; MODIFY)

```
---
name: opioid_naloxone_verification
description: Ensure active opioid orders have matching naloxone prescription
tags: [medagentbench-v2, medication-safety, rxnorm]
version: 2
provenance:
  epoch: 2
  action: MODIFY
---

## Pattern Description

Every active opioid analgesic order must have a matching naloxone
prescription. The skill encodes a four-step procedure: locate opioids
by explicit RxNorm codes, confirm status == active, search for a
naloxone counterpart, and only if absent, create a naloxone
MedicationRequest.

## Common Failure Patterns

- Missing code filter, treating the whole bundle as if it contained
  only opioids.
- Ignoring status, counting completed or entered-in-error orders as
  active.
- Incorrect naloxone code, matching free-text instead of an exact
  CPT/RxNorm code.
- Empty bundle != no opioids; a bundle may carry non-opioid meds and
  each entry must be checked.

## Procedure

### Step 1. Identify active opioid orders

GET /MedicationRequest?patient={patientId}

opioid_codes = [
  "860975",   # hydromorphone (RxNorm)
  "860976",   # oxycodone
  "860977",   # fentanyl
  "860978",   # hydrocodone
  "860979",   # morphine
]
# Keep entries whose medicationCodeableConcept.coding[].code is in
# opioid_codes AND whose status in {active, draft, on-hold}.

### Step 2. Search the same bundle for naloxone (RxNorm 860980)

If absent and the opioid list is non-empty, POST:
```

```
POST /MedicationRequest
{
  "resourceType": "MedicationRequest",
  "status": "active",
  "intent": "order",
  "medicationCodeableConcept": {
    "coding": [{
      "system": "http://www.nlm.nih.gov/research/umls/rxnorm",
      "code":   "860980",
      "display":"Naloxone"
    }]
  },
  "subject": { "reference": "Patient/{patientId}" },
  "authoredOn": "{today_iso}"
}

### Step 3. Report exactly one of three outcomes

FINISH(["No active opioid analgesic orders found ..."])      # case A
FINISH(["Active opioid orders have matching naloxone ..."])  # case B
FINISH(["Naloxone order created for patient ..."])           # case C

## Success Indicators

- Filters coding[].code against the opioid list and checks
  status == active before deciding.
- POSTs naloxone with the exact RxNorm code only when truly missing.
- FINISH message accurately reflects whether a naloxone order was
  created or not.
```

Patient Identifier to FHIR Reference Resolution

(benchmark-procedural, cross-model writer contrast; GPT-5.4 low effort, MedAgentBench; MODIFY)

```
---
name: patient_identifier_to_fhir_reference_resolution
description: Resolve MRN to Patient.id before any dependent FHIR request
tags: [medagentbench, fhir, identifier-resolution]
version: 3
provenance:
  epoch: 4
  action: MODIFY
---

## Pattern Description

When a task supplies an MRN it must be treated as a search key, not a
guaranteed downstream resource reference. The agent must first issue
GET /Patient?identifier=..., read Patient.id from the response, and
only then issue dependent searches or writes that need patient= or
subject.reference.

The skill prevents two distinct failure modes: (i) firing dependent
requests before the patient lookup response arrives, and (ii) reusing
the raw input MRN, stripped digits, placeholders, or template text
where a resolved Patient.id is required.

## Common Failure Patterns

- GET /Observation?patient=S0789363&code=A1C without first parsing
  entry[0].resource.id.
- GET /Observation?patient=6550627&code=A1C after stripping the
  leading 'S'.
- "subject":{"reference":"Patient/"} or "Patient/UNKNOWN" in POST
  bodies.
- Concatenating actions on one line, e.g. GET /Patient... GET
  /Observation... FINISH(...) before the first response arrives.
- Repeating the patient lookup multiple times after one successful
  match instead of caching the resolved id.
```

```
## Recommended Pattern

# 1. Resolve once.
GET /fhir/Patient?identifier={input_identifier}

# 2. Read the bundle.
resolved_id = bundle.entry[0].resource.id

# 3. Use that resolved id in every later step:
#    - searches:  patient={resolved_id}
#    - writes:    "subject": {"reference": "Patient/{resolved_id}"}

## Example

### Correct trajectory

GET /fhir/Patient?identifier=S0789363
   -> entry[0].resource.id = "S0789363"
GET /fhir/Observation?patient=S0789363&code=A1C
POST /fhir/ServiceRequest  {
  ...,
  "subject":{"reference":"Patient/S0789363"}
}
FINISH([5.2, "2022-08-09T15:33:00+00:00"])

### Wrong trajectory

GET /fhir/Observation?patient=6550627&code=A1C   # stripped 'S'
POST /fhir/ServiceRequest {
  "subject":{"reference":"Patient/UNKNOWN"}
}
FINISH([-1])                                    # never resolved

## Success Indicators

- Bundle is read before any dependent action.
- Exactly one patient lookup per task.
- Every later patient= parameter and Patient/{id} reference matches
  entry[0].resource.id; no transformed, blank, or placeholder
  references appear.
```