# FAILURE2SKILL: LEARNING RESIDUAL SKILLS FROM AGENT FAILURES FOR INTERACTIVE TOOL USE

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

## ABSTRACT

Large language model agents can often repair a single mistake when given feedback, yet they rarely turn that mistake into reusable operational knowledge. We study a focused setting for self-improving tool-use agents: after an agent equipped with a current skill library still fails, can the residual failure be compressed into a new skill that transfers to related tasks? We propose Failure2Skill, a failure-driven skill induction pipeline that diagnoses failed trajectories, clusters causal residuals, and writes compact, guarded skill cards for later retrieval. Unlike trace-combination baselines that summarize successful and failed runs together, Failure2Skill treats failures as targeted evidence about missing preconditions, API response shapes, exception paths, and state-update constraints. On a 54-task AppWorld residual-family evaluation using a DeepSeek Chat code agent, Failure2Skill solves 54/54 tasks, while the best combined success/failure skill baseline solves 37/54. Across 18 task families, Failure2Skill wins 8, ties 10, and loses 0; its average verifier pass fraction is 1.000 versus 0.843 for the best combined baseline. Ablations show that removing individual residual rules often collapses final success, including payment-card retry, relation filtering, receipt attachment, note-format preservation, and comment parsing. These results suggest that failed trajectories are not merely negative examples: when localized, they are high-density supervision for building reusable agent skills.

## 1 INTRODUCTION

Interactive agents increasingly solve tasks by writing code, calling APIs, observing errors, and revising their actions (Yao et al., 2023; Wang et al., 2024b; Trivedi et al., 2024). This interaction loop gives the agent information that is absent from the original prompt: the exact API response shape, which identifiers are accepted by state-changing calls, which date interpretation is used by the environment, and which apparently reasonable action creates collateral damage. A natural ambition is to turn this experience into a skill library that improves future behavior without fine-tuning the underlying language model.

Most self-improvement approaches give positive experience a privileged role. Successful trajectories can be summarized into demonstrations or executable skills; reflective methods ask the model to critique and revise outputs after feedback (Shinn et al., 2023; Madaan et al., 2023; Wang et al., 2024a). However, success traces can hide the decision boundary that matters for transfer. A trace may show that a payment request was approved, but not that the agent must retry non-expired funding cards when the first card fails. A trace may show a playlist update, but not that all comments on a selected task contain add/remove operations and that URLs may include trailing punctuation. In these cases, the useful reusable knowledge is concentrated in the residual failure.

We use the term *residual failure* for a failure that remains after the agent already has a relevant skill library. It is not a generic unsolved task. It is the part of the error distribution left over after prior skills have removed easier failure modes. Residual failures are especially valuable because they localize what the current library does not yet encode. In AppWorld (Trivedi et al., 2024), where tasks are evaluated by state-based unit tests and collateral damage checks, these residuals often point to precise missing rules: use the phone app's relationship filter before selecting Venmo users; preserve an existing SimpleNote ledger format before inserting a new month; attach downloaded Venmo receipts when recording Splitwise payments.



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.

This paper asks whether an agent can improve by repeatedly converting such residual failures into reusable skills. We introduce Failure2Skill, a pipeline that runs a code agent with a skill library, diagnoses failures using trajectories and verifier reports, clusters recurring causal residuals, and synthesizes compact skill cards with preconditions, procedural rules, and anti-patterns. At inference time, a retriever selects a small set of relevant skills and injects them into the same agent scaffold.

Our contribution is an empirical and methodological one: (1) we formalize residual failure as a unit of supervision for skill-library growth; (2) we implement a failure-to-skill pipeline for interactive API agents; (3) we compare failure-derived skills against success/failure trace-combination baselines on AppWorld residual families; and (4) we use ablations to show that the induced rules, rather than merely additional tokens, are responsible for the observed gains. On 18 AppWorld task families, Failure2Skill reaches 100.0% success over 54 task instances, compared with 68.5% for the strongest combined baseline available for each family.

## 2 Problem Setup

We consider an interactive task distribution $\mathcal{D}$. A task instance $x_i \sim \mathcal{D}$ is paired with an initial environment state $s_i^0$, an agent-facing natural-language instruction, API documentation, and a hidden verifier. The agent produces a trajectory

$$\tau_i = (o_i^0, a_i^1, o_i^1, \ldots, a_i^T, o_i^T), \tag{1}$$

where actions $a_i^t$ may include generated code and API calls, and observations $o_i^t$ include return values and errors. A verifier $V_i$ maps the final state and trajectory artifacts to a pass fraction,

$$p_i = V_i(\tau_i, s_i^T) \in [0, 1], \tag{2}$$

where $p_i = 1$ denotes full task success and intermediate values denote partial verifier satisfaction. The success rate and average pass fraction over a set $E$ are

$$\mathrm{SR}(E) = \frac{1}{|E|} \sum_{i \in E} \mathbf{1}[p_i = 1], \qquad \mathrm{PF}(E) = \frac{1}{|E|} \sum_{i \in E} p_i. \tag{3}$$

An agent is conditioned on a skill library $\mathcal{L}_t = \{s_1, \ldots, s_m\}$. Each skill is a short natural-language card containing a trigger, a procedure, and guardrails. Given a token budget $B$, retrieval selects

$$R_B(x, \mathcal{L}_t) = \arg \max_{\mathcal{L}' \subseteq \mathcal{L}_t,\ \mathrm{tok}(\mathcal{L}') \le B} \sum_{s \in \mathcal{L}'} \mathrm{rel}(x, s), \tag{4}$$

where rel is a task-skill relevance score. The inference objective is

$$\max_{\mathcal{L}_t} \ \mathbb{E}_{x_i \sim \mathcal{D}_{\mathrm{test}}} \left[ \mathbf{1} \left[ V_i(A(x_i, R_B(x_i, \mathcal{L}_t))) = 1 \right] \right]. \tag{5}$$

**Residual Failures.** Let $D(\tau_i, V_i, \mathcal{L}_t)$ be a diagnosis function that maps a failed run to a causal description $d_i$ of the missing behavior. We call $i$ a residual failure at iteration $t$ when

$$i \in \mathcal{R}_t \quad \Longleftrightarrow \quad p_i < 1 \ \ \wedge \ \ \neg \mathrm{covered}(d_i, \mathcal{L}_t). \tag{6}$$

Intuitively, a residual failure is the part of the task still unsolved after retrieved skills have already shaped the agent. This definition separates residual repair from ordinary prompt expansion: the next skill should encode the missing causal rule, not repeat the full task solution.

## 3 Failure2Skill

Failure2Skill is a closed-loop procedure for growing a skill library from failed trajectories. Figure 1 gives the pipeline. The system keeps the base language model and agent scaffold fixed; learning occurs through auditable text skills.

**Step 1: Run with the Current Library.** For each task, the agent receives the instruction, API documentation, and retrieved skills $R_B(x, \mathcal{L}_t)$. It can issue code actions and complete the task. We log tool calls, return values, exception traces, final answers, and verifier reports.

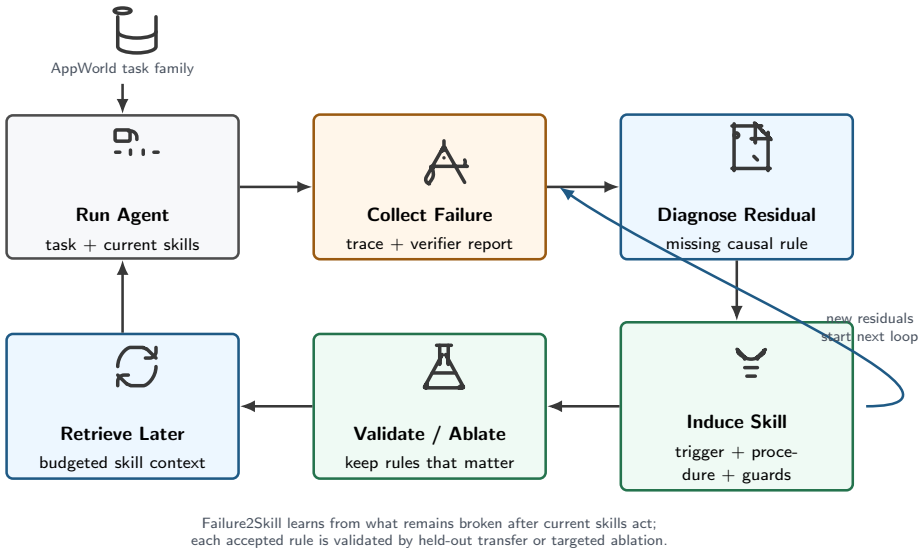


Figure 1: Failure2Skill converts failed interactive trajectories into reusable skill cards. Failed runs are diagnosed, clustered into residual failure modes, compressed into guarded skills, validated through ablations or held-out families, and retrieved for later tasks under a token budget.

**Step 2: Diagnose the Residual.** For failed runs, a diagnostic pass compares the instruction, trajectory, and verifier evidence. The output is not a broad natural-language reflection, but a candidate causal atom: for example, "used the wrong direction field for received payment requests", "updated a note without preserving the existing month-order convention", or "selected the first payment card and did not retry another valid card after the API rejected it". Diagnostics are grouped into a taxonomy covering response-shape errors, credential/login errors, pagination coverage, target-set construction, temporal parsing, state-completion semantics, and collateral state damage.

**Step 3: Cluster and Compress.** Recurring diagnoses are clustered by whether the same procedural intervention would have prevented them. For a cluster $C$, Failure2Skill writes a skill

$$s_C = g(C) = \{\text{trigger}, \text{procedure}, \text{guards}, \text{anti-patterns}\}. \tag{7}$$

The trigger states when the skill applies; the procedure states concrete API-level steps; guards state conditions that prevent negative transfer; anti-patterns name tempting but wrong shortcuts observed in the failures.

**Step 4: Validate and Gate.** Candidate skills are evaluated on source-family repairs and, when available, related held-out families. We keep success rate, completion rate, pass fraction, token cost, and negative transfer indicators. Ablations remove one rule at a time from a successful skill. A rule is treated as causally important when removing it preserves most of the prompt budget but degrades final success or pass fraction.

**Step 5: Retrieve at Test Time.** At inference, tasks retrieve only the most relevant skills. This keeps Failure2Skill distinct from appending an ever-growing diary to the prompt. In the current experiments, retrieval is family-aware and budgeted; scaling to automatic semantic retrieval is future work.

## 4 Experimental Design

**Benchmark.** We evaluate on AppWorld (Trivedi et al., 2024), an execution environment of everyday applications with realistic users, API tools, and state-based unit-test verifiers. AppWorld is a useful stress test for residual skills because many failures are not visible in the final answer alone: a run

Table 1: Best-vs-best AppWorld residual-family evaluation. Each family has three task instances. Failure2Skill uses failure-derived skills; Combined uses the strongest success/failure trace-combination baseline available for that family.

| Method | Families | Successes | Success rate | Avg. pass | Avg. tokens |
|---|---|---|---|---|---|
| Failure2Skill | 18 | 54/54 | 100.0% | 1.000 | 1265 |
| Combined | 18 | 37/54 | 68.5% | 0.843 | 707 |
| Family wins for Failure2Skill | 18 | 8 wins, 10 ties, 0 losses | | | |

Table 2: Results by task group. All groups use the same three-instance-per-family evaluation protocol.

| Task group | Families | Failure2Skill successes | Combined successes |
|---|---|---|---|
| Cross-app notes/debts | 5 | 15/15 | 8/15 |
| Payments and Venmo | 6 | 18/18 | 14/18 |
| Spotify | 5 | 15/15 | 12/15 |
| Todoist workflows | 2 | 6/6 | 3/6 |
| Total | 18 | 54/54 | 37/54 |

can choose the wrong user, mutate extra state, omit a receipt attachment, or satisfy only part of a multi-app workflow.

**Agent and Model.** All experiments use the same OpenAI-compatible code-agent scaffold with 'deepseek-chat', a maximum of two interaction turns, and a 4096-token generation cap. The model is not fine-tuned. Skills are inserted as text into the agent context. We report both final success and average verifier pass fraction.

**Task Families.** The main comparison uses 18 AppWorld task families, each evaluated on three task instances, for 54 total runs per method. The families cover Spotify library reasoning and updates, Venmo payment and aggregation workflows, SimpleNote-to-Venmo/Splitwise transfers, Todoist comment-driven reassignment, and cross-app receipt handling. Most are from AppWorld 'test_normal'; two training-split families are included where they serve as source-family transfer or controlled comparison cases. We report family-level results rather than claiming full-benchmark AppWorld performance.

**Methods.** We compare the following methods. **No Skill** runs the same agent scaffold without learned skills and is used to identify initial bottlenecks. **Failure2Skill** uses failure-derived residual skills. **Combined** and **LLM Combined** are trace-combination baselines that compress successful and failed traces into a single skill-like prompt. For the main table, we use the best available combined baseline for each family, including manually structured combined skills and LLM-compressed combined skills. **Ablations** remove individual rules from a successful skill while keeping the remaining card intact.

## 5 RESULTS

Table 1 summarizes the main finding. On the 18-family residual evaluation, Failure2Skill solves every task instance, while the best combined baseline solves 37 of 54. The absolute success-rate gap is 31.5 percentage points. The pass-fraction gap is smaller but still substantial, 1.000 versus 0.843, indicating that many combined-baseline failures make partially correct state changes but miss one critical residual rule. Failure2Skill uses more prompt budget, averaging 1265 skill tokens versus 707 for the combined baselines, but the additional text is concentrated in executable guards and exception-handling rules.

Table 2 shows that the gains are not isolated to one app. The largest gaps occur in cross-app workflows, where success depends on translating information across notes, payments, shared expenses, files, and comments. These tasks reward local procedural knowledge: which field identifies the counterparty,

Table 3: Representative family-level wins. 'Pass' is average verifier pass fraction for the combined baseline.

| Family | Failure2Skill | Combined | Combined pass |
|---|---|---|---|
| Approve this-month Venmo requests from roommates | 3/3 | 0/3 | 0.857 |
| Make Venmo payments/requests from a trip-debt note | 3/3 | 1/3 | 0.524 |
| Record Splitwise payments from Venmo receipts | 3/3 | 0/3 | 0.333 |
| Update Spotify playlist titles with dominant genre | 3/3 | 1/3 | 0.556 |
| Apply Todoist playlist comments to Spotify playlists | 3/3 | 0/3 | 0.700 |
| Update SimpleNote monthly Venmo expense log | 3/3 | 2/3 | 0.700 |

Table 4: Selected ablations. Removing a single residual rule often breaks final success while keeping most of the skill card.

| Family | Removed rule | Success | Avg. pass |
|---|---|---|---|
| Payment requests | Retry non-expired payment cards after API rejection | 0/3 | 0.857 |
| Payment request + withdraw | Remove temporal traces from combined baseline | 0/3 | 0.182 |
| Trip-debt note to Venmo | Map 'I owe' / 'owes me' to payment direction | 0/3 | 0.429 |
| Trip-debt note to Venmo | Search Venmo users beyond friend list | 0/3 | 0.619 |
| SimpleNote expense log | Preserve 'MM/YY =¿ $amount' note schema | 0/3 | 0.433 |
| Splitwise + Venmo receipts | Attach downloaded receipt path to payment record | 0/3 | 0.467 |
| Todoist reassignment | Flatten nested 'sections[].tasks' response | 0/3 | 0.222 |
| Todoist + Spotify | Parse add/remove lines from every task comment | 0/3 | 0.267 |

how to map "I owe" versus "owes me" into Venmo action direction, or how to attach downloaded receipts to Splitwise payments.

The representative wins in Table 3 illustrate why residual skills matter. The combined baseline for approving current-month payment requests obtains a high average pass fraction, yet fails all three final tasks. The missing rule is not the broad payment-request concept; it is the operational fallback of trying non-expired payment-card identifiers until the approval API accepts one. Similarly, the Todoist–Spotify workflow partially identifies the relevant project and playlist, but fails when it does not parse every add/remove comment or robustly extract playlist IDs from pasted URLs.

## 6 Ablations: What the Residual Rules Encode

Table 4 reports targeted ablations. These are not broad prompt-length controls: each row removes a concrete residual rule while leaving the rest of the skill largely intact. The failures show that Failure2Skill is not merely providing additional domain exposition. It is encoding narrow operational constraints that are hard to infer from the task instruction alone.

Three patterns recur. First, many residuals are *API semantics* failures: the model knows what to do, but calls an endpoint with the wrong identifier, direction, login field, or response-shape assumption.

Second, residuals are often *target-set* failures: the model acts on a plausible user, note, playlist, or group, but the verifier requires the exact set and rejects collateral changes. Third, residuals include *format-preservation* failures: the model computes the right quantity but writes it into the wrong persistent schema. Success traces are weak supervision for these boundaries because they show one legal path, whereas the failure plus verifier report reveals which boundary was crossed.

## 7 Discussion

**Why Failure-Derived Skills Transfer.** The useful skill is frequently smaller than the task. For example, "approve this month's payment requests from roommates" combines relation lookup, payment-request direction, temporal filtering, and payment-card fallback. A monolithic demonstration can conflate these. A residual skill isolates one missing operation and can therefore transfer to another family, such as approving requests and then withdrawing remaining balance. This modularity explains why Failure2Skill can tie combined baselines on families where broad trace summaries suffice, while winning on families with brittle API or state constraints.

**Why Combined Traces Fail.** Combining successful and failed trajectories is informative, but it can blur the causal role of a small branch. If a successful trace happens to use a valid payment card, the summary may never say what to do when the first card is rejected. If one Todoist task has one comment, a combined summary may not require scanning all comments. Failure2Skill starts from the branch that broke, making exception-handling and guard conditions first-class.

**Token Cost.** Failure2Skill uses more skill tokens on average than combined baselines. This is expected because residual cards contain triggers, procedures, and anti-patterns. The cost is still modest relative to the 4096-token generation cap, and the retrieval setting means the full library is not always loaded. Future work should study compression, automatic rule deduplication, and dynamic skill selection under stricter budgets.

## 8 Related Work

**LLM Agents and Tool Use.** ReAct introduced interleaved reasoning and acting for language-model agents (Yao et al., 2023). Toolformer showed that language models can learn to invoke external tools from self-supervised signals (Schick et al., 2023). CodeAct studies executable code as a flexible action space for agents (Wang et al., 2024b). AppWorld provides the execution and evaluation environment used in this paper, with stateful applications, many APIs, and state-based unit tests (Trivedi et al., 2024). Failure2Skill is complementary to these agent scaffolds: it changes how experience is converted into reusable prompt-time knowledge.

**Reflection and Self-Improvement.** Reflexion stores verbal feedback from previous trials (Shinn et al., 2023), and Self-Refine improves a model's output through iterative feedback and revision (Madaan et al., 2023). Voyager builds an open-ended library of executable skills in Minecraft (Wang et al., 2024a). These systems demonstrate the value of memory and feedback. Failure2Skill focuses specifically on failed interactive API trajectories and asks how to compress them into small guarded skills whose causal contribution can be tested by ablation.

**Skill Libraries.** Skill libraries can improve reuse but also risk retrieval noise, over-broad rules, and negative transfer. Failure2Skill addresses this by learning from residuals, adding explicit triggers and anti-patterns, and validating individual rules. The current study is a small but concrete step toward auditable, failure-driven library growth.

## 9 Limitations

This study is deliberately focused. It evaluates 18 residual AppWorld families, not the full AppWorld benchmark. Family selection and some final skill-card acceptance were researcher-audited in the current prototype; a stronger system should automate candidate gating and held-out validation. The experiments use one base model and one agent scaffold, so model-scale generality remains

open. Finally, AppWorld is synthetic, although stateful and richly structured; deployment in real personal-data systems would require stronger safety, permissions, and privacy controls.

## 10 CONCLUSION

Failure2Skill treats failed trajectories as high-density supervision for self-improving agents. By diagnosing residual failures, clustering recurring missing behaviors, and writing compact guarded skills, it turns brittle API and state-update mistakes into reusable prompt-time knowledge. In a focused AppWorld residual-family study, this approach substantially outperforms best available combined success/failure trace summaries and withstands targeted ablations. The central lesson is simple: for interactive agents, what remains broken after the current skill library acts may be the cleanest signal for what the library should learn next.

## REPRODUCIBILITY STATEMENT

All reported numbers in the paper are derived from saved AppWorld run artifacts in the accompanying repository. The main tables are generated from `experiments/appworld_deepseek_method_comparison.csv`, `experiments/appworld_deepseek_main_table.csv`, and `experiments/appworld_deepseek_ablation_table.csv`. Each row records the dataset split, family identifier, task offset, method name, metrics path, success count, completion rate, average pass fraction, and skill-token count. The model used in these runs is 'deepseek-chat' through an OpenAI-compatible API endpoint. The repository contains no API keys; credentials must be provided through environment variables at run time.

## ETHICS STATEMENT

The experiments use AppWorld's synthetic applications and fictitious users. The skills studied here can operate state-changing APIs, so real-world deployment would require strict user authorization, audit logs, rollback mechanisms, and conservative retrieval gates. The current work is intended as a controlled study of agent learning from failure, not as a permission model for real personal data.

## AI USE STATEMENT

Generative AI tools were used as writing and coding assistants during preparation of this draft, including help with experiment summarization, LaTeX drafting, and figure construction. All experimental results reported here come from saved run artifacts and verifier outputs, and the authors are responsible for the final content.

## REFERENCES

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, Shashank Gupta, Bodhisattwa Prasad Majumder, Katherine Hermann, Sean Welleck, Amir Yazdanbakhsh, and Peter Clark. Self-refine: Iterative refinement with self-feedback. In *Advances in Neural Information Processing Systems*, 2023. URL https://papers.nips.cc/paper_files/paper/2023/hash/91edff07232fb1b55a505a9e9f6c0ff3-Abstract-Conference.html.

Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. Toolformer: Language models can teach themselves to use tools. In *Advances in Neural Information Processing Systems*, 2023. URL https://papers.nips.cc/paper_files/paper/2023/hash/d842425e4bf79ba039352da0f658a906-Abstract-Conference.html.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. In *Advances in Neural Information Processing Systems*, 2023. URL https://papers.nips.cc/paper_files/paper/2023/hash/1b44b878bb782e6954cd888628510e90-Abstract-Conference.html.

Harsh Trivedi, Tushar Khot, Mareike Hartmann, Ruskin Manku, Vinty Dong, Edward Li, Shashank Gupta, Ashish Sabharwal, and Niranjan Balasubramanian. AppWorld: A controllable world of apps and people for benchmarking interactive coding agents. In Lun-Wei Ku, Andre Martins, and Vivek Srikumar (eds.), *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 16022–16076, Bangkok, Thailand, August 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.acl-long.850. URL https://aclanthology.org/2024.acl-long.850/.

Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. Voyager: An open-ended embodied agent with large language models. *Transactions on Machine Learning Research*, 2024a. URL https://openreview.net/forum?id=ehfRiF0R3a.

Xingyao Wang, Yangyi Chen, Lifan Yuan, Yizhe Zhang, Yunzhu Li, Hao Peng, and Heng Ji. Executable code actions elicit better llm agents. In *Proceedings of the 41st International Conference on Machine Learning*, 2024b. URL https://proceedings.mlr.press/v235/wang24i.html.

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*, 2023. URL https://openreview.net/forum?id=WE_vluYUL-X.

# A Full Family-Level Results

Table 5: Full best-vs-best comparison across the 18 residual families.

| Family | Split | Failure2Skill | Combined | Winner |
|---|---|---|---|---|
| CSV debts to Venmo/Splitwise receipts | test_normal | 3/3 | 3/3 | Tie |
| Approve current-month payment requests | test_normal | 3/3 | 0/3 | Failure2Skill |
| Approve requests and withdraw balance | test_normal | 3/3 | 3/3 | Tie |
| Dinner-share Venmo requests from notes | train | 3/3 | 3/3 | Tie |
| SimpleNote trip expenses to Splitwise | test_normal | 3/3 | 1/3 | Failure2Skill |
| Trip-debt note to Venmo payments/requests | test_normal | 3/3 | 1/3 | Failure2Skill |
| Splitwise payments from Venmo receipts | test_normal | 3/3 | 0/3 | Failure2Skill |
| Most/least followed Spotify artist | test_normal | 3/3 | 3/3 | Tie |
| Liked-song genre by Spotify source | test_normal | 3/3 | 3/3 | Tie |
| Rank Spotify playlist artists | test_normal | 3/3 | 3/3 | Tie |
| Update Spotify playlist titles by genre | test_normal | 3/3 | 1/3 | Failure2Skill |
| Spotify title ranking by source | train | 3/3 | 2/3 | Failure2Skill |
| Todoist reassignment from comments | test_normal | 3/3 | 3/3 | Tie |
| Todoist comments to Spotify playlists | test_normal | 3/3 | 0/3 | Failure2Skill |
| Venmo bill amount by category/year | test_normal | 3/3 | 3/3 | Tie |
| SimpleNote monthly Venmo expense log | test_normal | 3/3 | 2/3 | Failure2Skill |
| Venmo monthly sent/received totals | test_normal | 3/3 | 3/3 | Tie |
| Venmo requested amount over day windows | test_normal | 3/3 | 3/3 | Tie |

# B Additional Ablation Results

# C Artifact Map

Table 6: Additional rule-removal ablations. Full skills solve 3/3 in each listed family.

| Family | Removed rule | Success | Avg. pass |
|---|---|---|---|
| Spotify playlist artists | Credential handling and Spotify username login | 0/3 | 0.500 |
| Spotify liked genre | All residual skills beyond the shape skill | 2/3 | 0.833 |
| Spotify liked genre | Shape/login skill while keeping residual skills | 0/3 | 0.500 |
| Payment request source family | Phone login via supervisor phone number | 0/3 | 0.167 |
| Payment request source family | Phone-contact 'relationship' filter | 0/3 | 0.667 |
| Payment request + withdraw | Payment-card retry | 2/3 | 0.939 |
| SimpleNote trip expenses to Splitwise | Relation filter for target trip groups | 0/3 | 0.433 |
| Todoist reassignment | Comment-intent predicate | 0/3 | 0.407 |
| Todoist + Spotify | Relation-only project fallback | 2/3 | 0.733 |
| Todoist + Spotify | Playlist-ID regex for pasted URLs | 2/3 | 0.900 |
| CSV debts | Venmo payment-card funding and retry | 0/3 | 0.789 |
| CSV debts | Splitwise receipt attachment | 0/3 | 0.947 |
| Venmo bill amount | Bill-description filter | 0/3 | 0.500 |
| Venmo bill amount | Temporal parsing | 1/3 | 0.667 |
| Venmo monthly totals | Temporal parsing | 2/3 | 0.833 |

Table 7: Primary artifacts used to prepare the submission draft.

| Artifact | Purpose |
|---|---|
| `experiments/appworld_deepseek_main_table.csv` | Source table for no-skill, Failure2Skill, combined, and ablation rows. |
| `experiments/appworld_deepseek_method_comparison.csv` | Best-vs-best family comparison used in Tables 1, 2, and 5. |
| `experiments/appworld_deepseek_ablation_table.csv` | Rule-removal ablations used in Tables 4 and 6. |
| `experiments/failure2skill_pipeline_figure.pdf` | Pipeline figure with Lucide-style icons. |
| `runs/appworld_skill_libraries/` | Skill libraries evaluated in the reported runs. |
| `runs/appworld_deepseek_chat_*` | Metrics, trajectories, reports, and failure logs for each AppWorld run. |