Linköping University | Department of Computer and Information Science
Master's thesis, 30 ECTS | Datateknik
2017 | LIU-IDA/LITH-EX-A--2017/001--SE

Learning from Agentic Traces

– Domain adaption for multi-hop tool use agents

Inlärning från resonemangspår

Mathias Ahlgren
Mårten Saltin

Supervisor : Emil Wiman
Examiner : Mattias Tiger

Linköpings universitet
SE–581 83 Linköping
+46 13 28 10 00 , www.liu.se

Upphovsrätt

Detta dokument hålls tillgängligt på Internet - eller dess framtida ersättare - under 25 år från publiceringsdatum under förutsättning att inga extraordinära omständigheter uppstår.

Tillgång till dokumentet innebär tillstånd för var och en att läsa, ladda ner, skriva ut enstaka kopior för enskilt bruk och att använda det oförändrat för ickekommersiell forskning och för undervisning. Överföring av upphovsrätten vid en senare tidpunkt kan inte upphäva detta tillstånd. All annan användning av dokumentet kräver upphovsmannens medgivande. För att garantera äktheten, säkerheten och tillgängligheten finns lösningar av teknisk och administrativ art.

Upphovsmannens ideella rätt innefattar rätt att bli nämnd som upphovsman i den omfattning som god sed kräver vid användning av dokumentet på ovan beskrivna sätt samt skydd mot att dokumentet ändras eller presenteras i sådan form eller i sådant sammanhang som är kränkande för upphovsmannens litterära eller konstnärliga anseende eller egenart.

För ytterligare information om Linköping University Electronic Press se förlagets hemsida `http://www.ep.liu.se/`.

Copyright

The publishers will keep this document online on the Internet - or its possible replacement - for a period of 25 years starting from the date of publication barring exceptional circumstances.

The online availability of the document implies permanent permission for anyone to read, to download, or to print out single copies for his/hers own use and to use it unchanged for non-commercial research and educational purpose. Subsequent transfers of copyright cannot revoke this permission. All other uses of the document are conditional upon the consent of the copyright owner. The publisher has taken technical and administrative measures to assure authenticity, security and accessibility.

According to intellectual property law the author has the right to be mentioned when his/her work is accessed as described above and to be protected against infringement.

For additional information about the Linköping University Electronic Press and its procedures for publication and for assurance of document integrity, please refer to its www home page: `http://www.ep.liu.se/`.

© Mathias Ahlgren
Mårten Saltin

Abstract

Large language models are increasingly deployed as agents that reason and call tools to solve multi-step tasks. In deployment these agents generate rich execution traces, the reasoning and tool calls behind each answer, and then discard them, so the same mistakes recur and successful behaviour is never reused. Adapting a deployed agent to its domain normally requires labeled supervision or a verifiable reward on the final answer, neither of which is available for a multi-tool agent operating on an open-ended domain.

This thesis investigates whether an agent's own traces can be reused to improve it without gold labels, using only quality judgments produced by a larger language model from the traces themselves. A blind, rubric-based judge scores each trace on process-quality dimensions without ever seeing the reference answer, and this signal drives three trace-reuse pipelines on a Qwen3.5-9B agent: filtered supervised fine-tuning, in-context memory rule curation, and replay-based preference learning. The judge and the pipelines are evaluated across structurally different benchmarks, HotPotQA, ^{2}-bench, and StableToolBench, with ToolHop as an additional validation domain. reference answer, and this signal drives three trace-reuse pipelines on a Qwen3.5-9B agent: filtered supervised fine-tuning, in-context memory rule curation, and replay-based preference learning. The judge and the pipelines are evaluated across structurally different benchmarks, HotPotQA, ^{2}-bench, and StableToolBench, with ToolHop as an additional validation domain.

The blind judge produces a usable signal that tracks ground-truth success, with a strength bounded by how far a benchmark's failures are visible in the trace, strong on tool-composition tasks and weak where success depends on hidden environment state. Converting that signal into downstream task success proves harder. Filtered fine-tuning does not raise task success and regresses on retail, memory rules help only an agent strong enough to follow them, and replay-based preference learning improves only single-tool tasks without transferring. The thesis contributes a validated label-free quality signal and a diagnosis of the specific reasons it does not yet convert into reliable gains on a small open agent.

Acknowledgments

First of all we are grateful to our supervisor at Linköping University, Emil Wiman, for supporting us throughout the project with valuable discussions and feedback, and for proofreading and helping us with the writing of this thesis. We also thank our examiner, Mattias Tiger, for the ideas and inspiring knowledge that helped us turn our loose ideas into a defined research direction. Finally, we would like to give a large thanks to FindMyFactory for supporting our thesis with resources and for giving us the opportunity to work on an open research project rather than asking for a concrete deliverable.

Contents

List of Figures

List of Tables

Keywords

$pass^k$ Agent reliability metric (from τ-bench): the fraction of tasks for which *all* k independent rollouts succeed. Unlike *pass@k* (success if *any* of k attempts succeeds), $pass^k$ rewards consistency, one failure among the k trials marks the task unsolved, so it degrades sharply as k grows and exposes a policy's run-to-run instability

A100 NVIDIA A100 Tensor-Core data-centre GPU (40/80 GB HBM2e); the single accelerator the 9B agent and its LoRA fine-tuning fit on

bf16 bfloat16: a 16-bit float with an 8-bit exponent (the dynamic range of fp32) and a 7-bit mantissa; halves memory versus fp32 while avoiding fp16 overflow

MCP Model Context Protocol; the interface each benchmark exposes its tools through

TRL Transformer Reinforcement Learning (Hugging Face); provides the SFTTrainer and DPOTrainer used in Tracks 1 and 3

Unsloth LoRA/QLoRA parameter-efficient fine-tuning library with fused kernels

vLLM high-throughput open-source LLM inference/serving engine whose PagedAttention KV-cache management exposes an OpenAI-compatible API

1 Introduction

Large language models (LLMs) are increasingly deployed as *agents*, these systems combine reasoning and tool use to execute multi-step tasks autonomously. Unlike the standalone language models from which they are built, agents are typically not trained end-to-end. The common setup is a foundation model paired with a system prompt, a tool registry, and a target domain, and the resulting agent is deployed without the model itself ever having seen that configuration during training. The agent then executes tasks, generates rich traces of reasoning and tool use, and discards them between sessions. Successful trajectories are never learned from and failures are not retained, the same mistakes recur on consecutive runs.

This thesis focuses on a class of agent tasks that are referred to as *multi-hop tool problems*, tasks where success depends on executing a sequence of tool calls, where each call's output influences the next tool call, and where a single misstep can invalidate the agent's trajectory. We evaluate multi-hop question answering [32] that assesses the model's capability to find the connection between two entities through a series of searches and lookups using the HotPotQA benchmark. We also evaluate the popular multi-tool customer service benchmark Tau-2's telecom, retail and retails settings [33] as well as Stabletoolbench [8] for their wide tool calling and long horizon tasks and Toolbench for its tool calling.

The central question of this thesis is therefore whether the execution traces, the reasoning and tool calls an agent generates, of a smaller open-source agent on multi-hop tool problems can be reused to improve that agent without access to golden labels, using only quality judgments produced by larger language models from the agent's own executions, and not the gold answers themselves.

Closing this loop is worth doing because the gains arrive right away and build over time. An agent that learns from its own judged traces stops repeating the mistakes that break multi-hop tasks. It drops the redundant or exploratory tool calls that a fixed, deployed agent would otherwise make on every run, and it reaches the right answer more often and in fewer steps. Every call it avoids is also tokens it avoids, so the savings are real: lower inference cost, less waiting for the user, and less energy per task. This adds up at the scale agents now run.

The rule that this signal uses no gold label is what makes the loop practical rather than just appealing. Labeling multi-hop traces by hand is slow and expensive, a single trace can run to tens of thousands of tokens of reasoning and tool input and output, and in the specific domains where agents actually run, the data often cannot leave the organization to be labeled at all. Taking the signal from the agent's own runs instead lets it improve using user stories,

stay inside the deployment, and track the real task distribution, including the rare cases that curated benchmarks never reach.

Answering this question requires a feedback signal that does not depend on golden labels. We curate the signal from a *blind judge*, a larger language model that scores a trajectory from the trace alone, without ever seeing the reference answer. This turns the discarded trace into a reusable improvement signal (Figure 1.1), and raises a second question of how the signal is best applied. We study three trace reuse strategies that are parametric and not parametric. Filtered supervised fine-tuning, which retrains the agent on its own judge-approved trajectories, a memory curation loop which creates reusable rules created at inference time without touching the weights, and replay-based prefrence learning, which pairs an agent's failed trajectory with a corrected replay and optimizes for the corrected.

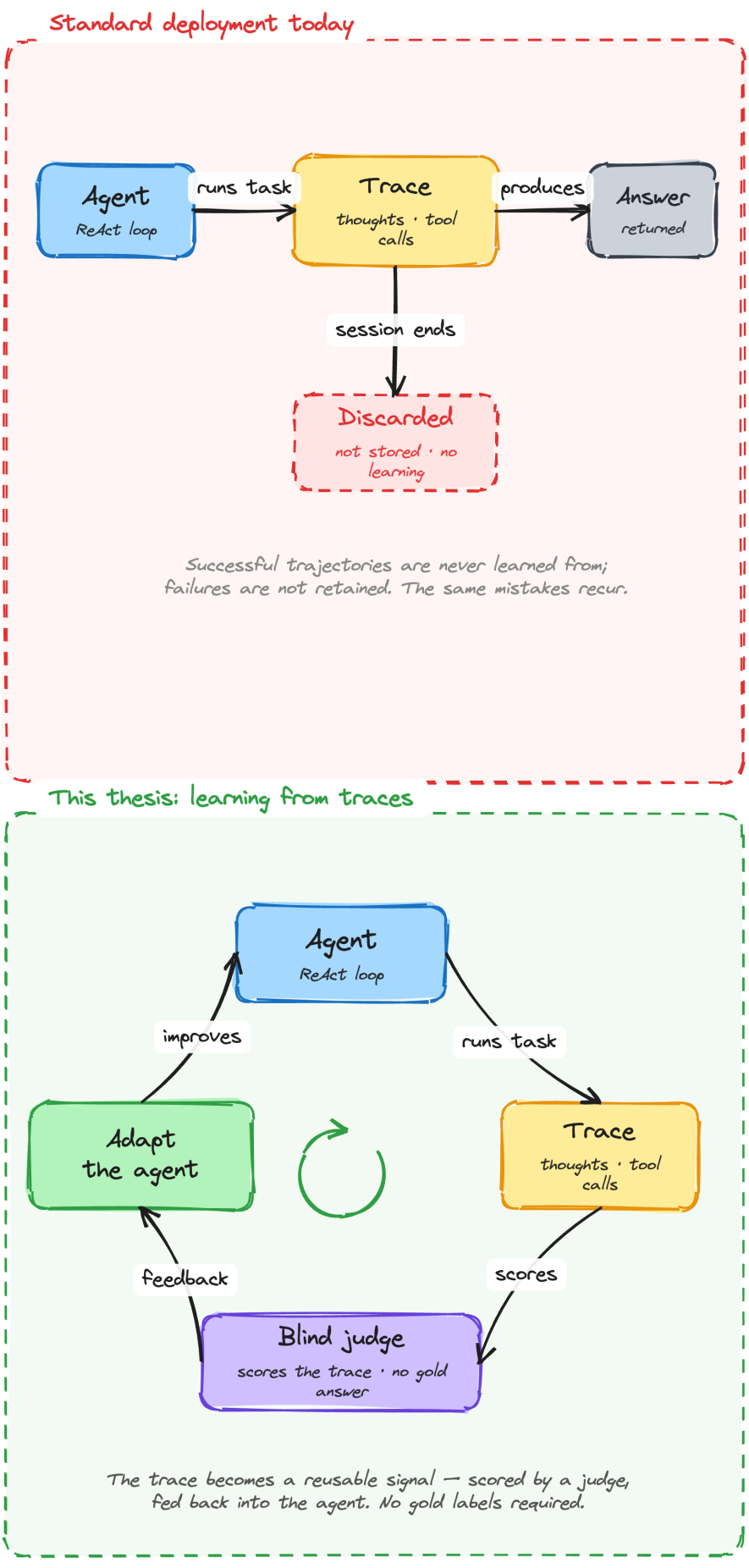


Figure 1.1: The trace lifecycle. Deployment discards the agent's trace (above). This thesis reuses it as a blind-judged learning signal (below)

1.1 Motivation

The standard approaches to adapting language models for new tasks rely on labeled supervision. Reinforcement learning from human feedback [13], direct preference optimisation [17], and verifiable-reward methods such as Group Relative Policy Optimisation [18] all require either annotated preference pairs or a verifiable reward on the final answer. Neither is available for a deployed agent operating on a specific multi-hop tool domain, gathering preference data at the scale of trace volumes is impractical as traces can span tens of thousands of tokens, and the absence of a single correct answer rules out verifiable rewards.

The work on continual learning for agent's has begun to address this gap. Reflexion [19] first introduced the concept of verbal reinforcement learning, in which an agent reflects on failures and stores insights as natural-language summaries. ExpeL [41] continued on this idea by extracting reusable experiences across tasks, several works has built on this ACE, GEPA, ReasoningBank [40] [1] [14] and has refined how such memory is created and retrieved. Each of these approaches, however, require either a clear task boundary, a binary success/failure label or a set of training tasks.

This thesis takes the opposite starting point, the situation an online deployment is in, with an agent serving traffic. For every task, a full trace is generate that contains which tools it called in which order, with what arguments and its reasoning in the execution. Today that trace is stored and largely discarded, nothing in the deployment loop turns it into an improvement signal. The question this thesis pursues is whether those traces can be reused in an offline setting with no golden labels and no environmental reward, by having a separate model read each trace and score it on process-quality dimensions, and then improving from that score. The premise that a process-level signal is informative is not new, in mathematical reasoning, supervising a solution's step rather than only its final answer reduces errors and yields better results [23, 11]. The difference here is the constraint, those methods had step-level labels and an outcome to check against, a deployed multi-tool agent has neither, so the only signal available is one computed over the trajectory itself, and that is the signal this thesis builds on.

1.2 Aim

The aim of this thesis is to investigate whether a smaller open-source agent can be improved continually and adapted to specific multi-hop tool domains by reusing its own execution traces, with trajectory quality assessed by a blind multi-dimensional rubric-based LLM judge, that scores traces along process-quality dimensions without ever seeing the golden answer.

Specifically, we aim to:

1. Design and validate the blind judge, demonstrating that it produces a useful curation signal across structurally different multi-hop tool benchmarks.
2. Use the judge to drive three downstream trace-reuse pipelines: filtered supervised fine-tuning, memory rule curation, and replay-based preference learning that reconstructs corrective trajectories from failed traces.

1.3 Research questions

1. To what extent can a blind, multi-dimensional, rubric-based LLM judge reliably classify the quality of agentic traces without access to gold answers, and does the resulting signal transfer across structurally different multi-hop tool benchmarks?
2. To what extent can filtered supervised fine-tuning, using the blind judge's quality signal, improve agent performance on multi-hop tool benchmarks?

3. How can memory-based curation through in-context rule injection adapt an agent to a new domain?

4. To what extent can replay-based preference learning, using corrective interventions guided by the judge, improve an agent's performance on multi-hop tool benchmarks?

1.4 Delimitations

This thesis operates within the following scope. The agent is restricted to a single base model, primarily Qwen3.5-9B from the Qwen 3 series [31].

The agent architecture is restricted to the ReAct [34] architecture, we do not investigate hierarchical agents, multi-agent systems, beyond the single-loop case. The training methods are restricted to supervised fine-tuning with LoRA [10] and Direct Preference Optimisation [17].

All experiments are conducted in English. The judge used, is a proprietary large language model from a frontier lab such as OpenAI and is the only quality signal in the pipeline, the data used comes only from the model's generated outputs. The study is conducted within the technological capabilities of large language models as of early 2026.

The work has been done with the startup Findmyfactory AB which has provided the resources for running the models and provided tokens for the API usage of proprietary LLMs.

1.5 Contributions

The contributions of this thesis are the following.

- A blind, multi-dimensional, rubric-based LLM judge that scores agent traces on process quality without ever seeing the golden answer, validated across four structurally different benchmark settings (HotPotQA, τ^2-bench retail, τ^2-bench telecom, and ToolHop).
- An analysis of where this signal is reliable and where it is not, showing that its strength is bounded by how far a benchmark's failures are visible in the trace the judge reads, strong on tasks whose outcome the trace exposes.
- Evaluating three approaches of trace-reuse pipelines built on the judge signal, filtered supervised fine-tuning, in-context memory rule curation, and replay-based preference learning.

1.6 Thesis outline

The thesis is structured as follows. Chapter 2 presents the theory the work builds on, covering agentic language models, continual learning, and the evaluation and reuse of agent traces, and reviews the related work the thesis positions itself against. Chapter 4 describes the method, the agent and judge setup, the benchmarks, and the three trace-reuse tracks. Chapter 5 reports the results, first the validation of the blind judge and then each track in turn. Chapter 6 interprets these results, reviews the methodology, and situates the work in a wider context. Chapter 7 answers the research questions and outlines future work. Supplementary tables are collected in Chapter 8.

2 Theory

This chapter introduces the concepts needed to understand the three trace-reuse pipelines studied in this thesis. Section 2.1 introduces language models as autoregressive policies. Section 2.2 extends this view to tool-using ReAct agents and defines an agentic trace. Section 2.3 explains continual adaptation and distinguishes parametric updates from non-parametric memory. Section 2.4 motivates process-level judging of traces when golden answers are unavailable. Section 2.5 introduces the training objectives used to turn judged traces into model updates.

2.1 Large language models

A large language model (LLM) is a neural network trained to predict the next token in a sequence of text. Repeating this prediction one token at a time then appending each token to the input and predicting the next.

$$\pi_\theta(y \mid x) \;=\; \prod_i \pi_\theta(y_i \mid x,\, y_{<i})\,, \tag{2.1}$$

where x is the input (a prompt) and y the generated output (a completion). We write π_θ for the model with parameters θ given a prompt it produces a completion.

Modern LLMs are decoder only transformers [25] each combining multiple masked multi-head self-attention layers with feed-forward layers. The attention layer is where tokens exchange information, each token attends to itself and every preceding token, the influence of one token on another is computed by scaled dot-product attention,

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^\top}{\sqrt{d_k}}\right) V, \tag{2.2}$$

where Q, K and V hold the tokens' query, key and value vectors. The naming follows a key-value lookup, each token's query Q is matched against every other token's key K via scaled dot product, and the resulting similarity weights determine how much of each token's value V contributes to the final output.

2.1.1 Prompt engineering and In-context learning

Because an LLM's output is conditioned on the whole prompt and on the tokens it has generated so far, anything placed in the prompt can shape what the model does next. Brown et al. [3] showed that this goes well beyond following an instruction. A language model can be shown a handful of examples of a task in the prompt. The model can then learn from these examples and carry out the task based on them. This capability of a LLM to learn from what it is injected into its prompt is often referred to as in-context learning and can be used for teaching and helping an LLM execute the given tasks. Today with models spanning context windows of up to a million tokens a popular subfield of prompt engineering is context engineering. Context engineering revolves around techniques to keep the most relevant context in the context window at all times, such as retriever just what is needed for solving the current task, removing stale context or tool results that are no longer necessary for the task.

2.1.2 Tool calling

Tool calling enables language models to become agents as it allows them to interact with the outside world and the environment they are deployed to. Tool calling can enable the agents to perform many external tasks such as searching in databases to ground its answers in data through RAG (Retrieval Augmented Generation), execute code, and interact with external APIs to read or modify state in the environment they are deployed to. Tool calling is what has enabled language models to move from generating text to being able to perform tasks autonomously by calling tools and injecting the responses into their context windows. Tool calling is done by injecting structured tool templates into the prompt with instructions on how the tool can be used, what arguments it takes and what data it returns. Figure 2.1 shows such a tool template as it is presented to the model, and Figure 2.2 shows the same tool serialized as the JSON schema the model receives. This allows for the model to reason about and call the tools it deems necessary when solving a task. When the model emits a tool call, a runtime outside the model parses it, executes the corresponding function and appends the returned data back into the context, so the model can condition its next step on what the tool produced. The model itself never executes anything, it only produces the call, which keeps the boundary between the language model and the environment explicit and means that the set of actions available to the agent is fully determined by the tool registry it is given at deployment time.

`get_current_weather`

Description. Returns the current weather for a given location, including temperature and conditions.

Parameters.

`location` *string, required.* City and country, e.g. `"Stockholm, SE"`.

`unit` *string, optional.* One of `"celsius"` or `"fahrenheit"`. Defaults to `"celsius"`.

Returns. A JSON object with fields `temperature` (number), `unit` (string), and `conditions` (string).

Figure 2.1: A tool description as it is presented to the model, a name, a description explaining the tool, two parameters location and unit for the model to choose from, and what the tool returns.

```
{
  "type": "function",
  "function": {
    "name": "get_current_weather",
    "description": "Returns the current weather for a given location.",
    "parameters": {
      "type": "object",
      "properties": {
        "location": {
          "type": "string",
          "description": "City and country, e.g. \"Stockholm, SE\"."
        },
        "unit": {
          "type": "string",
          "enum": ["celsius", "fahrenheit"],
          "description": "Temperature unit. Defaults to celsius."
        }
      },
      "required": ["location"]
    }
  }
}
```

Figure 2.2: The `get_current_weather` tool schema as serialized into the model's prompt.

2.2 Agentic LLM systems

This section introduces the agent architecture used throughout the thesis, defines the trajectory representation that subsequent sections depend on, and presents the benchmarks and environments the methods are evaluated in.

2.2.1 ReAct and tool-augmented reasoning

The ReAct framework introduced by Yao et al. [34]. The ReAct framework implements an autoregressive loop, where in each step the agent first produces a thought and a following tool call, the tool call is then executed in the agent's environment. The environment returns an observation to the agent, which the agent then reasons about before generating a new thought and a new tool call. This process is continued until the agent generates a `finish` action carrying the final answer. Figure 2.3 illustrates this loop.

The action space of a ReAct agent is given by a set $\mathcal{A} = \{\text{tool}_1, \ldots, \text{tool}_K\}$ of named functions, each with a natural language description that explains the tool's usage and what it returns, as well as a parameter schema describing its input arguments.

The sequence of reasoning, tool calls and observations that an agent creates is often referred to as a trajectory:

$$\tau = (s_0, t_0, a_0, o_0, t_1, a_1, o_1, \ldots, t_T, a_T, \hat{y}), \tag{2.3}$$

where s_0 is the initial state (typically a system prompt together with a user query), t_i is a sequence of thought tokens, $a_i \in \mathcal{A}$ is the action chosen at step i, o_i is the observation the environment returns, and $\hat{y}$ is the final answer extracted from the terminal `finish` action.

A trace is not only an output, it is a record of decisions from tool observations. It contains choices of actions, arguments, observations, reasoning, and the final answer. This makes a trace richer than a standard prompt-completion pair, but also harder to score because multiple traces can solve the same task and final-answer correctness does not fully identify process quality.

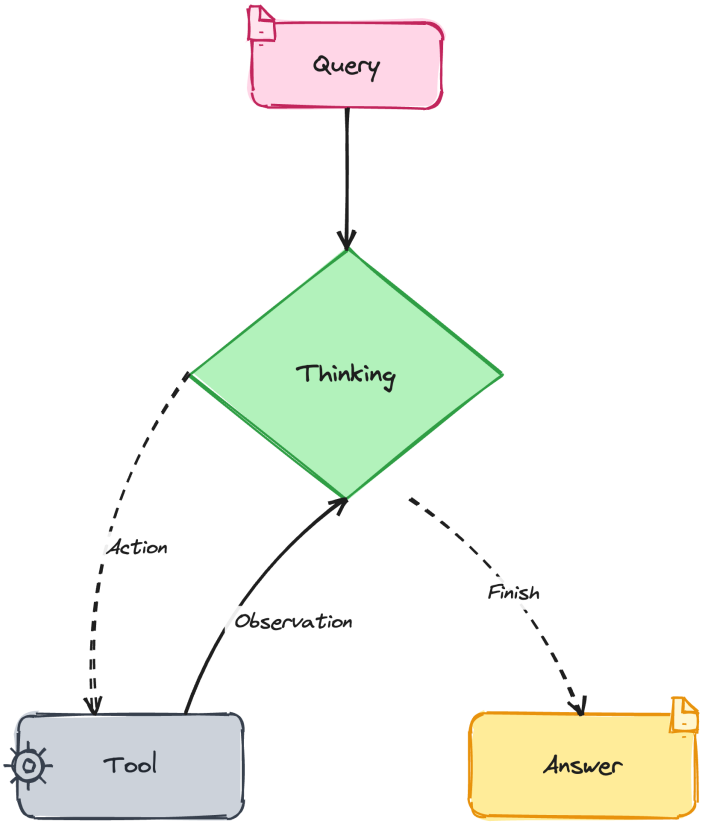


Figure 2.3: The ReAct loop. The model emits a thought and selects an action from the tool registry, the environment returns an observation that is appended to the context for the next iteration. The loop repeats until the model emits the `finish` action carrying the final answer.

2.2.2 Multi-hop tool problems and benchmarks

The term *multi-hop tool problem* denotes a task whose solution requires chaining several tool calls, since no single call returns enough information to answer the query. Unlike math or code generation, where the final answer is unambiguous and a checker can verify it, multi-hop tool problems have two properties that complicate evaluation. First, many different trajectories can be correct, there is usually no single "right" sequence of calls. Second, a trajectory can reach the correct answer while taking a poor route to it, and a sound trajectory can be derailed by a single stale or partial observation. Both properties mean that grading only the final answer can be misleading on these tasks, which is why trajectories have to be evaluated rather than outcomes.

HotPotQA 2.2.2 is a multi-hop question-answering dataset with a small, read-only tool surface and a deterministic answer, it isolates reasoning over evidence. τ^2-bench is a customer-service benchmark with a stateful environment, written domain policies, and a user simulation. StableToolBench 2.2.2 is a large-scale tool-use benchmark built on a cached, deterministic API server.

HotPotQA

HotPotQA [32] is a multi-hop question-answering dataset of roughly 113,000 question-answer pairs built on English Wikipedia. Answering a question requires combining information from more than one document. The dataset was introduced to push question answering beyond simple lookup.

HotPotQA contains two questions types. Bridge questions require identifying an intermediate bridge entity before the answer can be reached, a first fact must be retrieved in order to know what the second lookup should be about. Comparison questions ask for a comparison of a shared property between two entities, and include yes or no answers. Beyond the answer itself, every question is annotated with sentence-level supporting facts, the specific sentences that constitute the evidence needed to derive the answer. The dataset defines two evaluation settings. In the distractor setting the system is given ten paragraphs, the two gold paragraphs plus eith retrieved distractors, and must answer from the fixed context. In the full wikipedia setting no paragraphs are supplied and the system must retrieve its own evidence from the whole of Wikipedia. HotPotQA's standard answer metrics are exact match and a token level F1 score. An example task and trace is shown in figure 4.1 in the method.

τ^2-bench

τ^2-bench [33] is a benchmark for conversation tool-using agents in customer-service settings. Each task places the agent in a domain with a database backed environment, a set of tools that read and modify that environment, a written domain policy that the agent must follow, and a user simulated by a separate language model. The agent has to hold a multi-turn conversation with the user, call tools to carry out the user's request, and follow the domain policies.

A task in τ^2-bench is scored by comparing the final state of the environment database against the task's goal state and by checking that the actions taken are consistent with the domain policy. An example task and trace is shown in figure 4.2 in the method.

StableToolBench and Toolbench

StableToolBench [8] is a tool use benchmark derived from ToolBench [16]. Toolbench is a large scale benchmark for tool learning built from over 16,000 real-world REST APIs collected from a public API hub and spanning dozens of categories, each tasks pairs a natural language instruction with one or more of those APIs. ToolBench groups its instructions by combinatorial difficulty into three level, single tool instructions, intra-category multi tool instruction, and intra collection mutili-tool instructions, G1, G2, and G3. G1 tasks are solvable with a single tool, G2 tasks require several tools drawn from one category, and G3 tasks require several tools from different categories

StableToolBench reports two metrics, both refinements of ToolBench's original pass and win rates, and both restricted to tasks pre-identified as solvable so that genuinely impossible tasks do not add noise. The Solvable Pass Rate (SoPR) is the average task score over solvable tasks, where an automatic LLM evaluator labels each run as solved, unsure or unsolved and score these 1, 0.5 and 0. The Solvable Win Rate (SoWR) is the fraction of solvable tasks on which the agent's solution is preferred to a fixed reference solution by a pairwise LLM judge. Because both metrics are decided by a language model evaluator. An example task and trace is shown in figure 4.3 in the method.

2.3 Continual learning

Continual learning is the problem of training a model on a stream of data so that it acquires new knowledge and capabilities while retaining the ones it already has. This section explains how continual learning is adapted for language models.

2.3.1 Continual learning for agents

For an agentic system, the data stream in continual learning consists of the trajectories the agent produces while solving tasks. Each trajectory contains the choices made by the agent, the observations returned by its tools, and the final answer or action. The trace therefore records not only what the agent answered, but how it arrived there.

This creates both an opportunity and a difficulty. The opportunity is that deployed agents continuously generate experience in the exact configuration in which they are used, the same system prompt, tool registry, and environment. If this experience can be reused, the agent can adapt to new query types, changing tools, shifting user behaviour, or domain-specific failure modes without requiring a manually curated dataset for every change. The difficulty is that the stream is not automatically labeled. Some trajectories contain behaviour that should be reinforced, while others contain mistakes that should not be retained. Learning from the stream therefore requires a curation signal that decides which traces are useful.

2.3.2 Catastrophic forgetting

The central failure mode in continual learning is catastrophic forgetting. It occurs when training a neural network on new data improves performance on the new distribution but sharply reduces performance on data or tasks previously learned. The problem arises because the same parameters are used to represent many behaviours. Gradient updates that make the model fit new examples can therefore interfere with parameter settings that were important for previous capabilities.

For language models, catastrophic forgetting can appear when a model is fine-tuned too narrowly on one task, domain, or style and loses performance on more general language modelling or on other tasks it previously handled well. In an agentic setting, the same risk applies to traces, fine-tuning an agent on newly collected trajectories may improve behaviour on the current domain, but it can also degrade earlier tool-use skills, answer formats, or reasoning patterns. This is why continual learning methods must consider not only how quickly the agent adapts, but also how much useful prior behaviour is retained.

2.3.3 Parametric and non-parametric adaptation

A language model can be adapted to new data in two different ways. Parametric, which updates the model's weights through re-training. The model changes and the change persists unless reversed by further training. Non-parametric adaptation leaves the weights untouched and instead changes the input the model sees at inference time, typically by retrieving relevant text from an external store and adding it to the prompt. The two strategies can also be combined. A system can fine-tune on a curated subset of its data and keep a separate non-parametric memory.

2.4 Evaluating agentic traces

A method that learns from agent traces needs a way to score a trajectory when no reference answer is available. This section covers using a language model as the judge, the biases such a judge can show, the difference between scoring the outcome and the process, the blind setting the judge operates in, and the metrics used to assess the judge and to grade the agent's own answers.

2.4.1 LLM-as-a-judge

A common way to evaluate outputs is to use a large language model as the judge. Zheng et al.[43] showed this works on open-ended tasks, GPT-4's verdicts agreed with human raters about as often as the human raters agreed with one another. This is the LLM-as-a-judge

paradigm. The judge here is a larger, closed-source model than the open agent it scores. LLM-as-a-judge is a general judging paradigm that doesn't require the judge to to be trained on the specific material it judges. With a prompt and a target explanation of how the LLM should judge it can be used in many different settings.

2.4.2 Biases in LLM judges

An LLM judge is not free of biases. Zheng et al.[43] identified three biases it can show. Position bias: when two responses are compared, the order in which they are shown can change which one is preferred. Verbosity bias: a longer response is preferred over a shorter one regardless of content. Self-enhancement bias: a judge prefers responses from its own model family. These are properties of the judge model, not of the task, so any use of an LLM judge has to allow for them.

2.4.3 Outcome- and process-based signals

A trace can primarily be judged on two things. The outcome which is the agent's final answer or where the traces ends or on the process that the trajectory produced. When a golden answer is available, scoring the outcome is often the choice, and is used in domains such as mathematics or code, where the answer is unambiguous and a checker can verify it. On multi-hop tool problems, the outcome can mislead, an agent can reach the correct answer through irrelevant tool calls, and an agent with sound reasoning can be defeated by one stale or missconfigured tool call, therefore the final answer alone does not say whether the run was competent. The reward modelling literature found the same gap in mathematical reasoning where a reward model that scores reasoning steps is more reliable than one that scores only the final answer [23, 12]. The distinction between an outcome signal and a process signal is illustrated in Figure 2.4.

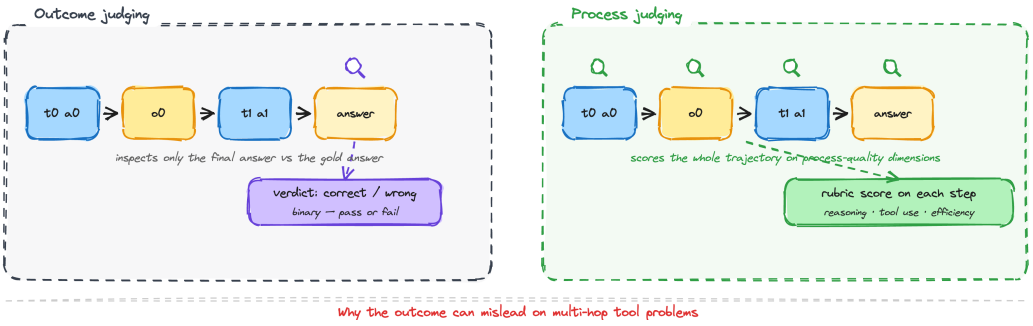


Figure 2.4: Output and process level analysis

2.4.4 Rubric-based judges

A rubric-based judge scores against a fixed set of named dimensions, each with a written description and a numerical scale, rather than giving one overall preference. The judge produces a vector of dimension scores rather than a single number, and these can be averaged into one quality figure when a scalar is needed. The dimensions of the rubric can be general or adapted to the domain the judge is judging. Table 2.1 shows an example rubric from SummEval [7].

Dimension	Scale	Definition
Coherence	1–5	"the collective quality of all sentences"-the summary should be well-structured and well-organised, building from sentence to a coherent body of information.
Consistency	1–5	"the factual alignment between the summary and the summarized source"-contains only statements entailed by the source; hallucinated facts are penalised.
Fluency	1–3	"the quality of the summary in terms of grammar, spelling, punctuation, word choice, and sentence structure", with explicit Poor/Fair/Good level descriptors.
Relevance	1–5	"selection of important content from the source"-redundancies and excess information are penalised.

Table 2.1: The SummEval rubric.

2.5 Training from traces

This section introduces the training procedures the thesis use: supervised fine-tuning with parameter-efficient adaptation, rejection-sampling fine-tuning, direct preference optimisation.

2.5.1 Supervised fine-tuning and LoRA

Supervised fine-tuning (SFT) adapts a pre-trained language model to a target distribution by maximising the conditional likelihood of target sequences,

$$\mathcal{L}_{\mathrm{SFT}}(\theta) = -\,\mathbb{E}_{(x,y)\sim\mathcal{D}}\big[\log \pi_\theta(y \mid x)\big], \tag{2.4}$$

where π_θ is the model and $\mathcal{D}$ a dataset of input–output pairs. For agent traces, x is the system prompt together with the tool registry and y is the trajectory, the loss is applied only to model-generated tokens- which are thoughts and actions and masked over environment-returned observations.

Full-parameter SFT of a multi-billion-parameter model is expensive in both memory and compute. LoRA, introduced by Hu et al., is the standard parameter-efficient alternative [10]. It freezes the pre-trained weights $W_0 \in \mathbb{R}^{d\times k}$ of selected linear layers and injects a trainable low-rank decomposition $\Delta W = BA$, with $B \in \mathbb{R}^{d\times r}$, $A \in \mathbb{R}^{r\times k}$, and $r \ll \min(d,k)$, so the forward pass becomes

$$h = W_0 x + \frac{\alpha}{r} BA\, x, \tag{2.5}$$

where α is a scaling hyperparameter. Hu et al. report that LoRA matches full fine-tuning quality on a wide range of tasks while reducing the number of trainable parameters by orders of magnitude. QLoRA, introduced by Dettmers et al., extends it by quantising the frozen base weights to 4-bit precision and back-propagating through the dequantised weights, with little loss in fine-tuning quality [6].

2.5.2 Rejection-sampling fine-tuning

Rejection-sampling fine-tuning (RFT) is a self-training procedure first articulated as a technique by Yuan et al. [37] and used at scale in the post-training of LLaMA 2 [22]. Given a base policy, a set of prompts, a quality criterion Q, and a threshold, it samples several candidate trajectories per prompt, retains those whose quality score exceeds the threshold, and fine-tunes the model on the retained set. The procedure can be iterated, each round producing a new sampling policy.

Algorithm 1 Rejection-sampling fine-tuning [37]

Require: base policy π_0; prompts $\mathcal{X}$; quality criterion Q; threshold η; samples per prompt N; rounds K
Ensure: fine-tuned policy π_K
0: $\pi \leftarrow \pi_0$ {sampling policy}
0: **for** $k = 1, \ldots, K$ **do**
0: sample $\tau_{i,j} \sim \pi(\cdot \mid x_i)$ for each $x_i \in \mathcal{X}$, $j = 1, \ldots, N$ {rollouts}
0: $\mathcal{D}_k \leftarrow \{ (x_i, \tau_{i,j}) : Q(\tau_{i,j}) \geqslant \eta \}$ {keep high-quality trajectories}
0: $\pi_k \leftarrow \arg\min_{\pi'} \mathcal{L}_{\mathrm{SFT}}(\pi'; \mathcal{D}_k)$, initialized from π_0 {fine-tune base policy}
0: $\pi \leftarrow \pi_k$ {new sampling policy for next round}
0: **end for**
0: **return** π_K =0

The original RFT used a gold-answer match as the quality criterion Q. However in theory any quality criterion could be used for ranking the outputs. Algorithm 1 shows the full rejection-sampling fine-tuning procedure.

2.5.3 Direct preference optimisation

Direct preference optimisation (DPO) optimises a policy directly against a dataset of pairwise preferences, bypassing the explicit reward-model training and reinforcement-learning stages of RLHF [17]. Given a dataset $\mathcal{D} = \{(x, y_w, y_l)\}$ of prompts x with a chosen response y_w and a rejected response y_l, the DPO loss is

$$\mathcal{L}_{\mathrm{DPO}}(\theta) = -\,\mathbb{E}_{(x,y_w,y_l)\sim\mathcal{D}}\left[\log \sigma\left(\beta \log \frac{\pi_\theta(y_w \mid x)}{\pi_{\mathrm{ref}}(y_w \mid x)} - \beta \log \frac{\pi_\theta(y_l \mid x)}{\pi_{\mathrm{ref}}(y_l \mid x)}\right)\right], \tag{2.6}$$

where π_{ref} is a frozen reference policy (typically the SFT initialisation) and β controls the strength of the implicit KL constraint between the trained policy and the reference.

3 Related work

This chapter reviews the prior work the thesis builds on, specifically it reviews the three directions of learning that the thesis builds upon.

3.1 Filtered fine-tuning from agent traces

This section covers prior work on rejection-based filtering with supervised fine-tuning (SFT).

To the best of our knowledge, the first work that created a generation, filtering and fine-tuning approach for improving a language model's performance on a specific task was introduced by Zelikman et al. in STaR [38]. Zelikman et al. created the filter-then-SFT loop, where for each iteration they sampled rationales that corresponded with the gold label and used them to fine-tune the model. On the GSM8K grade-school math benchmark, STaR improved performance from approximately 3% to 10.7% on a GPT-J 6B backbone, substantially exceeding the few-shot chain-of-thought baseline.

The rejection-based filtering approach was further developed by Yuan et al. [37], who showed that mathematical reasoning in language models could be improved by training on filtered correct examples. Yuan et al. generated reasoning traces and used a solver to label each trace as correct or incorrect. The correct traces were then used to further train the language model. This method of generating correct samples for training was subsequently used at wide scale in the post-training of Llama-2 [22].

Singh et al. continued this line in ReST$^{\mathrm{EM}}$ [20], developing a method that could replace expensive human-annotated traces with an EM-like loop: first sampling traces, judging them with a binary signal, using the filtered traces as training data, and then repeating the process for several rounds. Similarly to Zelikman and Yuan, they showed performance gains on the MATH benchmark, but also on the APPS coding benchmark, demonstrating that the recipe generalises across domains where a binary verifier exists.

Closest to our work is FireAct by Chen et al. [4], which applies rejection-based filtering to agent trajectories on HotPotQA using exact match against the gold answer as the filter. They showed that LoRA fine-tuning on GPT-4 ReAct traces enabled a Llama-2-7B model to improve from 14.8% to 26.2% exact match on HotPotQA. FireAct also distils GPT-4's behaviour into the smaller model, since the trajectories used for fine-tuning are generated by GPT-4 rather than by the model being trained.

Zeng et al. generalise this approach in AgentTuning [39], building the AgentInstruct dataset across six environments (AlfWorld, WebShop, Mind2Web, KG, OS, Database) again GPT-4 is used to generate the trajectories, the filter that decides which traces enter the training set.

The common theme is that the quality criterion is a binary signal grounded in a verifiable task, gold-answer match for math and multi-hop question answering, or unit-test execution for code. Settings in which no such verifier exists, where no single correct answer can be checked against, remain an open research direction.

3.2 Memory-based curation and context engineering

This section relates to memory-based methods for continual learning and the in-context learning that enables them to work. Memory-based methods are primarily based on in-context learning and few-shot prompting that can be retrieved during inference from a memory layer. A memory-based problem can in general be described as follows: first, memory needs to be generated, either through analysing a full trace or having the agent itself decide when something is to be saved. Then, memories have to be compressed into some format that can be used for in-context learning. Lastly, the correct summaries have to be retrieved at the correct time. Reflexion [19] laid the groundwork for memory-based agents by introducing the idea of converting a failed trajectory and its outcome into a natural-language reflection that gets saved to a small episodic buffer and reused when the same task is attempted again. Reflexion lifted the performance on HumanEval from 80% to 91% with GPT-4 as the actor model. Voyager [26] and Generative Agents [15] continued on this, with Voyager creating a skill library in Minecraft, saving skills and growing unbounded over time, and Generative Agents storing facts selected at recall time by a weighted combination of recency, importance, and embedding.

ExpeL by Zhao et al.[42] set the scene for the kind of memory-based layer that we are working with. To reduce the need for labelled task-specific data, they curated a memory bank with a way to write to it via operations such as add, edit, upvote, and downvote to manage the size and scale of the memory bank. ExpeL increased GPT-3.5-turbo's performance on HotPotQA from 28% to 39%, and from 40% to 59% on ALFWorld.

The work of Zhao et al. has been continuously adapted over the past years, with papers such as ACE [40] that improve the structure of how things are saved. Two further papers, GEPA [1] and ReasoningBank [14], improve how the memory bank evolves over time. GEPA introduces an evaluation loop that only admits new memories if they improve performance on a held-out set. In ReasoningBank, memories are created from both correct and failed traces.

Several papers have worked on the memory layer for storing facts and preferences over time, with Mem0 [5] being a notable example. The difference with these memory systems is that they target user facts and preferences for chatbot-style implementations of agents, rather than storing a global task-execution memory.

What's consistent throughout the task-style memory papers is that they rely on an outcome label or task-success for storing and curating memory, a reward that has to be curated per task, not judged at inference. Only ReasoningBank uses the model's own self-judgement for pass or failure. This can be difficult in an online setup with a model's tools and configuration, once again showing the need for a general task learning signal.

3.3 Replay based methods

This section covers works related to replay based methods within machine learning and how they've been adapted to agentic trajectories. To the best of our knowledge the idea of replaying a trajectory from a failure point comes from Hindsight Experience Replay (HER) [2]. The

problem that HER tries to solve is that, if a trajectory failed up until a point and the reward is based on the full trajectory, the model would not get a reward for what it had done until that point. To solve this, for traces that reached a failure state, they relabel the goal with a state the agent actually reached, in order to learn from that trace and show that a failed trace can still carry a learnable training signal.

The idea of replay-based learning for agents first came up in Trial and Error: Exploration-Based Trajectory Optimization (ETO) [21], where, through an exploration phase, many different traces are generated on the same query with rewards returned by the environment. The trace–reward pairs are then used to create preference pairs for DPO from one successful and one failed trace, in order to teach the model the correct trajectory. Iterative Step-level Process Refinement (IPR) [29] continues on ETO's track by using Monte Carlo rollouts to sample new actions and new trajectories from expert trajectories, rewarding them with signals from the environment, and showing that the agent's actions can be improved at many different steps of a trace. A similar approach is used in Agent-R [36], where self-critique is used to find the first erroneous step. The theme of the previous paper's have been to sample by generating new tool calls from the agent or through reflection, another method that have been used to create alternative traces are through thinking injection, Wu et al [28] showed that it increased perfomance.

3.4 Positioning

The three tracks reviewed above share one bottleneck, they all depend on a verifiable success signal. Whether through exact match, unit tests, or a verifier, in domains where there is no clear answer these methods do not work. Today, when an agent is deployed in an online setting, vast amounts of data are collected about how the agent is performing, and none of those successes and failures are captured or used. The gap is that in these settings there is no reliable way to improve the agent from the successes and failures it stumbles upon.

Closing this gap would change what deployed traffic is worth. Today every trace is discarded the moment the session ends, and as noted in the introduction the same mistakes recur because nothing in the deployment loop learns from them. A reliable process-level signal turns that discarded traffic into a self-improvement loop: an agent could adapt to its own domain continually, from the successes and failures it already produces, without waiting on human-annotated preference pairs or a hand-built verifier, neither of which exists for most real deployments. For the agent, this means competence that compounds with use rather than resetting at every session, the longer it runs, the better it fits the tools and tasks it is actually given. For the people who deploy and rely on these agents, it means more dependable systems in exactly the open-ended domains where today's label- and verifier-dependent methods cannot reach, and adaptation that scales with usage instead of with annotation budget.

4 Method

This chapter describes the methods and experimental setups used in the thesis. Firstly, an overview of the thesis is presented, along with the benchmarks, how the trace data is generated from them, and the agent setup used. Afterwards, the implementation of the rubric judge follows, along with how it is used for filtering and selecting traces. After covering the experimental setup, the method for each track is discussed: filtered supervised fine-tuning, memory rule curation, and finally replay-based.

4.1 Overview

The thesis is organised around an agent, a rubric-based LLM judge, and three downstream trace-reuse pipelines. The agent is a Qwen3.5-9B language model wrapped in a ReAct loop and served via vLLM. The judge is a separate, fixed GPT-5.4 instance that scores the traces produced by the agent against a rubric. Each downstream pipeline consumes the judged traces in a structurally different way. The first filters traces and fine-tunes the agent on the retained ones, the second extracts in-context rules from contrastive trace pairs and injects them at inference time without changing the agent's weights, and the third replays failed traces and constructs preference based pairs.

4.2 Benchmarks

We evaluate on three benchmarks, HotPotQA and τ^2-bench which are used and were chosen for their structural difference: HotPotQA isolates multi-hop reasoning over a small read-only tool surface, while τ^2-bench, in its retail, airline telecom domains, exposes the agent to a state-mutating environment with a simulated user and a larger set of the tools. The third benchmark, StableToolBench, is used by Track 3.

4.2.1 HotPotQA

HotPotQA [32] is a multi-hop question-answering dataset of 113K Wikipedia-based question-answer pairs. Each question requires composing information from at least two supporting passages, and the dataset includes both bridge and comparison questions. The agent interacts with HotPotQA through two read-only tools. `search` accepts an entity name and returns

the first paragraph of the corresponding Wikipedia article when an exact match exists, or a list of similar entity names otherwise. `lookup` retrieves a specific sentence from the most recently searched article by keyword. The agent terminates a trajectory by emitting the `finish` action with its candidate answer. Queries and gold answers are loaded from the standard JSONL release. We extend the HotPotQA evaluation, in the standard benchmark final-answer correctness is measured by Exact Match (EM) against the gold string and by Token F1 between the predicted and gold answer. As EM rejects answers that are correct but differ in surface form, we include a semantic equivalent judgement of the answer against the gold reference.

Figure 4.1 shows a real two-hop HotPotQA trace, where the agent must bridge two Wikipedia pages before it can answer.

```
Question: Into Dust is a song by the alternative rock band
          formed in which city?

search("Into Dust")
-> "Into Dust" is a song by American duo Mazzy Star ...

search("Mazzy Star")
-> Mazzy Star is an American alternative rock band formed
   in 1988 in Santa Monica, California ...

finish("Santa Monica, California")
```

Figure 4.1: A HotPotQA trace from the baseline agent. The agent searches the song, finds the band, then searches the band to recover the city.

4.2.2 τ^2-bench

τ^2-bench [33] is a customer-service benchmark in which both the agent and a simulated user can take turn-based actions in a shared, state-mutating environment. We use the retail, telecom and airline domains. Retail provides 114 tasks involving order tracking, returns, exchanges, and refunds. Telecom adds a dual-control structure where the agent must elicit information from the user before taking irreversible tool actions, and in both the simulated user is itself a language model with its own scripted goals.

We follow the official τ^2-bench evaluation protocol, with one adjustment: we simulate users with GPT-4.1-mini to keep evaluation cost manageable. The agent is given the standard budget of 100 steps per task. Final-task success is measured by the official τ^2 evaluator, which compares the final database state against gold post-condition. For collecting traces only the rubrical blind judge is used.

Figure 4.2 shows a condensed retail task. The agent first authenticates the user, then looks up the order, and only then calls a state-mutating tool, reflecting the policy constraint that irreversible actions follow verification.

```
User: I received an office chair, but parts arrived broken.
      I want to exchange it. My name is Mei Davis, zip 80217.

find_user_id_by_name_zip(first_name="Mei",
    last_name="Davis", zip="80217")
-> "mei_davis_8935"

get_user_details(user_id="mei_davis_8935")
-> {name: "Mei Davis", orders: ["#W2890441", ...], ...}

get_order_details(order_id="#W2890441")
-> {status: "delivered",
    items: [{item_id: "8069050545", name: "Office Chair"}]}

User: Give me the gray leather chair instead, item 1071497737.

exchange_delivered_order_items(order_id="#W2890441",
    item_ids=["8069050545"], new_item_ids=["1071497737"],
    payment_method_id="credit_card_1061405")
-> {status: "exchange requested"}

Agent: Your exchange has been successfully processed.
```

Figure 4.2: A condensed τ^2-bench retail trace from the baseline agent. The user is authenticated by name and zip, the order is retrieved, and the delivered chair is exchanged for another variant.

4.2.3 StableToolBench

StableToolBench [9] is used by Track 3 and is itself partitioned. The G1 and G2 subsets are used to collect the trajectories from which the replay-DPO preference pairs are constructed; the G3 subset is held back as a held-out evaluation set, on which the tool responses are deterministically cached so that the same query trace is reproducible across runs. The benchmark partitions queries into six subsets by combinatorial difficulty: G1 (single-tool, single-instruction), G2 (single- or multi-tool, multi-instruction), and G3 (multi-tool, with cross-tool reasoning). The first two groups are subdivided by query class: G1 instruction, G1 category, G1 tool, G2 category, G2 instruction. Replay-based traces are collected from the five G1 and G2 subsets, and the G3 instruction split is excluded from the released preference dataset.

Figure 4.3 shows a G1 single-tool task, where one API call from the query's tool set is sufficient to satisfy the instruction.

```
Instruction: Get the current exchange rate from USD to EUR.

get_exchange_rate(base="USD", target="EUR")
-> {rate: 0.92, timestamp: "2026-01-15T10:00:00Z"}

finish("The current USD to EUR exchange rate is 0.92.")
```

Figure 4.3: A StableToolBench G1 trace. A single tool call from the query's RapidAPI tool set answers the instruction.

For all benchmarks, traces are collected by running the agent on a set of training queries that differ per benchmark. The generation configuration is described in Chapter 4.3. The agent's reasoning mode is enabled so that the model thinks before calling each tool. Each trace is stored as an ordered sequence of blocks $\tau = (m_0, m_1, \ldots, m_T)$, where each block is

either a thought, a tool call with its arguments and observed output, or the final answer. In the τ^2-bench setting the traces also contain the simulated user's responses. Figure 4.4 shows an example of such a trace and how its blocks follow one another.

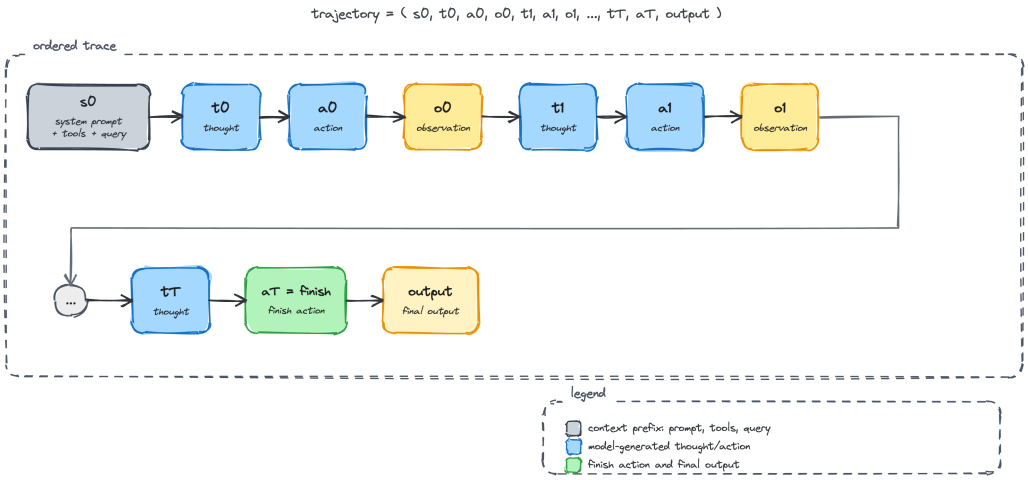


Figure 4.4: An example agent trace. The agent alternates between a thought and the tool call it leads to, the environment returns an observation that is appended to the context, and this repeats until a final block carries the agent's answer. The full ordered sequence of blocks is what we refer to as a trace.

4.3 Agents

The langauge model we use is Qwen3.5-9B, from the Qwen3 series [30]. The choice is driven by three factors. First, the model is open-source and deployable on a single A100, which is necessary for the parameter-efficient fine-tuning experiments in Chapter 4.5 and Chapter 4.7 and for the efficent inference at the 9B scale. Second, the model has a thinking-mode variant that emits an explicit chain of internal reasoning. Reasoning models tend to perform better than their non-reasoning counterparts, and the reasoning trace also gives the judge a signal to assess when judging. Lastly the model is one of the best open source model's for the 9B parameter range so that the experiments are evaluated on the latest language models.

The agent follows the ReAct framework as defined in Chapter 2.2.1. The tool registry is benchmark-specific. HotPotQA exposes `search` and `lookup` while the τ^2-bench exposes each domain's tools. Tool calls are dispatched through the official Model Context Protocol (MCP) interface of each benchmark, and tool outputs are parsed from their internal representation into plain text before being appended to the agent's context.

The agent is served with vLLM in OpenAI-compatible mode with native function calling enabled. The model parameters are fixed across all experiments at the settings recommended for Qwen3.5-9B with reasoning enabled are used: temperature 0.6, top-p 0.95, a maximum response length of 4,096 tokens. The same parameters are used during trace collection and during final evaluation. The cross-model experiment in Chapter 4.6 additionally runs the memory pipeline with GPT-5.4-mini as the agent in place of Qwen3.5-9B, that setup is described together with Track 2.

4.4 Judge implementation

The judge is used throughout the thesis to evaluate the quality of an agent's traces. The judge is a blind, rubric-based LLM judge, it sees only the agent's trajectory and the original task, never the gold answer, and scores the trajectory against rubrics designed for each benchmark. The judge produces the quality signal that drives all three tracks.

4.4.1 Rubric

The rubric are benchmark-specific. For HotPotQA it has four dimensions, each rated on a six-point scale from 0 to 5:

- **Evidence Retrieval** - whether the agent retrieved the documents needed to answer the question.
- **Evidence–Answer** - whether the answer is supported by the evidence retrieved.
- **Reasoning Coherence** — whether the chain of thoughts and tool calls are consistent.
- **Answer Specificity** - whether the final answer addresses the question precisely rather than paraphrasing or dodging.

The process-quality score is the unweighted mean of the dimension scores. On HotPotQA the judge additionally returns a binary correct/incorrect verdict, and every judgement carries a self-reported confidence in the range [0, 100]. The process-quality score is the signal the downstream tracks consumes.

$$q(x) = \frac{1}{|\mathcal{D}_b|} \sum_{d \in \mathcal{D}_b} s_d(x), \qquad s_d(x) \in \{0, 1, 2, 3, 4, 5\}. \tag{4.1}$$

where

- x is the judged *trace*, i.e. the (task, trajectory) pair the blind judge is shown;
- b is the benchmark and D_b the set of rubric dimensions defined for it (the four HotPotQA dimensions above, or the three τ^2-bench dimensions of Table 4.1);
- $|D_b|$ is the number of those dimensions (4 for HotPotQA, 3 for τ^2-bench);
- $s_d(x) \in \{0, \ldots, 5\}$ is the integer score the judge assigns trace x on dimension d, on the six-point scale.

Table 4.1: Rubric dimensions $d \in D_b$ used by the blind judge. Each dimension contributes exactly one integer score $s_d(x) \in \{0, \ldots, 5\}$ to the process-quality score $q(x)$ of Eq. (4.1): HotPotQA uses $|D_b| = 4$ dimensions and τ^2-bench uses $|D_b| = 3$.

Benchmark	Dimension d	$s_d(x)$	Description
HotPotQA	Evidence Retrieval	s_{ER}	Did the agent retrieve the documents needed to answer the question?
	Evidence–Answer Alignment	s_{EAA}	Is the final answer supported by the evidence retrieved?
	Reasoning Coherence	s_{RC}	Is the chain of thoughts and tool calls internally consistent?
	Answer Specificity	s_{AS}	Does the final answer address the question precisely?
τ^2-bench	Action Quality	s_{AQ}	Are the right tools called with the right parameters?
	Execution Efficiency	s_{EE}	Is the trajectory free of redundant or wasted tool calls?
	Task Completion	s_{TC}	Are all sub-issues addressed and verified before closing?

4.4.2 Judge model

All trace judging in this thesis is done with GPT-5.4-mini, with reasoning effort set to medium. Two judges share this model and differ only in their prompt. The rubric judge scores the trajectory on the rubric dimensions of Chapter 4.4.1, and the binary judge returns a single pass or fail verdict without the dimension scores. Using one model for both keeps the comparison between the rubric and the binary verdict free of any effect from a change of judge. As commercial models are updated at regular intervals, all judging was carried out during the first half of 2026, and the reported results correspond to the GPT-5.4-mini version available in that period.

GPT-5.4-mini was chosen for two reasons. First, the volume of judging in this thesis is large, as every collected and evaluated trace is scored across all tracks, so a frontier-scale judge on every trace would have been prohibitively expensive and a smaller commercial model was required. Other frontier judges were not compared, as the cost of judging at this scale and the API access available to the project made a single consistent model the practical choice.

A separate semantic judge, also GPT-5.4-mini, is used only on HotPotQA, where it serves as a correctness metric rather than as one of the judges under study. It decides whether the agent's final answer is semantically equivalent to the gold answer, which is needed because Exact Match rejects answers that are correct but differ in surface form, making it unreliable on its own.

4.4.3 Validating the judge

The judge is validated against the ground truth on all benchmarks it is applied to, so that the quality signal it produces can be trusted in each setting.

On HotPotQA the judge's blind verdict is tested against whether a trace actually led to a correct answer. The validation uses a set of 493 HotPotQA traces where each is scored by the judge, and the verdict is compared against answer correctness, determined by the semantic judge. The rubric judge is compared against a simpler baseline that produces a single pass/fail verdict without scoring the rubric dimensions, which tests whether the rubric adds value over an unstructured verdict. For each judge configuration, accuracy, precision, and recall against the correctness labels are reported. Precision, which is the share of accepted

traces that are in fact correct, matters most, because a false positive places an incorrect trace into the training data.

On the retail domain of τ^2-bench the judge is validated differently, because the benchmark supplies its own ground truth with the official evaluator that returns a task-success outcome for every task. The validation uses only the baseline agent's evaluation traces, and it reports the process-quality gradient that shows how the official task-success rate varies across the range of the process-quality score. A meaningful signal should produce a rising gradient, traces the judge scores higher should succeed more often.

The judge is validated a third time on ToolHop [35], a benchmark of 995 multi-hop tool use tasks. ToolHop is used by no track of the thesis as each query of the benchmark consists of its own tool definitions. Toolhop is included only as an additional validation domain for the judge.

4.5 Track 1 - Filtered supervised fine-tuning

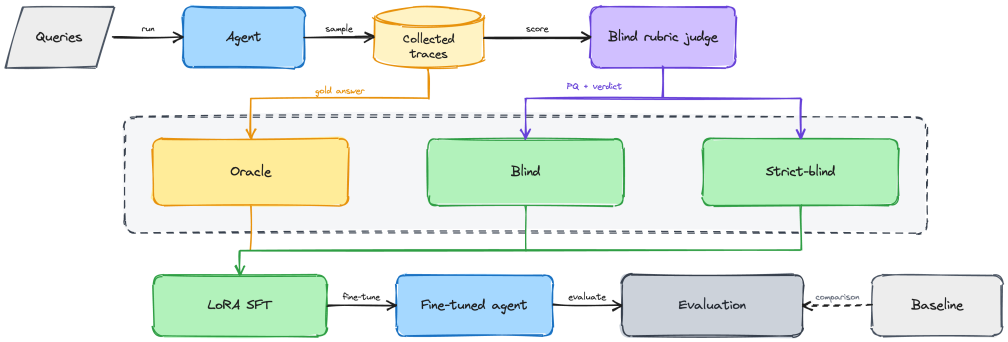


Figure 4.5: Track 1 fine-tunes the agent on its own traces, curated with no golden labels. The agent generates traces on training queries, a blind rubric judge scores each trace's process quality (PQ) without seeing the golden answer, a per-condition filter keeps a subset; and the agent is LoRA fine-tuned on the retained set, then evaluated on a held-out split against the baseline. Three filters of increasing strictness are compared, Oracle (correct against the golden answer, an upper bound that uses labels), Blind (the blind judge's correct verdict), and Strict-blind (blind-correct and PQ >= 4.5), alongside the no-fine-tuning Baseline. One iteration of the rejection-sampling fine-tuning loop with the rubric judge as the quality criterion.

Track 1 uses the judged traces as training data for supervised-finetuning (Figure 4.5). The judge scores every collected trace, the process-quality score selects the traces to keep, and the agent is fine-tuned on the retained traces.

Track 1 instantiates the rejection-sampling fine-tuning loop of Algorithm 1 with the rubric judge as the quality criterion, but only one iteration of the loop is executed.

4.5.1 Curation conditions

Track 1 compares fine-tuning the agent on different partitions of the collected traces. They differ only in the filter that decides which traces are kept. The baseline applies no fine-tuning and is the comparison point. First, the oracle filter keeps the traces marked correct by the benchmark's gold-standard outcome, which is the meta-judge for HotPotQA and the official τ^2 evaluator for τ^2-bench. The blind filter is what a non rubric based judge scored as correct,

and the strict-blind filter adds a continuous process-quality threshold on top of the blind judge's verdict.

Table 4.2: Curation conditions for Track 1. Each condition fine-tunes the agent on the traces selected by its filter. The baseline applies no fine-tuning.

Condition	Filter
Baseline	no fine-tuning
Oracle filter	traces marked correct against the gold answer
Blind filter	traces the blind judge marks correct
Strict-blind filter	traces the blind judge marks correct, with process-quality $\geqslant 4.5$

The full set of conditions was evaluated on HotPotQA, where the agent has substantial headroom. On τ^2-bench the comparison is narrower. Retail is evaluated with the baseline, oracle, and strict-blind conditions. Telecom is evaluated with the baseline and blind conditions only, as its baseline pass rate is already near the benchmark ceiling, so the filters have no headroom to be told apart, and the remaining conditions were not run.

4.5.2 Trajectory formatting

Each retained trace is converted into a multi-turn chat conversation. The conversation begins with a system message containing the benchmark-specific instructions and tool-registry description, followed by a user message with the query. Each tool call in the trace becomes an agent message with the structured function call, followed by a tool message containing the parsed output. The conversation ends with a final agent message carrying the agent's answer. The full conversation is then serialised through the model's chat template into a single token sequence. The SFT loss is masked for the user, system, and tool messages, and applied only to agent tokens.

4.5.3 Training configuration

Fine-tuning runs on a single NVIDIA A100 with 80 GB of memory, which fits the 9B base model in bf16 together with the LoRA adapters and gradient checkpointing. The training stack is built on the Unsloth library [24], for memory-efficient transformer kernels, and the TRL library, which provides the supervised fine-tuning trainer.

LoRA [10] is applied to the attention and feed-forward projection matrices of every layer, with rank $r = 16$, scaling factor $\alpha = 16$, and no adapter dropout. The optimiser is 8-bit AdamW with a learning rate of 2×10^{-4}, a cosine schedule with 10 warmup steps, and weight decay 0.01. The effective batch size is 8, from a per-device batch size of 1 with gradient accumulation over 8 steps. Training runs for three epochs, with a maximum sequence length long enough to contain the full multi-turn trajectories without truncation.

The configuration is held constant across all curation conditions and all benchmarks.

4.5.4 Evaluation

Each fine-tuned model is evaluated against the baseline agent. For HotPotQA, the held-out set are 313 questions drawn from the validation split. Each model answers every question once. For τ^2-bench, the fine-tuned models are evaluated on the official task splits, 114 retail and 114 telecom tasks, under the official τ^2 evaluator described in Chapter 4.2.2.

4.6 Track 2 - Memory rule curation

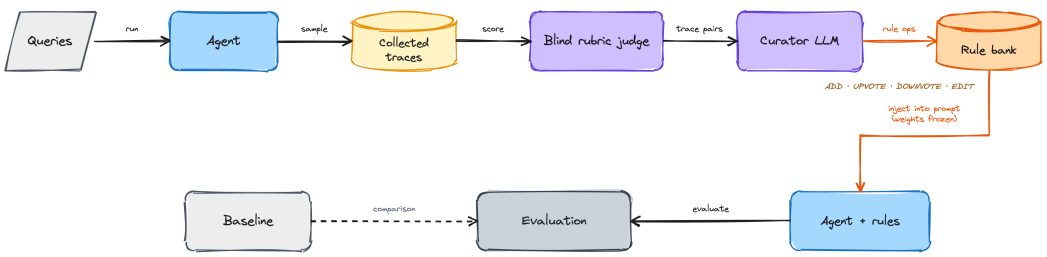


Figure 4.6: Track 2 - memory rule curation. From the agent's judged traces, a curator LLM distils a capped bank of "when [trigger], [action]" rules, scoring traces with a blind rubric judge, pairing comparable queries low- versus high-PQ, and editing the bank with atomic ADD/UPVOTE/DOWNVOTE/EDIT operations. At inference the full bank is injected into the prompt (no retrieval needed) and the frozen agent is tested against a no-rules baseline, requiring no fine-tuning, it is the only track also run on a foundation model (GPT-5.4-mini) rather than Qwen3.5-9B.

Instead of fine-tuning the model Track 2 curates a bank of natural language rules from the agent's own judged traces and injects them into the agent's prompt at inference. The judge scores the collected traces, a curator LLM extracts rules from contrastive pairs of those traces, and the rule bank is then consulted by the agent on every task. This section describes the curation algorithm, which is a single pass over the collected traces.

4.6.1 Rule extraction

Rules are extracted from contrastive pairs of trajectories. The judged traces are first grouped by task signature, the task type together with the issues it raises, so that only traces of comparable queries are contrasted. Within a group, a low quality trace is paired with a high quality one whenever their process quality scores differ enough for the contrast to be informative, and the pairs are ordered so that the biggest failures come first. The pairs are then processed in small batches, in a single pass over the collected traces.

For each batch, a curator LLM is shown the batch, the current rule bank, and a list of open "problems" which are recurring failure patterns not yet addressed by any rule. It returns a set of atomic operations on the rule bank: `ADD` a new rule, `UPVOTE` or `DOWNVOTE` an existing rule, `EDIT` a rule into a refined version, or register a new `PROBLEM`. Every operation is logged with a pointer to the contrastive pair that produced it, so each rule remains traceable to the traces that justify it. The curator's operations accumulate across the batches of the pass into the final rule bank. This contrastive extraction design follows the works described in the relation work Section 3.2.

4.6.2 Rule schema

Each rule is a record built around a `trigger`, a "when ..." condition, and an `action`, naming the concrete tool or step to take. A rule also stores the contrastive traces it was extracted from, a `derived_from` tag recording whether it came from a failure, a success, or a contrast, and a `status` marking its position in a lifecycle: `PROPOSED` on creation, `ACTIVE` once it has accumulated support, `VALIDATED` once well supported, `SUPERSEDED` when an edit replaces it, and `REMOVED` when it is pruned.

A rule's track record is kept in two counters, `helpful_count` and `harmful_count`, updated by the curator's upvote and downvote operations and summarised by an `importance` score. An `EDIT` does not overwrite a rule: the old version is marked `SUPERSEDED` and a new child version is created, so the bank retains the full lineage of every rule. An example record from the τ^2-bench retail rule bank:

Listing 4.1: A rule from the τ^2-bench retail rule bank.

```
{
  "id":              "R0-02",
  "trigger":         "changing items in a pending order",
  "action":          "call modify_pending_order_items with item_ids and
     matching new_item_ids",
  "status":          "VALIDATED",
  "derived_from":    "contrast",
  "importance":      15,
  "helpful_count":   13,
  "harmful_count":   1,
  "evidence_traces": ["trace_2024_pass", "trace_3142_fail"]
}
```

4.6.3 The curation

The rule curation is done in a single pass, which proceeds in four steps:

1. Generation. The agent is either run on a sample of tasks or taken from a store of traces.
2. Judging. The rubrical judge scores every trajectory according to a curated rubric.
3. Extraction. The curator processes the contrastive batches and applies its operations to the rule bank.
4. Pruning. Rules whose support has fallen away are removed, and well supported rules are promoted.

4.6.4 Prompt injection

The rule bank is capped at a fixed number of active rules. When it is full, the curator must downvote a rule before a new one can be added. The rules in the bank are creates as natural language guidelines of the form `"when [trigger], [action]"` and placed in the agent's system prompt. In these experiments the bank stays small enough that every rule is injected on every task. This is done as the thesis does not focus on retrieval.

4.6.5 The GPT-5.4-mini experiment

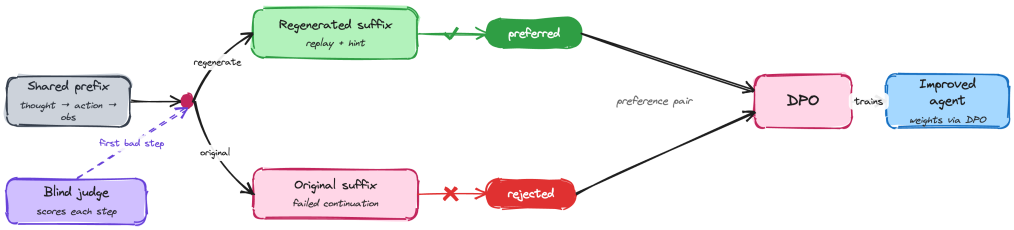
To test whether the curated rules help an agent strong enough to act on them, the memory pipeline is also run with GPT-5.4-mini in place of the Qwen3.5-9B agent and that the memory track is the only track that allows for a foundation model to be used as it does not require fine-tuning the model. The GPT-5.4-mini agent is run on the retail and airline tasks to collect a baseline, rules are curated from those judged traces by a GPT-5.4 curator, and the agent is then run again with the rules injected as described section 4.6.1

4.6.6 Evaluation

The memory pipeline is evaluated the same way as the other tracks with a baseline which the rules are curated from, and the model with the rules injected into. For retail and airline

the pools are the official τ^2-bench splits, scored by the official τ^2 evaluator. For HotPotQA the pool is the held out question set, scored as in Track 1. For the two GPT-5.4-mini τ^2-bench deployments each condition is run three times so that $pass^k$ reliability can be reported.

4.7 Track 3 - Replay-based preference learning



Track 3 implements the preference update operator of Figure 4.7 : instead of reshaping the trajectory distribution (4.5) or shifting terminal output decisions through prompt context (4.6), it shifts the weight distribution over choices by training the agent against corrective prefrence pairs reconstructed from failed traces. Many failed traces are unrecovarable, they go wrong at an identifiable point, and the rest of the trajectory beyond that point can be replayed towards a new direction. The original failed suffix and the regenerated corrective suffix together form a prefrence pair on which DPO can be trained. Unlike 4.5, the trace source is StableToolBench rather than HotPotQA.

4.7.1 Tree-replay generation

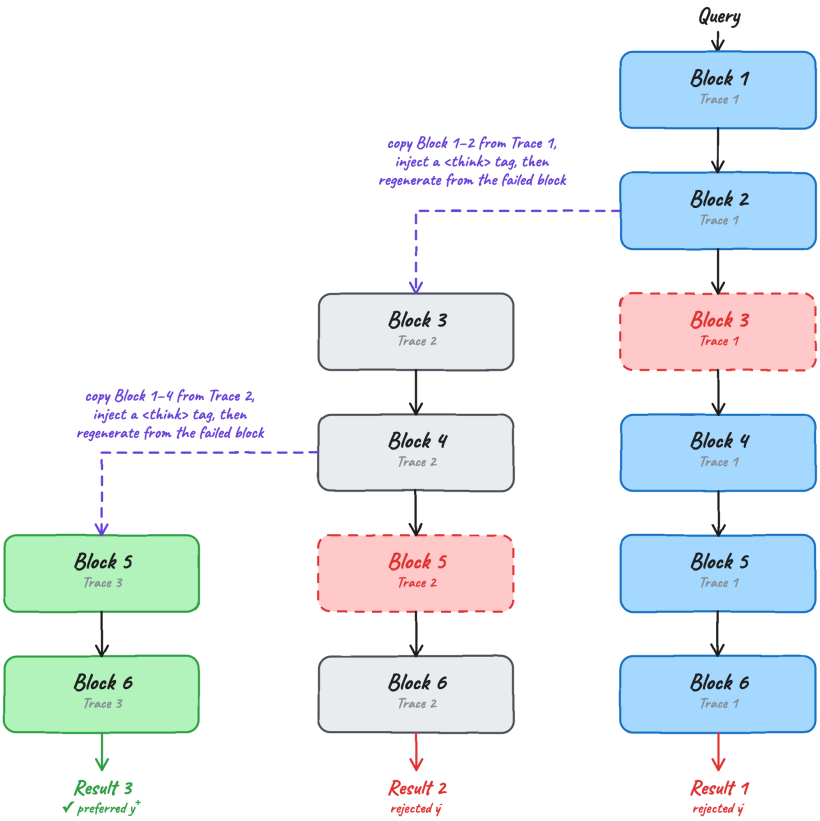


Figure 4.7: Track 3 branch-and-replay trace generation. The per-block judge marks the first bad step (red) in the root trajectory (blue); each replay copies the prefix, injects a corrective tip via the Qwen <think> channel, and regenerates the suffix until one passes (green), yielding a DPO preference pairs with one passed trace and one rejected trace.

Trace generation follows a per-block branch and replay procedure (Figure 4.7). For each StableToolBench query the agent is run once to produce a root trajectory, a per-block judge then scores every agent turn in isolation and identifies the first bad step, the earliest agent turn whose dimensions scores fall below a per dimension score. The judge also emits a thinking step , for example "call *ListAllTools* before issuing a category filter". This thinking step does not include any answer, but a guidance to improve the trace generation. This is then injected into the prior agent turn's **reasoning content** block, exploiting the agent **<thinking>** channel to condition the next sample without altering the visible trajectory. Rejudging each new generated trace independently with the same rubric. Repeating this for all failed queries in the StableToolBench G1 and G2 subsets.

4.7.2 DPO Training

Preference learning is performed with **DPOTrainer** from the TRL library [27] on top of the Qwen3.5-9B base model loaded in 4-bit NF4 precision with double quantisation and a bfloat16 compute dtype, following QLoRA Dettmeres [6].

LoRA adapters of rank r = 16 with scaling factor $\alpha = 32$ and adapter drouput 0.05 are inserted into the attention projection matrices (Query,Key,Value) and the feed-forward projection matrices (gate,up,down) of every transformer layer.

Listing 4.2: DPO Training

```
from trl import DPOConfig, DPOTrainer

config = DPOConfig(
    output_dir='./dpo_output',
    num_train_epochs=3,
    per_device_train_batch_size=1,
    gradient_accumulation_steps=8,
    learning_rate=5e-5,
    lr_scheduler_type='cosine',
    warmup_ratio=0.1,
    logging_steps=10,
    save_steps=100,
    save_total_limit=2,
    bf16=True,
    optim='paged_adamw_8bit',
    beta=0.1,
    max_length=4096,
    truncation_mode='keep_end',
    remove_unused_columns=False,
    report_to='none',
    gradient_checkpointing=True,
    seed=42,
)

trainer = DPOTrainer(
    model=model,
    args=config,
    train_dataset=ds,
    processing_class=tokenizer,
)

trainer.train()
trainer.save_model('./dpo_final')
```

4.7.3 Evaluation

Track 3 is evaluated on two benchmakrs, neither of which is part of the training data. HotPotQA with multi-hop question answering, τ^2-retail multi-turn with customer-service tool use, a setting that shares neither the tool surface nor the interaction structure of the training traces.

The model is evaluated on the HotPotQA held-out distribution used by Trace 1, restricted to the 314 questions whose *level* field is **hard**. The Qwen3.5-9B is served with vLLM and runs the ReAct loop with the **search** and **lookup** tools. Each test question yields one trajectory. Three answer-grading metrics are reported. Exact Match and Token F1 and the semantic judge are computed against the golden answer.

The same checkpoint is then evaluated on the domain of τ^2 retail with 114 tasks, under the official τ^2 benchmark evaluation protocol.

5 Results

This chapter reports the findings in four parts. The blind judge of Section 4.4.2 is validated first, as its signal drives all three tracks, followed by one section per track against the same Qwen3.5-9B baseline. In every table a higher value is better unless the header marks it with ↓, and the best value per column is in bold. The findings are presented here and interpreted in Chapter 6.

- **Judge** (Section 5.1): the process-quality score separates successful from failed traces on every benchmark, the gap widest on ToolHop and narrowest on τ^2-bench retail.
- **Track 1, filtered SFT** (Section 5.2): filtering raised the judge's own process-quality score but did not convert it into task success, unchanged on HotPotQA and regressing on τ^2-bench retail.
- **Track 2, memory rules** (Section 5.3): the rule bank lifts the GPT-5.4-mini τ^2 deployments ($+6.1$ retail, $+4.0$ airline) and stays within noise on Qwen3.5-9B and HotPotQA.
- **Track 3, replay** (Section 5.4): SoPR gains on the single-tool G1 subsets, flat on G2, none on held-out G3, and at most a marginal cross-benchmark gain on retail that stays within run-to-run noise.

5.1 LLM judge validation

The blind judge scores a trace without the gold answer. This section tests whether its scores agree with ground truth on HotPotQA, τ^2-bench, and ToolHop. As a graded score the process quality separates successful from failed traces on every benchmark, with the gap widest on ToolHop and narrowest on τ^2-bench retail, and as a binary verdict it reaches about 81% accuracy on HotPotQA.

On HotPotQA the binary and rubric prompts reach about 81% accuracy and 88% precision against the semantic-judge ground truth (Table 5.1), agreeing within 0.6 points on every metric, so the rubric adds its four dimension scores at no cost to the verdict. The confusion behind the verdicts, an 11% to 12% false-positive rate among accepted traces, is given in Table 8.1.

Table 5.1: Blind judge verdict on HotPotQA, against the semantic-judge ground truth over 493 traces, for the binary pass/fail prompt and the four-dimension rubric prompt. The two prompts agree within 0.6 points on every metric, so the rubric adds its dimension scores at no cost to the verdict.

Condition	Accuracy ↑	Precision ↑	Recall ↑	F1 ↑	Avg. Conf.
Binary, blind	**81.1%**	**88.1%**	82.4%	**85.1%**	89.9%
Rubric, blind	80.7%	87.6%	**82.5%**	84.9%	89.3%

Filtering accepted traces by the judge's self-reported confidence trades recall for precision (Table 5.2), where precision reaches 91.6% at the 90% threshold while keeping 249 of the traces.

Table 5.2: Precision–recall trade-off for the binary blind judge at rising confidence thresholds on HotPotQA. Curated is the number of traces kept, Wrongful the false positives among them. The median self-reported confidence among the false positives is 92%, so confidence buys cleaner data only by discarding correct traces rather than by isolating the wrong ones.

Threshold	Precision ↑	Recall ↑	Curated	Wrongful ↓
$\geqslant$ 70%	88.0%	**82.0%**	301	36
$\geqslant$ 75%	88.8%	81.1%	295	33
$\geqslant$ 80%	90.2%	76.8%	275	27
$\geqslant$ 85%	90.9%	74.6%	265	24
$\geqslant$ 90%	91.6%	70.6%	249	21
$\geqslant$ 95%	**94.5%**	48.0%	164	**9**

On τ^2-bench and ToolHop the judge produces a graded process-quality (PQ) score (Section 4.4.1). Official success rises monotonically with PQ on both τ^2-bench splits (Figure 5.1), with the per-band success rates reported in Table 8.2 for τ^2-bench and Table 8.3 for ToolHop. The success rate climbs from 27.3% below PQ 4 to 84.2% at PQ $\geqslant$ 4.5 on retail, and most sharply on ToolHop, from 16.2% to 91.1%.

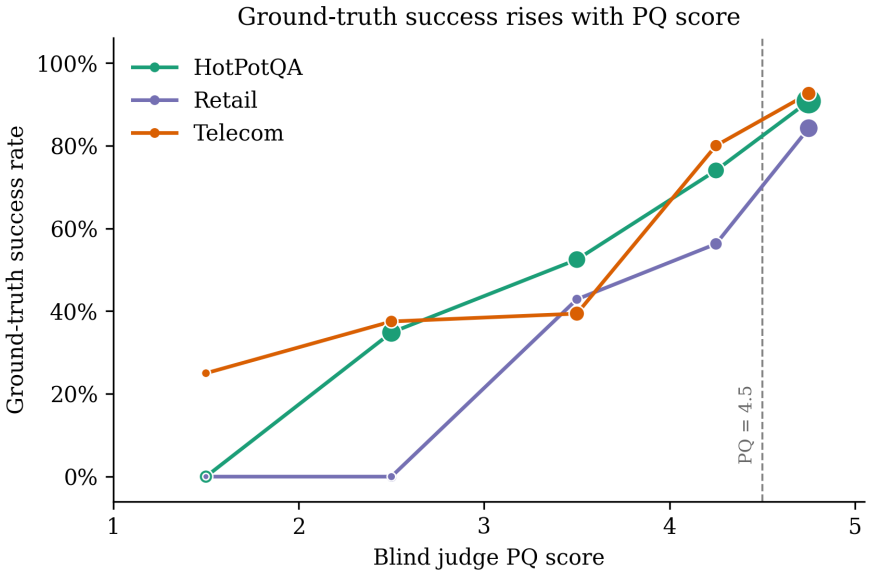


Figure 5.1: Official task-success rate against the blind judge's process-quality score on the τ^2-bench baseline traces. Success rises monotonically with PQ on both retail and telecom, the gradient a usable judge must produce.

The per-dimension separation across all four benchmark settings is collected in Table 5.3. The judge reads competence from how evidence is used rather than whether it was retrieved on HotPotQA (Answer Specificity gap 2.25 against Evidence Retrieval 0.69), and the separation is widest on ToolHop and narrowest on τ^2-bench retail, where no dimension separates passed from failed traces by even one point.

Table 5.3: Per-dimension separation of the blind judge across benchmarks. Success is the mean dimension score on the positive class (the correct-answer class on HotPotQA and ToolHop, the passed-task class on τ^2-bench) and Failure the mean on its complement, with Gap their difference. Bold marks the most separating dimension per benchmark. Separation is widest on ToolHop and narrowest on τ^2-bench retail.

Dimension	Success	Failure	Gap ↑
HotPotQA			
Evidence Retrieval	3.49	2.80	0.69
Evidence–Answer Alignment	4.41	2.36	2.04
Reasoning Coherence	4.47	2.93	1.54
Answer Specificity	4.69	2.43	**2.25**
τ^2-bench retail			
Action Quality	4.78	3.89	**0.89**
Execution Efficiency	4.51	4.30	0.21
Task Completion	4.86	4.15	0.71
Process Quality	4.72	4.11	0.61
τ^2-bench telecom			
Action Quality	4.02	3.74	0.28
Execution Efficiency	3.61	2.71	0.90
Task Completion	4.28	2.68	**1.60**
Process Quality	3.97	3.04	0.93
ToolHop			
Action Quality	4.45	2.15	2.30
Execution Efficiency	3.94	1.23	2.71
Task Completion	4.99	1.92	**3.06**
Process Quality	4.46	1.77	2.69

5.2 Track 1: filtered supervised fine-tuning

The agent is fine-tuned on its own traces, kept by one of three filters: Oracle (correct by gold answer), Blind (correct by the blind judge), and Strict blind (Blind plus PQ $\geqslant$ 4.5). Across both benchmarks the filters did not convert into task success, and filter purity did not predict the outcome, as the cleaner Oracle filter regressed further on retail than the judge-filtered Strict blind. On HotPotQA accuracy stays within about one point of Baseline on every metric (Table 5.4), and on τ^2-bench retail both filters lower task success (Table 5.5). On telecom both Baseline and Blind sit at the ceiling (97.4% and 97.2%), leaving no headroom.

Table 5.4: HotPotQA held-out evaluation, matched on the $n = 311$ questions answered by every variant. No filter moves accuracy beyond about one point of Baseline, and Strict blind is first or tied first on every metric.

Condition	Training traces	Exact match ↑	Token F1 ↑	Semantic judge ↑
Baseline	—	**43.1%**	56.6%	66.2%
Oracle	662	41.2%	55.5%	66.2%
Blind	608	42.8%	57.1%	67.2%
Strict blind	487	**43.1%**	**57.7%**	**67.5%**

Table 5.5: Filtered SFT on τ^2-bench retail, $n = 3$ trials of 114 tasks, scored by the official evaluator under the GPT-4.1-mini user simulator. pass^k is the chance all k rollouts pass. Both filters lower task success, Oracle by 7.9 points on pass^1 and Strict blind by 4.4, while tool calls per task fall slightly (no arrow, as fewer helps only at equal success).

Condition	Training traces	pass^1 ↑	pass^2 ↑	pass^3 ↑	Tool calls/task
Baseline	—	**71.9%**	**58.2%**	**49.1%**	7.5
Oracle	400	64.0%	49.4%	43.9%	7.3
Strict blind	279	67.5%	50.0%	39.5%	7.1

The retail regression is concentrated in wrong-action trials, with the per-task outcome transitions in Table 8.4 and the failure budget in Table 8.5. The judge's own scores on the fine-tuned models move in opposite directions on the two benchmarks, up to $+0.11$ on HotPotQA (Table 5.6) and down to -0.09 on retail (Table 5.7), the retail drop matching the pass-rate ordering.

Table 5.6: Blind judge scores on the fine-tuned HotPotQA models (four dimensions, mean is PQ). Every SFT condition raises PQ above Baseline's 3.77 by at most 0.11, so the agent learns the rubric surface without converting it to accuracy.

Condition	Evidence Retr.	Evidence–Ans.	Reasoning	Answer Spec.	PQ ↑	Δ PQ
Baseline	3.82	3.63	3.70	3.92	3.77	—
Oracle	3.89	3.71	3.82	3.95	3.84	+0.08
Blind	3.92	3.78	3.81	4.01	**3.88**	+0.11
Strict blind	3.91	3.78	3.89	3.94	**3.88**	+0.11

Table 5.7: Blind judge scores on the fine-tuned retail models (three dimensions, mean is PQ). Both filters lower PQ, in the same order as the pass rate.

Condition	Action Quality	Exec. Efficiency	Task Completion	PQ ↑	Δ PQ
Baseline	4.41	4.44	4.55	**4.47**	—
Oracle	4.27	4.43	4.45	4.38	−0.09
Strict blind	4.34	4.39	4.48	4.40	−0.07

5.3 Track 2: memory rule curation

Rules are curated from the agent's judged traces and injected at inference, a sample bank shown in Table 5.8. The rule bank helped only the two GPT-5.4-mini τ^2-bench deployments, +6.1 pp on retail and +4.0 pp on airline, while the Qwen3.5-9B and HotPotQA deployments stayed within 1.3 points (Table 5.9). The gain holds but narrows across three runs (Table 5.10) and is mirrored by a small rise in the judge's own PQ score (Table 5.11).

Table 5.8: Representative curated rules. The retail and HotPotQA banks are Qwen3.5-9B's, the airline bank GPT-5.4-mini's. Each rule is a "when trigger, action" guideline injected into the system prompt.

#	Rule
τ^2-bench retail	
1	When a request spans multiple orders, complete each requested action with the tools for that order before replying.
2	When changing items in a pending order, call `modify_pending_order_items` with the item ids and the matching replacement item ids.
3	When swapping a delivered item variant, call `exchange_delivered_order_items` with the item ids and replacement item ids.
HotPotQA	
1	When the question is a bridge question with a role phrase such as "founder", search the anchor entity, look up the role phrase, then search the person it returns.
2	When the question compares two named entities, search each once, then look up the requested attribute on each page before deciding.
τ^2-bench airline	
1	When the user lacks reservation ids, use the ids returned by `get_user_details` before calling `get_reservation_details`.
2	When a reservation is ineligible after `get_reservation_details`, call `transfer_to_human_agents` once.

Table 5.9: Task success with the rule bank injected against no rules, n tasks matched across both runs. Only the two GPT-5.4-mini τ^2 deployments move beyond a point (+6.1 retail, +4.0 airline), the three others stay within 1.3.

Benchmark	Agent	n	No rules ↑	With rules ↑	Δ (pp)
τ^2-bench retail	Qwen3.5-9B	114	**72.9%**	**72.9%**	0.0
τ^2-bench retail	GPT-5.4-mini	114	57.9%	**64.0%**	+6.1
τ^2-bench airline	GPT-5.4-mini	50	58.0%	**62.0%**	+4.0
HotPotQA	Qwen3.5-9B	314	66.2%	**67.5%**	+1.3
HotPotQA	GPT-5.4-mini	314	**66.6%**	65.3%	−1.3

Table 5.10: GPT-5.4-mini pass^k on τ^2-bench over three runs, rules against no rules. pass^k is the chance a task passes in all k runs. The mean gain narrows at higher k.

Deployment	Metric	No rules ↑	With rules ↑	Δ (pp)
retail	pass^1	59.3%	**64.3%**	+5.0
	pass^2	43.1%	**48.5%**	+5.4
	pass^3	35.4%	**38.6%**	+3.2
airline	pass^1	58.5%	**60.0%**	+1.5
	pass^2	46.9%	**51.3%**	+4.4
	pass^3	40.8%	**44.0%**	+3.2

Table 5.11: Blind judge scores for the three τ^2 memory deployments, rules against no rules (PQ is the mean of the three dimensions). PQ rises 0.04 to 0.10 with rules, the same direction as task success.

Deployment	Dimension	No rules ↑	With rules ↑	Δ
retail Qwen3.5-9B	Action Quality	4.24	**4.52**	+0.28
	Execution Efficiency	**4.41**	4.34	−0.07
	Task Completion	4.45	**4.54**	+0.09
	Process Quality	4.37	**4.47**	+0.10
retail GPT-5.4-mini	Action Quality	4.34	**4.38**	+0.04
	Execution Efficiency	4.49	**4.55**	+0.07
	Task Completion	4.38	**4.39**	+0.01
	Process Quality	4.40	**4.44**	+0.04
airline GPT-5.4-mini	Action Quality	3.51	**3.58**	+0.07
	Execution Efficiency	**4.00**	3.98	−0.02
	Task Completion	3.18	**3.24**	+0.07
	Process Quality	3.56	**3.60**	+0.04

Reading the retail tasks whose outcome changed, every GPT-5.4-mini recovery attributable to a rule follows one mechanism, looking up the account's orders once authentication succeeds, while on Qwen3.5-9B no change attributable to a rule is a recovery (Table 8.6).

5.4 Track 3: replay-based preference learning

The Qwen3.5-9B agent is trained on corrective preference pairs from StableToolBench G1 and G2 (Section 2.2.2) and evaluated against the same base agent. The gains concentrate on the single-tool G1 subsets, fade to flat on the multi-tool G2 subsets, disappear on the held-out G3 subset, and leave at most a marginal gain on cross-benchmark retail that stays within run-to-run noise (Tables 5.12 and 5.13).

Table 5.12: SoPR on StableToolBench under the GPT-5.4-mini judge, trained against baseline, Δ in points. SoPR maps the Solved/Unsure/Unsolved verdict to 1/0.5/0. Gains concentrate on the single-tool G1 subsets (pooled +5.3), vanish on multi-tool G2 (pooled +0.2), and reverse on the held-out G3 subset (−2.5).

Split	n	Baseline ↑	Trained ↑	Δ (pp)
G1_INSTRUCTION	95	16.8%	**19.5%**	+2.6
G1_CATEGORY	149	25.8%	**33.9%**	+8.1
G1_TOOL	113	15.0%	**19.0%**	+4.0
G2_INSTRUCTION	102	**13.7%**	10.3%	−3.4
G2_CATEGORY	120	17.1%	**20.4%**	+3.3
Pooled G1	357	20.0%	**25.4%**	+5.3
Pooled G2	222	15.5%	**15.8%**	+0.2
Total	579	18.3%	**21.7%**	+3.4
G3 (held out)	61	**16.4%**	13.9%	−2.5

Table 5.13: Replay-trained model against the same three-run baseline as Table 5.5 on the 114 τ^2-bench retail tasks, official evaluator under the GPT-4.1-mini user simulator. pass^k is the chance all k runs pass. The trained model leads the baseline by 2.1 points at pass^1, 2.9 at pass^2, and 5.3 at pass^3, with the pass^1 and pass^2 gaps inside the measured ± 5 pp retail noise. On HotPotQA both models score 67.1%, an exact tie.

Dataset	Metric	Baseline ↑	Trained ↑
τ^2-bench retail	pass^1	71.9%	**74.0%**
	pass^2	58.2%	**61.1%**
	pass^3	49.1%	**54.4%**

6 Discussion

This chapter analyses the results, the methodology, and the broader implications of the work. It begins with a detailed examination of the findings reported in Chapter 5, considering each in the context of the research questions of Section 1.3. It then reviews the methodology critically, from the design of the experiments to the choices that may have shaped the outcomes. Finally it situates the work in a wider context, addressing its practical relevance and its societal and ethical aspects.

6.1 Results

This section discusses what the results mean, how reliable the blind rubric judge is as a shared signal, and why each track did or did not work, covering the filtering, memory curation, and replay-based learning tracks in turn.

6.1.1 Judge

The judge is the part of this thesis that everything else rests on, since all three tracks use its verdict. That makes RQ1 the first question to answer, whether a blind, multi-dimensional rubric judge can score the quality of agentic traces without gold answers, and whether that signal carries over to benchmarks with different structures. The validation came in two parts. On HotPotQA the blind judge scored 493 traces against ground truth in two prompt conditions, a plain binary verdict and the multi-dimensional rubric, both reported in Table 5.1. A cross-benchmark study on τ^2-bench and ToolHop then checked whether the judge's graded process-quality score moves with official task-success outcomes. Together they give an answer, the blind judge is a usable but imperfect signal, and how reliable it is depends on how much of a benchmark's failures show up in the trace the judge reads, and on how the rubric is set up.

Verdict and PQ answer different questions The blind verdict and the PQ score answer two different questions. The verdict asks whether the judge can judge at all without the gold answer, and on HotPotQA it reaches about 81% accuracy and 88% precision (Table 5.1). PQ asks a ranking question instead, does a trace-only score move in the same direction as official success? It does on every benchmark, the official pass rate climbs as PQ rises, on retail from

27.3% below PQ 4 to 84.2% at PQ $\geqslant$ 4.5, on telecom from 37.7% to 92.6%, and on ToolHop from 16.2% to 91.1% (Table 8.2 and Table 8.3).

Rubric design and the domain gap The rubric should be read as a process rubric, not as a hidden way to recover the answer. On HotPotQA the dimensions with the largest passed-failed gaps are Answer Specificity (2.25) and Evidence-Answer Alignment (2.04), while Evidence Retrieval separates the two classes only weakly (0.69), shown in Table 5.3. This says the judge is strongest when the trace shows how the agent used the evidence, and weakest when correctness depends on something the trace does not show. The cross-benchmark results make the same point at the domain level, the process-quality score reaches AUC 0.94 on ToolHop, 0.79 on τ^2-bench telecom and 0.69 on τ^2-bench retail (Table 8.2 and Table 8.3). Retail is the hardest case because success is decided by the final state of the database and by policy checks, and no retail rubric dimension separates passed from failed traces by even one point. The gap between domains is therefore not just measurement noise. It is a built-in limit of a judge that only sees the trace.

What a PQ gap means inside the trace A large gap between the mean PQ of passed traces and the mean PQ of failed traces means the trace holds visible process features that go together with correctness, grounded use of evidence, specific answers, coherent reasoning, clean tool choice, efficient execution, and a finished task. It does not mean the judge has recovered the hidden gold answer. When two traces both succeed, PQ still has a use, it points to the better training example. A high-PQ success is more likely to be direct, grounded and repeatable, while a low-PQ success may have reached the right answer through luck, a long answer, or a weak explanation. That difference is exactly what trace reuse needs, because SFT and DPO are not only learning which final outcomes matter. They are also learning which ways of solving a task are worth copying.

Two case studies of PQ on successful traces To show what PQ adds on traces that already succeeded, we looked at two pairs where both traces passed the official outcome but the rubric judge gave one a low score and the other a high one. On HotPotQA the low- PQ trace (PQ 2.25) answered a bridge question about a financial company with approximately 6,800 employees, settled on Sampo Group, and added in its own final answer "though the user's approx 6,800 employees is not exactly listed for Sampo, it is the closest match from the search results". The high-PQ trace (PQ 4.5) answered a comparable bridge question about an actor in a 2012 Muccino film by retrieving one page that satisfied every clause of the question (year, country, genre, director) and one page that named the actor and the role together. Both passed exact match. Only one reached the answer through a derivation the trace itself supports.

On τ^2-bench retail, two multi-workflow Qwen-3.5-9B baseline tasks both passed the official evaluator. The low-PQ trace, at PQ 3.0, tried to return a delivered item to the customer's credit card, received the error "Payment method should be the original payment method", silently retried on PayPal, and announced "I have successfully completed both actions" without asking. The customer replied "Oh, wait, I thought the refund for the hiking boots would be on my credit card too, not PayPal" and the conversation ended in escalation. The high-PQ trace, at PQ 5.0, gated every write on an explicit per-action user confirmation, made no failed tool calls, and exited cleanly. In both pairs the official outcome was met, yet the trace itself held either an admission that the evidence did not support the answer or a tool call the customer had not approved. The blind judge sees this, and the official outcome cannot.

Confidence could not clean the set The tracks filter on the judge's verdict and process-quality score, not on its confidence, but it is worth asking whether confidence could have cleaned the kept traces further. It could not. The judge reports its own confidence score, and raising the threshold from 70% to 95% lifts precision from 88.0% to 94.5% but drops

recall from 82% to 48%, so a high-precision cut throws away more than half of the correct traces (Table 5.2). The false positives are confident too, with a median confidence of 92%, so confidence cannot tell the judge's right answers from its wrong ones without also losing recall. Even at the 90% threshold precision reaches only 91.6% at 70.6% recall, so about 8% noise would survive the strictest setting that still keeps usable data. Confidence thresholding could trade recall for precision, but it could not make the set clean, which is why the tracks rely on the verdict and the process-quality threshold instead.

What the signal is good for For trace reuse the ranking matters more than recall. SFT uses the judge as a filter and only needs the top traces worth copying, while DPO uses it as a contrast generator and needs both the top traces and the worst traces to move away from. Neither needs every correct trace recovered, so the pipeline can run at low recall as long as it is clean at the two extremes. A confident false positive at the high end teaches the model the wrong behaviour, and a noisy low end makes the DPO pairs unclear, which is why being clean at the two ends matters more than the lost recall. This is also why the rubric is worth using even though it does not beat the plain binary verdict, the dimension scores give reuse a structure that a single verdict does not.

6.1.2 Filtered supervised finetuning

The filtered fine-tuning track did not improve task success over the baseline (Table 5.4). The idea was taken from earlier work that uses generated traces to improve a model through filtered fine-tuning. The hope was that if the judge could tell good traces from bad ones, that signal could be used to pick the traces the model trains on, and because the traces are scored by the rubric judge and not by gold answers, the model would learn good behaviour rather than overfit to specific answers. On HotPotQA the training set was kept separate from the evaluation set, so the filtered set should have taught the model how a strong trace is done. It did not. Accuracy stayed within about one point of the baseline on every metric, and the only steady change was shorter messages and fewer tokens.

The clearest result is that the judge's own score and task success came apart. The rubric scores rose from baseline to strict blind across all four HotPotQA dimensions, with Reasoning Coherence moving the most at $+0.19$ (Table 5.6), yet none of this turned into higher benchmark accuracy. The model learned the surface features the rubric rewards without learning to solve more tasks. Retail makes the same point more sharply. The 100% gold-correct Oracle filter fell from 71.9% to 64.0% on pass[1], a 7.9 pp drop that clears the ± 5 pp noise floor, while the judge-filtered Strict blind fell to 67.5%, a smaller drop the floor cannot tell from noise (Table 5.5). Since even a gold-perfect filter regressed beyond noise, the problem was not the judge's selection, fine-tuning the agent on its own traces simply added no skill it did not already have.

A few things could explain the missing gain. The filtered training sets may have been too small, as LoRA can need thousands of examples. The signal from a single good trace may be too weak to move the model. And because the training traces come from the same model being fine-tuned, they may already sit inside its distribution, so there is little new for SFT to learn.

6.1.3 Memory

The memory track helped GPT-5.4-mini on τ^2-bench but not on any other benchmark. On GPT-5.4-mini the rule bank moved retail by $+6.1$ pp and airline by $+4.0$ pp (Table 5.9). On Qwen-3.5-9B the same pipeline left retail within noise of the 72.9% baseline, and HotPotQA stayed within noise on both agents. The two non-results have different causes, set out below.

Qwen-3.5-9B did not improve on retail because it could not follow the rules well enough. Among the retail tasks whose outcome changed, no Qwen recovery can be traced to a rule,

while three regressions can (Table 8.6). In two of the three the agent used a delivered-order rule on a pending-order request and refused an exchange the baseline would have handled, and in the third it applied a multi-order address change to a request that named only one order. The rules fired, but the agent could not tell when they applied and when they did not.

HotPotQA did not move on either agent, for a different reason. Its failures are not about which tool is called in which order, they are about how well the model can search and reason. Few-shot prompting can teach a model a procedure, but it cannot give it more reasoning depth, so with no procedural failure for the rules to fix, the bank had nothing to work with and the held-out change stayed within noise.

Where the rules do help, the help is focused rather than spread out. The GPT-5.4-mini retail bank was built from GPT-5.4-mini's own baseline traces, and of the recovered tasks traced to a rule, seventeen share one mechanism. The baseline often authenticated the customer and then stopped, asking for an order id and ending the conversation when none was given. The rules told the agent, once authentication had succeeded, to look the account's orders up and call `get_order_details` on each, and all seventeen recovered tasks followed this route. The blind judge saw the same change in its own scores, with process quality up by 0.04 points on both retail and airline (Table 5.11), a small but steady rise that confirms the rules pushed the agent toward behaviour the rubric also rates higher.

Two design choices set the limits of the bank. First, the whole bank is injected on every task. It is capped at fifteen rules, and every rule sits in the prompt whether or not it fits the current task, which is what caused the Qwen regressions, a delivered-order rule fired on a pending-order request because nothing filtered the rules per task. Picking only the rules whose trigger matches the task would avoid this. Second, the curation has no check between rule updates, a change is made to the bank and the next run is observed, but there is no way to undo a change if the new bank does worse.

6.1.4 DPO

Track 3 addresses RQ4, whether preference pairs rebuilt from an agent's own failed traces, with the judge giving the corrective direction, can improve the agent without gold labels. The trained model was compared against the same baseline in three places, on the StableToolBench G1 and G2 subsets the pairs were drawn from, on the held-out G3 subset of the same benchmark, and on HotPotQA and τ^2-bench retail. Read together they point to a model that improves on the simple single-tool tasks, with the gain shrinking as a task needs more tools and more planning.

StableToolBench The in distribution result is a spread out rather than uniform gain (Table 5.12). Pooled across the three G1 subsets the SoPR rises from 20.0% to 25.4%, a gain of 5.3 percentage points, with a large single gain on G1 category of 8.1 pp. Pooled across the G2 subsets it is flat at 0.2 pp, G2 category gaining 3.3 pp while G2 instruction regresses by 3.4 pp, and on the held-out G3 subset it falls by 2.5 pp. The benefit is concentrated where a task is solvable with a single tool and gets worse as the task demands more tools across different categories.

This follows from how the pairs are built. Every pair shares its full prefix and differs only in the suffix after a single bad step the judge picked out, so the training signal is focused on one local decision, recovering from a specific mistake. That fits single-tool tasks, where success usually turns on one such decision, and does not fit G3, where success turns on planning a longer task. The model learns to fix a step, not to plan, and the ordering of the gains, G1 above G2 above G3, is exactly what that difference predicts.

Retail Retail shares neither the tools nor the multi-turn setup of the StableToolBench training traces, so it tests transfer to a different distribution. Against the matched three-run baseline the trained model leads by 2.1 points at pass^1 (74.0% against 71.9%) and 2.9 at pass^2

(61.1% against 58.2%), both inside the measured ± 5 pp retail noise, so neither is a reliable effect (Table 5.13). The gap grows to 5.3 points at pass^3 (54.4% against 49.1%), which sits at the edge of that noise band. As far as this experiment can show, replay-based preference learning gives at most a small transfer to retail, and what there is shows up in the all-three-runs-pass case rather than in single runs.

Retail noise The repeated measurement is itself a finding, it shows that a single retail run is a noisy estimate. pass^k falls steeply for the trained model, from 74.0% at $k = 1$ to 61.1% at $k = 2$ to 54.4% at $k = 3$, and the baseline falls the same way, from 71.9% to 58.2% to 49.1% (Table 5.13). Only 54.4% of retail tasks pass all three trained runs against a 74.0% single-run rate, a gap of close to twenty points, and that is the size of run-to-run variation any small transfer claim on this benchmark has to beat.

HotPotQA HotPotQA, the other out-of-distribution benchmark, gives the same answer by a different route, the trained and baseline models tie at 67.1% on the hard split (Table 5.13). This fits the in-distribution result. HotPotQA's hard questions fail on search and reasoning depth, which needs composition, not on a single fixable step, and its read-only search setup is unlike the tool-composition traces the pairs came from, so the single-step recovery signal has nothing to act on.

Taken together, Track 3 produced one clear effect, a gain on single-tool tasks inside the training distribution on StableToolBench. The answer to RQ4 is therefore narrow. As built here, replay-based preference learning helps the agent on the simplest tasks and shows at most a small, within-noise gain elsewhere, so the evidence does not support a claim of general or transferable improvement.

6.2 Method

This sections cover a discussion of the methods used in the thesis that led to answers.

6.2.1 Methodological considerations

The choice of benchmark affects all parts of the thesis. When choosing a benchmark, the experiments are locked into that benchmark's design and domain. All the methods used in this thesis require a trace gathering step to simulate traces being executed in online production, and for that it is necessary that the benchmark has a reliable number of tasks to use, and ideally a clear split between train and eval. Most of the benchmarks for evaluating agents are quite new and have a small task size. The choice of HotPotQA was made because HotPotQA is a large benchmark with both a train and an eval split. For the direction of the work HotPotQA is not a great benchmark, as it only depends on the two tools search and lookup, and is more of a general reasoning benchmark than a domain-learning benchmark

For multi-hop tool benches the choice was more difficult. Our criteria were that the benchmark must be difficult for modern models, popular, and have a large corpus. τ^2-bench was the option that met those criteria. Other benchmarks such as ToolHop are good in their tasks, but they are not domain-specific, since each task has its own tools, which is not what the thesis sets out to test. In the end we went with HotPotQA and τ^2-bench retail as the primary benchmarks, since Qwen-3.5-9B had a baseline we could improve on retail. Airline at 50 tasks seemed too low for us. We also tried telecom, since telecom offers the ability to generate synthetically more tasks, but we realised the baseline ceilings out. With that we decided to move on to retail as the primary eval benchmark.

The multi-hop tool bench tasks we evaluate require tens of thousands of tokens, and tasks can span up to 100 steps. With this in mind there are many levels of noise that make the domain difficult to work in. Language models are noisy by default under the reasoning

setup we use, with the temperature and sampling strategy needed for running Qwen. Besides that, the user simulation also introduces noise, since it makes different judgements on similar topics across runs. All in all it makes the retail domain difficult to work with. With the small sample sizes for tau-2 we had to use the eval set as both a training and eval set, which is not preferable in any machine learning setting.

For the choice of base model we chose Qwen-3.5-9B as our baseline. It is a powerful and new model that runs on the A100 we had access to. We also wanted a native reasoning model, since the strongest models today are all reasoning models, and for our thesis the reasoning acts as a signal that can be analysed and used for grading. For the thesis we believe a 7B and a 14B model would have worked too. If an older model is chosen the gains or losses would no longer be as interesting, since it would not be a frontier model. One thing we were not satisfied with on the model was that the instruction-following on the memory benchmark did not seem as good with Qwen-3.5-9B. This led us to choose GPT-5.4-mini as a separate evaluation, to see whether rules could be injected into a model that could act on them, which it showed.

6.2.2 Replicability

The procedure of this thesis is written down in the Method chapter and the settings and code are available in the public repository, so the pipeline can be rebuilt and rerun. What cannot be promised is that a rerun returns the same numbers, and the reasons are worth stating plainly, because they apply to every number the thesis reports.

The first source of variation is the agent itself. Qwen-3.5-9B is sampled at temperature 0.6 and top-p 0.95, which is the setting the reasoning agent needs, so two runs of the same agent on the same task can take different paths and reach different outcomes. The same trace cannot be reproduced exactly. The second source is the simulated user. On τ^2-bench the user is GPT-4.1-mini, and it does not repeat its turns word for word across runs, so it can push otherwise identical tasks into very different conversations. These two sources add up on the state-changing benchmark, where one different user turn early in a task can change the final database state the official evaluator checks.

In practice this gives a measured noise floor, not an assumed one. Two runs of the same baseline agent on τ^2-bench retail differ by about ± 5 pp overall at $n = 3$, and the repeated pass^k measurement in Table 5.13 shows how steeply single-run success drops under repetition, from 71.9% at pass^1 to 49.1% at pass^3 for the baseline. This is why the thesis does not read the small retail differences as real effects. The DPO retail gains of 2.1 pp at pass^1 and 2.9 pp at pass^2 sit inside that ± 5 pp band and are not treated as reliable, and even the 5.3 pp pass^3 gain is reported only as sitting at the edge of it (Table 5.13). The same care applies to the Track 1 retail drops in Table 5.5, where the Baseline pass^1 of 71.9% is itself just one noisy sample. Every retail comparison in this thesis is read against this floor, and any difference smaller than it is reported as not reliable.

Two more things limit replicability. The judge is fixed to a single closed model, GPT-5.4-mini at medium reasoning effort, and all judging was done in the first half of 2026. A reader who reruns the judge later faces a model the provider may have quietly updated in the meantime, so the judge results can be reproduced against that fixed model and time window rather than against the model name alone. The other limit comes from the data. The τ^2-bench task counts are small, so the eval set had to be used as both the training pool and the evaluation pool, which is not a setup any machine learning project would choose. A rerun that copies our setup has the same overlap, and it is one more reason the retail numbers should be read as a diagnostic rather than as a clean estimate of how well the method generalises to held-out data.

Would a different judge change the results? A fair concern is that the whole signal in this thesis rests on one judge, so a different model might draw the line between passed and

failed in a different place. The one check we have is an ablation that swapped the GPT-5.4-mini judge for the full GPT-5.4 model and found no real difference in its verdicts, which suggests the rubric is not just a product of the smaller model's limits. That check is limited, because both models are from the same family, so it only tells us about scale within one design, not about a very different judge built on different training data or a different rubric. Such a judge could reasonably weight the process dimensions in another way, and the place this would matter most is the retail world-state blind spot, where process quality already separates outcomes only weakly, at an AUC of 0.69 against 0.94 on ToolHop (Table 5.3). The AUC and per-dimension gaps limit this risk, since they show the current judge does carry a real signal that tells the classes apart, but they do not remove it, because they measure how well this one judge agrees, not how stable the result is across different judge designs. The honest position is that the judge is the validated result for this judge family, and showing it carries over to a very different judge is left for future work.

The numbers in this thesis can be reproduced only in distribution and only against the fixed judge and time window, so what can be safely reproduced is the direction and the order of the effects rather than any single percentage point.

6.2.3 Validity

Judge confidence One threat to the validity of the judge signal is that the judge has to read competence off the surface of the trace. The blind judge has no access to ground truth and must judge skill from the trace alone, the agent's explanation, its tool argument choices, or its final answer. These are exactly the features a confident agent can produce, whether or not the reasoning behind them is correct. The result is a confidence cascade, an agent that sounds sure of itself produces traces that the blind judge reads as more competent, and the judge then gives those traces a higher process-quality score, which is the signal the filtering and preference tracks then train on. The judge's own false positives line up with this, they carry a median self reported confidence of 92% (Table 5.2), which is what one would expect if the judge is anchored on surface features of the trace that an over-confident agent supplies. The risk is therefore that the pipeline rewards traces that look competent rather than traces that are correct.

A closed evaluation loop The replay-based results sit at the end of a loop in which one model family, GPT-5.4-mini, plays four roles, the rubric judge that scores candidate traces, the step-level critic that finds the bad action in a failed trajectory, the source of the corrective suffix that becomes the chosen side of each preference pair, and, at evaluation time, the Solvable Pass Rate scorer. The same model therefore chooses which traces are trained on, which step counts as the failure, which continuation is preferred, and finally whether the trained agent has improved. The in-distribution pass rate is itself a judge call by the same model against the same rubric used at training time, so the agent receives a strong, low noise signal in one direction, produce traces the GPT-5.4-mini rubric scores as competent. Even a judge that perfectly tracked task success would still push learning onto the behaviours it can see and reward, and an imperfect judge pushes learning onto the overlap of competent behaviour and judge preference, which the in-distribution test cannot tell apart. The pooled G1 gain of $+5.3$ pp, from 20.0% to 25.4% (Table 5.12), should be read with this in mind. The clearest sign that part of it is alignment to the judge rather than real skill is that under independent scorers the gain disappears, τ^2-bench retail moves only $+2.1$ pp at pass^1, inside the noise floor, and HotPotQA ties the baseline (Table 5.13).

Judge calibrated The rubric was never calibrated. The dimensions, their equal weight in the unweighted mean, and the process-quality threshold of 4.5 used by the strict-blind filter were all chosen by hand before the experiments and never tuned against a held-out signal. The calibration this thesis reports measures how well that fixed rubric already tracks success,

it does not adjust the rubric to track it better. There is clear room to do so. The per-dimension gaps show that some dimensions carry little signal on some domains, on retail Execution Efficiency separates passed from failed traces by only 0.21 points against 0.89 for Action Quality (Table 5.3), so an unweighted mean gives a barely discriminating dimension as much say as a useful one. A rubric whose dimensions, weights and threshold were fit per domain rather than assumed might pull a stronger signal from the same judge, and that tuning is left for future work.

6.3 The work in a wider context

This section situates the work in its larger context: what it is good for, what it could do harm, and what it costs.

6.3.1 Self-improving deployed agents

The real appeal of learning from agentic traces is that an agent gets better at its job using only the work it already does. A deployed agent produces a trace every time it serves a request, and a blind process judge can score that trace without a human writing a gold answer for it. Removing the labelling step matters in a wider sense, because the labelling step is what usually keeps an agent tied to an outside labelling pipeline and stops it from improving inside its own deployment. The real benefit of trace-driven improvement is not only a higher success rate. It is fewer wasted or dropped actions, fewer tool calls, fewer tokens, and so less time and less energy spent per task, on infrastructure where each served request has a real compute cost.

Two results in this thesis show that this efficiency benefit is real even when the accuracy benefit is not. In Track 1 the mean number of tool calls per retail task falls from 7.5 for the baseline agent to 7.3 under the gold-correct Oracle filter and 7.1 under the blind Strict filter (Table 5.5). That reduction holds across both filtered conditions even though pass[1] success does not rise with it, sitting at 71.9% for the baseline against 64.0% and 67.5% for the two filtered models (Table 5.5). An agent that reaches a similar outcome with fewer tool calls is doing less wasted work per task, which is a real deployment gain on its own. The Track 2 memory mechanism shows the other half of the picture, where wasted work is turned into a completed task. For GPT-5.4-mini the curated rule bank produced 17 rule-attributable recoveries against 3 regressions (Table 8.6). Those recoveries were baseline runs that had authenticated the customer and then stalled, so an injected rule converted a run that would have been abandoned mid-task into one that finished. Fewer abandoned runs is exactly the kind of value a self-improving deployed agent is meant to deliver, and it comes from the agent's own past traces rather than from new labelled data.

This same closed quality carries a cost. A loop that trains on the agent's own self-generated traces can lock in the agent's own biases, because the material it learns from is the material it already produces. If the judge rewards a habit the agent already has, the loop reinforces that habit instead of correcting it, and a blind spot in the trace stays a blind spot in the next round of training. The benefit and the risk share one root, which is that the agent is both the source of the data and the thing being changed by it.

The lesson to carry forward is that trace-driven self-improvement can buy a deployed agent fewer wasted actions and fewer abandoned runs without any labelling step, provided the loop is watched so that it does not simply teach the agent to repeat itself.

6.3.2 Data privacy

The self-improving loop studied in this thesis is, at its core, a pipeline that moves the content of deployment traces forward through several stages. A trace is produced by the agent in deployment, it is sent to the blind judge for scoring, and the scored trace is then reused to build

a rule bank, a supervised fine-tuning set, or a set of preference pairs. Each of these stages keeps or forwards the raw content of the interaction. When that interaction is a real user session, the trace can contain personally identifiable information, and the loop then handles that information at every step rather than discarding it after the task is done.

Two properties of the design make this more than a general worry. First, the judge is a third-party closed frontier model. The traces leave the operator's own infrastructure and are sent to an outside provider for scoring, so any user data in the trace is shared with that provider in order to get the process signal. Second, the reuse mechanisms keep trace content by design. A curated rule is drawn directly from past user interactions, and a supervised fine-tuning example is a past trace turned into training data, so user content is not just read once but is kept inside the rule bank or built into the model weights. The rule bank in our deployments is small, capped at fifteen rules added in full on every task (Table 5.9), yet even a small bank is a lasting store built from earlier sessions, and a fine-tuning set has no such cap at all.

In this thesis no real user data was ever at risk. The user in the τ^2-bench deployments is a simulated user driven by GPT-4.1-mini rather than a person, so the traces hold no real personal data and the privacy risk described here is built into the design rather than something that actually happened. A real deployment would not have that protection. It would log real interactions, and the same loop that gave a modest task lift in our experiments would then be circulating, storing, and learning from genuine user data. The risk is therefore forward-looking, something the architecture implies rather than something the experiments measured, but it follows directly from how the loop is built.

For practice this is a matter of balance, not a ban. The ability shown here, getting a usable learning signal from traces without ground-truth labels, is real, but it has to be weighed against strict data-privacy and legal rules before it is used on real user data. Sharing trace content with an outside judge, keeping user-derived content in rule banks and training sets, and a user's right to have their data removed once it has been built into model weights are all questions a deployment would have to answer under the relevant data-protection law. None of these is solved by the technical result, and each becomes binding the moment the simulated user is replaced by a real one. Any operator who wants the loop has to treat trace minimisation, consent, and the choice between an outside and a self-hosted judge as part of the system design, not as an afterthought.

A self-improving loop that learns from deployment traces takes on the privacy duties of everything those traces contain, and it can only be deployed once those duties are met.

6.3.3 Dependence on closed-source frontier models

The pipeline studied in this thesis rests almost entirely on closed-source frontier models. A single GPT-5.4-mini model family acts as the blind process-quality judge, the step-level critic and the curator that writes the rules and corrective suffixes, and in the preference-optimisation track it also scores SoPR at evaluation time. The conversations on the τ^2-bench benchmarks are driven by a GPT-4.1-mini user simulator. The only fully open part is the agent itself, the Qwen-3.5-9B model served locally, which means that everything that produces or grades the learning signal sits behind a paid commercial service and everything that is being improved is open. That imbalance is worth stating plainly, because it shapes both what can be reproduced and who can run the method at all.

The first consequence is reproducibility. The judge endpoints are closed and can be changed by the provider without notice, so a number measured against GPT-5.4-mini in the first half of 2026 is tied to whatever version of that endpoint was current then, and a later caller may get a different model behind the same name. The Method chapter and the public repository fix the prompts, the rubric and the procedure, but they cannot fix the model on the other end of the call. This is why all judging in this thesis was done in the first half of 2026 and the time period is recorded exactly, so that the figures can be read against the model ver-

sions that were live when they were taken. An ablation that swapped the GPT-5.4-mini judge for the full GPT-5.4 showed no real difference in the process-quality scores, which suggests the signal is not specific to the smaller model, but it does nothing to remove the underlying dependence on a vendor that controls and quietly updates both endpoints.

The second consequence is access and cost. A team that cannot reach a frontier API cannot run the judge, the critic or the curator, and a team that can is subject to every quota, price change and policy decision the provider chooses to make. The privacy and cost sides of that dependence are taken up in the other wider-context subsections, but the main point belongs here, that the open weights of the agent do not by themselves make the method open, because the part that produces the supervision is closed.

The deeper problem is one of framing. The contribution of this thesis is a label-free quality signal, a way to judge a deployment trace without a gold answer and without human labelling. Yet that label-free signal is itself produced by a strong closed model, so the method does not remove the need for expensive supervision, it only changes its form, from human labels gathered after the fact to frontier-model judgements gathered on demand. The validation in this thesis is that the judge agrees with task outcomes well above chance across domains, with the area under the curve reaching 0.94 on ToolHop (Table 5.3), but that agreement belongs to one particular closed model and only lasts as long as access to that model.

The result of this thesis is therefore portable as an idea and not yet portable as an implementation, because the label-free signal that makes it attractive is, today, only obtainable from a closed model that the user does not control.

6.3.4 Compute and sustainability

A working argument for small models is that they are cheaper and greener to run and to adapt. The agent in this thesis is a 9B model served on a single A100, and both adaptation tracks are parameter efficient, LoRA for the filtered fine-tuning and 4-bit QLoRA for the preference training. Nothing in the training pipeline needs a large training cluster. If a method could continually improve a model of this size on a domain, an operator could run a small specialised model in place of a large general one and spend far less energy per task. That is the direction worth pursuing on sustainability grounds, and it is the direction this work tried to follow on the agent side.

The improvement pipeline does not stay small, however. Every trace the agent produces in deployment is scored by a large closed judge, and the same model family supplies the rubric verdict, the step-level critic and the corrective suffix used to build preference pairs. The judge cost grows with the number of deployment traces, not with the size of the agent, so the energy saved by running a small agent can be spent again on judging that agent's output. The pipeline moves compute from a small open model to a large closed one rather than removing it. This trade would be easy to accept if the downstream gains were large, but they are not. The memory bank moved GPT-5.4-mini retail by $+6.1$ pp and airline by $+4.0$ pp (Table 5.9), and the other tracks land within their noise floors, so the saved agent compute buys judging that returns a small or null effect on the open model the saving was meant to serve.

The second cost is access, not energy. The one place memory rules produced a clear lift needed the stronger GPT-5.4-mini, where the rule bank moved retail by $+6.1$ pp and airline by $+4.0$ pp (Table 5.9), while the open 9B model left its retail baseline unchanged at 0.0 pp on the same pipeline. The attribution audit shows why the gap is real and not noise. On GPT-5.4-mini seventeen recovered retail tasks are attributable to a rule against three regressions, whereas on the 9B model no recovery is attributable to a rule and three regressions are, because it fired rules on the wrong requests (Table 8.6). The ability to act on self-curated rules sat with the larger model. If only strong models, which are often closed, can turn their own traces into better behaviour, then self-improvement widens the existing gap between opera-

tors who can afford those models and operators who cannot. The cheaper and greener option is also the one that cannot yet help itself.

The two costs point at the same target. The worthwhile goal is self-improvement that is both sustainable and accessible, meaning a small open model that can learn from its own deployment traces without a large closed model in the loop to judge them and without a large closed model being the only thing able to act on what is learned. This thesis does not reach that goal, but it does point to the obstacles, the judge cost that grows with trace volume and the instruction-following the open agent lacked. Closing both is what would make small-model self-improvement the genuinely greener and more democratic option it is meant to be.

7 Conclusion

This thesis asked whether the execution traces that a small open-source agent produces during deployment can be reused to improve that agent without gold labels, using only quality judgments from a larger model applied to the agent's own behaviour. To answer this, we built a blind, multi-dimensional, rubric-based judge, validated it against ground truth, and then used its signal to drive three trace-reuse pipelines: filtered supervised fine-tuning, memory rule curation, and replay-based preference learning. The agent was a Qwen3.5-9B ReAct model, the judge was a frontier model used without ever seeing the reference answer, and the pipelines were evaluated across structurally different multi-hop tool benchmarks, namely HotPotQA, τ^2-bench, and StableToolBench, with ToolHop as an additional validation domain.

The central finding has two sides. The blind judge produces a usable, no golden label quality signal, and the strength of that signal is bounded by how far a benchmark's failures are visible in the trace it reads. Converting that signal into reliable downstream task success proved harder, and the thesis is as much a diagnosis of why that conversion is difficult on a small agent as it is a demonstration of the signal itself.

7.1 Research Questions

RQ1: To what extent can a blind, multi-dimensional, rubric-based LLM judge reliably classify the quality of agentic traces without access to gold answers, and does the resulting signal transferacross structurally different multi-hop tool benchmarks?

The blind rubric judge is a usable signal whose reliability is bounded by the benchmark that uses it and what the trace exposes. The graded process-quality score is consistent throughout the benchmarks used, with the increase in score the ground truth accuracy increases. However, the signal's strength differs throughout the benchmarks tested. On ToolHop, every rubric dimension separates passed from failed traces by a wide margin, however, the rubric dimensions become weaker on τ^2-bench retail, where database state is required for ground truth. The filtered-SFT case study in the discussions shows the consequence. We can therefore conclude that the judge as a intermediate for success depends on what the trace exposes and how the rubric is calibrated.

The strength of the signal, however, had high variance between benchmarks. It was strongest where the trace itself reveals how the agent worked, and weakest where success depends on something the trace does not show. On ToolHop, every rubric dimension separated correct from incorrec traces by a wide margin (AUC 0.94). On τ^2-bench retail the separation nearly vanished (AUC 0.69), because success there is decided by a final database state and by policy compliance that a trace-only judge cannot see, and no rubric dimension separated passed from failed traces by even one point. The domain gap is not a measurement quirk, but a structural boundary of any trace-only judge. We therefore conclude that the judge works as an intermediate signal for success only to the degree that a benchmark's failures are written into the trace, and that the rubric is calibrated to the domain. The signal transfers in shape, since the score always moves in the right direction, but not in strength.

RQ2: To what extent can filtered supervised fine-tuning, using the blind judge's quality signal, improve agent performance on multi-hop tool benchmarks?

The results were mixed and, where the benchmark had headroom, negative. Filtered supervised fine-tuning captured the rubric's signal but did not convert it into task success on either benchmark. On HotPotQA the rubric scores rose from baseline to the strict-blind filter across all four dimensions, with reasoning coherence improving the most (+0.19), and the filtered model ranked first or tied first on every task metric, yet the spread across all conditions stayed within roughly one point of the baseline. The model learned the surface features that the rubric rewards without translating them into higher accuracy.

On τ^2-bench retail the picture was sharper and went the wrong way. Both filters lowered task success, and the cleaner-by-construction oracle filter, which keeps only traces verified correct by the gold evaluator, regressed further (7.9 points on pass1) than the judge-filtered strict-blind condition (4.4 points). Filter purity therefore did not predict downstream performance, which points to the fine-tuning data rather than the filter as the limiting factor. The most likely reasons are that the filtered training sets were small, that traces drawn from the same model already lie within its own distribution and so carry little that is new to learn, and that the positive learning signal from an already-successful trace is weak. τ^2-bench retail run-to-run noise and the limited training-set size mean a small positive effect cannot be fully ruled out, but on the data available the conclusion is that filtered supervised fine-tuning does not improve agent task success on these benchmarks.

RQ3:How can memory-based curation through in-context rule injection adapt an agent to a new domain?

In-context rule injection adapts an agent to a new domain when the agent is strong enough to follow the injected rules and the domain's failures are procedural, but not otherwise. The effect divided cleanly along the agent. On τ^2-bench the rule bank lifted the GPT-5.4-mini deployments (+6.1 percentage on retail and +4.0 on airline) while leaving the Qwen3.5-9B deployment unchanged. The reason for the Qwen result is instruction-following rather than the rules themselves: the smaller agent fired rules on the wrong requests, for example applying a delivered-order rule to a pending-order request, so the bank held correct guidance that the agent could not apply reliably. Where the rules did work, the lift was concentrated rather than spread out. Of the recovered GPT-5.4-mini retail tasks, seventeen followed a single mechanism, looking up the customer's orders once authentication succeeded, a step the baseline often skipped.

On HotPotQA neither agent showed a lift beyond noise, and the reason is different. HotPotQA failures come from research depth and reasoning, not from calling the wrong tool in the wrong order, so there is no procedural failure mode for a rule to target. The gains that did appear were in mean task success rather than in reliability, since the pass3 consistency barely moved, and the approach carries open problems around rule-set scalability and per-task re-

trieval, because the bank is injected wholesale on every task with no filter for relevance. To answer the research question, in-context rule injection yields a net increase on procedural tool-use failures for an agent able to act on the rules, but not on general capabilities such as the reasoning that HotPotQA demands.

RQ4: To what extent can replay-based preference learning, using corrective interventions guided by the judge, improve an agent's performance on multi-hop tool benchmarks?

Replay-based preference learning helped the agent only on the simplest tasks, and the gain did not generalise. Training on corrective preference pairs reconstructed from the agent's own failed StableToolBench traces, where each pair shares a prefix and differs only after a single judge-identified bad step, raised the in-distribution Solvable Pass Rate. The improvement was concentrated on the single-tool G1 subsets (pooled +5.3 percentage), faded to flat on the multi-tool G2 subsets (+0.2 percentage), and reversed on the held-out G3 subset (2.5 percentage). This ordering follows directly from the training signal: because each pair differs after one bad step, the update teaches the agent to recover from a single local mistake rather than to plan a longer task, so the benefit shrinks as tasks demand more tools and longer horizons.

Neither cross-benchmark evaluation showed measurable transfer. On HotPotQA the trained and baseline models tied exactly at 67.1% on the hard split, and on τ^2-bench retail an apparent single-rollout gain collapsed under replication, with three rollouts placing the trained model within 0.7 percentage of the baseline. The answer to RQ4 is therefore narrow but precise: as constructed, replay-based preference learning yields one single-tool, in-distribution gain and no transferable improvement.

7.2 Future Work

The work points to several directions, we believe that the adapting a LLM based agent to a new domain is a new field of research and therefore there are several things that could be worked on from the thesis.

7.2.1 A purpose built domain-adaptation benchmark

No benchmark used in this thesis is sufficient on its own. HotPotQA is large but only uses two tools and rewards reasoning. τ^2-bench retail and airline use the right kind of tasks but ship too few of them, and the measured ± 5 percentage run-to-run noise at $n = 3$ swallows any small effect. Telecom is at the practical ceiling, and ToolHop assigns a different tool set per task, so it measures one-shot tool use rather than adaptation within a domain. A purpose-built benchmark would combine three properties: enough tasks to push the noise floor below the effect sizes of interest, a shared tool set and domain across all tasks so curated rules can transfer to held-out ones, and tasks that require multi-step, multi-tool reasoning. A reliable evaluator would also be needed.

7.2.2 Judge bias

The judge is the load bearing component of the pipeline, and two related risks deserve direct study. The first is the confidence cascade discussed, a confident sounding agent can produce traces that the blind judge reads as more competent, which raises both the process-quality score and the judge's self reported confidence, so confidence cannot cleanly separate the judge's correct verdicts from its incorrect ones. The second is that the same model family supplied both the judge and, in part of the work, the agent, so an evaluation that also uses that family cannot fully separate genuine task improvement from agreement with the judges

prefrences. Future work should measure the judge against a held out, independent evaluator and quantify how much of any reported fain is real competence rather tahn judge agreement.

7.2.3 Scale

Filtered fine-tuning used training ests in the hundreds of traces, where LoRA may need far more, and the preference and memoty loops were each run once rather than iterated. Running the training loops at a larger scale, with more collected traces, more iterations of the rejected-sample-and-retrain cycle, and enough repreated rollouts to drive the noise floor down, would test whether the small and small and sometimes inconsistent effects seen here grow into stable ones.

7.2.4 Methods

Each track has a clear next step. For memory curation, a per-task retrieval step that injects only the rules whose trigger matches the current task, paired with an evaluation step that can roll back a rule change that regresses, would address the two design choices that most limited the Qwen results. For replay-based preference learning, extending the corrective signal beyond a single bad step towards multi step or full trajectory preferences would test whether the method can teach planning rather than only local recovery. Finally, the three pipelines were studied in isolation. Combining them, for example by curating memory rules on top of a fine-tuned or prefrence-trained agent, is a natural direction once each component is better understood on its own.

7.3 Concluding remarks

The trace that an agent discards at the end of a session does contain a reusable signal, and a blind, larger model can read that signal from the trace alone, without the gold answer. Turning the signal into reliable, transferable gains on a small open-source agent is the harder half of the problem, and here the results are honest rather than overstated. The gains that appeared were narrow, and bounded by data size, benchmark noise, and the agent's own ability to act on what the signal recommends.

These are limits of the current setup, not of the idea. As agents are deployed at ever larger scale, the traces they generate are a free and growing resource, and learning to reuse them without labels remains a worthwhile and, on the evidence here, an achievable goal. The contribution of this thesis is to show that the signal is real, to map where it is strong and where it is weak, and how that signal could be used is something that will need continued work.

8 Supplementary results

This appendix collects the supporting tables referenced from the Results chapter (Chapter 5). They give the per-class and per-run breakdowns behind the headline tables, the confusion of the blind judge, its calibration on τ^2-bench and ToolHop, the retail outcome transitions and failure budget for filtered fine-tuning, and the retail rule-attribution counts for memory curation.

8.1 Judge validation

Table 8.1 gives the confusion behind the blind judge verdicts of Section 5.1, and Tables 8.2 and 8.3 the per-band calibration of the process-quality score on τ^2-bench and ToolHop.

Table 8.1: Confusion of the blind judge on the 493 HotPotQA traces, for both prompts. The judge accepts about 11% to 12% wrong traces (FP among PASS), the precision ceiling that matters for filtering, since a false positive enters the training data.

	PASS		FAIL	
Prompt	Correct (TP)	Wrong (FP)	Correct (FN)	Wrong (TN)
Binary	266	36	57	134
Rubric	268	38	57	130

Table 8.2: Blind judge calibration on the τ^2-bench 114-task baseline traces. AUC(PQ) is how well PQ ranks passed above failed tasks, and the three rightmost columns give the official pass rate in each PQ band. PQ tracks success on both splits and more strongly on telecom (0.79) than retail (0.69).

Split	n	Baseline pass	AUC(PQ) ↑	PQ <4	$4 \leqslant$ PQ < 4.5	PQ $\geqslant 4.5$
Retail	114	73.8%	0.69	27.3%	56.3%	84.2%
Telecom	114	60.0%	**0.79**	37.7%	80.0%	92.6%

Table 8.3: Blind judge on ToolHop, 995 tasks, against answer correctness. PQ separates correct from incorrect traces most sharply of any benchmark (AUC 0.94), with the correct rate rising from 16.2% below PQ 4 to 91.1% at PQ $\geqslant$ 4.5.

Benchmark	n	Baseline pass	AUC(PQ) ↑	PQ $<$ 4	4 $\leqslant$ PQ $<$ 4.5	PQ $\geqslant$ 4.5
ToolHop	995	50.5%	**0.94**	16.2%	82.2%	91.1%

8.2 Track 1: filtered supervised fine-tuning

Tables 8.4 and 8.5 break down the retail regression of Section 5.2 by per-task outcome transition and by failure mode.

Table 8.4: Retail outcome transitions, Baseline against each SFT condition over the 114 tasks, per-task outcome by majority of three rollouts. Oracle is the larger net regression (25 broken, 7 recovered), Strict blind the more churned (17 broken, 11 recovered).

Outcome	vs Oracle	vs Strict blind
Passed both runs	62	70
Passed, then failed	25	17
Failed, then recovered	7	11
Failed both runs	20	16

Table 8.5: Retail failure budget by mode, pooled over $n = 3$ trials (342 trials per condition). No action means zero state-mutating tool calls, wrong action means a state-mutating call that failed the evaluator. The added failures under both filters are mostly wrong actions.

Condition	Total trials	Total failures ↓	No action ↓	Wrong action ↓
Baseline	342	**96**	**13**	**83**
Oracle	342	123	21	102
Strict blind	342	111	20	91

8.3 Track 2: memory rule curation

Table 8.6 gives the per-task rule-attribution counts behind the retail outcome changes of Section 5.3.

Table 8.6: Retail outcome changes attributable to an injected rule, by agent. A recovery is a task Baseline failed and the rule run solved by following a rule, a regression the reverse. The lift is entirely on gpt-5.4-mini, while every Qwen3.5-9B change attributable to a rule is a regression.

Agent	Recoveries	Regressions
Qwen3.5-9B	0	3
gpt-5.4-mini	17	3

Bibliography

[1] Lakshya A Agrawal, Shangyin Tan, Dilara Soylu, Noah Ziems, Rishi Khare, Krista Opsahl-Ong, Arnav Singhvi, Herumb Shandilya, Michael J Ryan, Meng Jiang, Christopher Potts, Koushik Sen, Alex Dimakis, Ion Stoica, Dan Klein, Matei Zaharia, and Omar Khattab. "GEPA: Reflective Prompt Evolution Can Outperform Reinforcement Learning". In: *The Fourteenth International Conference on Learning Representations*. 2026. URL: https://openreview.net/forum?id=RQm2KQTM5r.

[2] Marcin Andrychowicz, Filip Wolski, Alex Ray, Jonas Schneider, Rachel Fong, Peter Welinder, Bob McGrew, Josh Tobin, Pieter Abbeel, and Wojciech Zaremba. *Hindsight Experience Replay*. 2018. arXiv: 1707.01495 [cs.LG]. URL: https://arxiv.org/abs/1707.01495.

[3] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel Ziegler, Jeffrey Wu, Clemens Winter, Chris Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. "Language Models are Few-Shot Learners". In: *Advances in Neural Information Processing Systems*. Ed. by H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin. Vol. 33. Curran Associates, Inc., 2020, pp. 1877–1901. URL: https://proceedings.neurips.cc/paper_files/paper/2020/file/1457c0d6bfcb4967418bfb8ac142f64a-Paper.pdf.

[4] Baian Chen, Chang Shu, Ehsan Shareghi, Nigel Collier, Karthik R Narasimhan, and Shunyu Yao. *FireAct: Toward Language Agent Finetuning*. 2024. URL: https://openreview.net/forum?id=RqUMWdDg52.

[5] Prateek Chhikara, Dev Khant, Saket Aryan, Taranjeet Singh, and Deshraj Yadav. *Mem0: Building Production-Ready AI Agents with Scalable Long-Term Memory*. 2025. arXiv: 2504.19413 [cs.CL]. URL: https://arxiv.org/abs/2504.19413.

[6] Tim Dettmers, Artidoro Pagnoni, Ari Holtzman, and Luke Zettlemoyer. "QLoRA: Efficient Finetuning of Quantized LLMs". In: *Thirty-seventh Conference on Neural Information Processing Systems*. 2023. URL: https://openreview.net/forum?id=OUIFPHEgJU.

[7] Alexander R. Fabbri, Wojciech Kryściński, Bryan McCann, Caiming Xiong, Richard Socher, and Dragomir Radev. "SummEval: Re-evaluating Summarization Evaluation". In: *Transactions of the Association for Computational Linguistics* 9 (2021). Ed. by Brian

Roark and Ani Nenkova, pp. 391–409. DOI: 10.1162/tacl_a_00373. URL: https://aclanthology.org/2021.tacl-1.24/.

[8] Zhicheng Guo, Sijie Cheng, Hao Wang, Shihao Liang, Yujia Qin, Peng Li, Zhiyuan Liu, Maosong Sun, and Yang Liu. "StableToolBench: Towards Stable Large-Scale Benchmarking on Tool Learning of Large Language Models". In: *Findings of the Association for Computational Linguistics: ACL 2024*. Ed. by Lun-Wei Ku, Andre Martins, and Vivek Srikumar. Bangkok, Thailand: Association for Computational Linguistics, Aug. 2024, pp. 11143–11156. DOI: 10.18653/v1/2024.findings-acl.664. URL: https://aclanthology.org/2024.findings-acl.664/.

[9] Zhicheng Guo, Sijie Cheng, Hao Wang, Shihao Liang, Yujia Qin, Peng Li, Zhiyuan Liu, Maosong Sun, and Yang Liu. *StableToolBench: Towards Stable Large-Scale Benchmarking on Tool Learning of Large Language Models*. 2024. arXiv: 2403.07714 [cs.CL].

[10] Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. *LoRA: Low-Rank Adaptation of Large Language Models*. 2022. arXiv: 2106.09685 [cs.CL].

[11] Hunter Lightman, Vineet Kosaraju, Yuri Burda, Harrison Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. "Let's Verify Step by Step". In: *The Twelfth International Conference on Learning Representations*. 2024. URL: https://openreview.net/forum?id=v8L0pN6EOi.

[12] Hunter Lightman, Vineet Kosaraju, Yuri Burda, Harrison Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. "Let's Verify Step by Step". In: *The Twelfth International Conference on Learning Representations*. 2024.

[13] Long Ouyang, Jeff Wu, Xu Jiang, Diogo Almeida, Carroll L. Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, John Schulman, Jacob Hilton, Fraser Kelton, Luke Miller, Maddie Simens, Amanda Askell, Peter Welinder, Paul Christiano, Jan Leike, and Ryan Lowe. *Training language models to follow instructions with human feedback*. 2022. arXiv: 2203.02155 [cs.CL]. URL: https://arxiv.org/abs/2203.02155.

[14] Siru Ouyang, Jun Yan, I-Hung Hsu, Yanfei Chen, Ke Jiang, Zifeng Wang, Rujun Han, Long Le, Samira Daruki, Xiangru Tang, Vishy Tirumalashetty, George Lee, Mahsan Rofouei, Hangfei Lin, Jiawei Han, Chen-Yu Lee, and Tomas Pfister. "ReasoningBank: Scaling Agent Self-Evolving with Reasoning Memory". In: *The Fourteenth International Conference on Learning Representations*. 2026. URL: https://openreview.net/forum?id=jL7fwchScm.

[15] Joon Sung Park, Joseph C. O'Brien, Carrie J. Cai, Meredith Ringel Morris, Percy Liang, and Michael S. Bernstein. *Generative Agents: Interactive Simulacra of Human Behavior*. 2023. arXiv: 2304.03442 [cs.HC]. URL: https://arxiv.org/abs/2304.03442.

[16] Yujia Qin, Shihao Liang, Yining Ye, Kunlun Zhu, Lan Yan, Yaxi Lu, Yankai Lin, Xin Cong, Xiangru Tang, Bill Qian, Sihan Zhao, Lauren Hong, Runchu Tian, Ruobing Xie, Jie Zhou, Mark Gerstein, Dahai Li, Zhiyuan Liu, and Maosong Sun. *ToolLLM: Facilitating Large Language Models to Master 16000+ Real-world APIs*. 2023. arXiv: 2307.16789 [cs.AI]. URL: https://arxiv.org/abs/2307.16789.

[17] Rafael Rafailov, Archit Sharma, Eric Mitchell, Stefano Ermon, Christopher D. Manning, and Chelsea Finn. *Direct Preference Optimization: Your Language Model is Secretly a Reward Model*. 2024. arXiv: 2305.18290 [cs.LG]. URL: https://arxiv.org/abs/2305.18290.

[18] Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, Y. K. Li, Y. Wu, and Daya Guo. *DeepSeekMath: Pushing the Limits of Mathematical Reasoning in Open Language Models*. 2024. arXiv: 2402.03300 [cs.CL]. URL: https://arxiv.org/abs/2402.03300.

[19] Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. "Reflexion: language agents with verbal reinforcement learning". In: *Proceedings of the 37th International Conference on Neural Information Processing Systems*. NIPS '23. New Orleans, LA, USA: Curran Associates Inc., 2023.

[20] Avi Singh, John D Co-Reyes, Rishabh Agarwal, Ankesh Anand, Piyush Patil, Xavier Garcia, Peter J Liu, James Harrison, Jaehoon Lee, Kelvin Xu, Aaron T Parisi, Abhishek Kumar, Alexander A Alemi, Alex Rizkowsky, Azade Nova, Ben Adlam, Bernd Bohnet, Gamaleldin Fathy Elsayed, Hanie Sedghi, Igor Mordatch, Isabelle Simpson, Izzeddin Gur, Jasper Snoek, Jeffrey Pennington, Jiri Hron, Kathleen Kenealy, Kevin Swersky, Kshiteej Mahajan, Laura A Culp, Lechao Xiao, Maxwell Bileschi, Noah Constant, Roman Novak, Rosanne Liu, Tris Warkentin, Yamini Bansal, Ethan Dyer, Behnam Neyshabur, Jascha Sohl-Dickstein, and Noah Fiedel. "Beyond Human Data: Scaling Self-Training for Problem-Solving with Language Models". In: *Transactions on Machine Learning Research* (2024). Expert Certification. ISSN: 2835-8856. URL: https://openreview.net/forum?id=lNAyUngGFK.

[21] Yifan Song, Da Yin, Xiang Yue, Jie Huang, Sujian Li, and Bill Yuchen Lin. "Trial and Error: Exploration-Based Trajectory Optimization of LLM Agents". In: *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Ed. by Lun-Wei Ku, Andre Martins, and Vivek Srikumar. Bangkok, Thailand: Association for Computational Linguistics, Aug. 2024, pp. 7584–7600. DOI: 10.18653/v1/2024.acl-long.409. URL: https://aclanthology.org/2024.acl-long.409/.

[22] Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, Dan Bikel, Lukas Blecher, Cristian Ferrer, Moya Chen, Guillem Cucurull, David Esiobu, Jude Fernandes, Jeremy Fu, Wenyin Fu, and Thomas Scialom. *Llama 2: Open Foundation and Fine-Tuned Chat Models*. July 2023. DOI: 10.48550/arXiv.2307.09288.

[23] Jonathan Uesato, Nate Kushman, Ramana Kumar, H. Francis Song, Noah Yamamoto Siegel, Lisa Wang, Antonia Creswell, Geoffrey Irving, and Irina Higgins. *Solving Math Word Problems with Process-based and Outcome-based Feedback*. 2023. URL: https://openreview.net/forum?id=MND1kmmNy0O.

[24] Unsloth AI. *Unsloth: Finetune LLMs 2x faster with 70% less memory*. https://github.com/unslothai/unsloth. GitHub repository, accessed 2026-05-30. 2024.

[25] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. "Attention is All you Need". In: *Advances in Neural Information Processing Systems*. Ed. by I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett. Vol. 30. Curran Associates, Inc., 2017. URL: https://proceedings.neurips.cc/paper_files/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf.

[26] 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*. 2023. arXiv: 2305.16291 [cs.AI]. URL: https://arxiv.org/abs/2305.16291.

[27] Leandro von Werra, Younes Belkada, Lewis Tunstall, Edward Beeching, Tristan Thrush, Nathan Lambert, Shengyi Huang, Kashif Rasul, and Quentin Gallouédec. *TRL: Transformer Reinforcement Learning*. https://github.com/huggingface/trl. 2020.

[28] Tong Wu, Chong Xiang, Jiachen T. Wang, G. Edward Suh, and Prateek Mittal. *Effectively Controlling Reasoning Models through Thinking Intervention*. 2025. arXiv: 2503.24370 [cs.LG]. URL: https://arxiv.org/abs/2503.24370.

[29] Weimin Xiong, Yifan Song, Xiutian Zhao, Wenhao Wu, Xun Wang, Ke Wang, Cheng Li, Wei Peng, and Sujian Li. "Watch Every Step! LLM Agent Learning via Iterative Step-level Process Refinement". In: *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*. Ed. by Yaser Al-Onaizan, Mohit Bansal, and Yun-Nung Chen. Miami, Florida, USA: Association for Computational Linguistics, Nov. 2024, pp. 1556–1572. DOI: 10.18653/v1/2024.emnlp-main.93. URL: https://aclanthology.org/2024.emnlp-main.93/.

[30] An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, et al. *Qwen3 Technical Report*. 2025. arXiv: 2505.09388 [cs.CL].

[31] An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, Chujie Zheng, Dayiheng Liu, Fan Zhou, Fei Huang, Feng Hu, Hao Ge, Haoran Wei, Huan Lin, Jialong Tang, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jing Zhou, Jingren Zhou, Junyang Lin, Kai Dang, Keqin Bao, Kexin Yang, Le Yu, Lianghao Deng, Mei Li, Mingfeng Xue, Mingze Li, Pei Zhang, Peng Wang, Qin Zhu, Rui Men, Ruize Gao, Shixuan Liu, Shuang Luo, Tianhao Li, Tianyi Tang, Wenbiao Yin, Xingzhang Ren, Xinyu Wang, Xinyu Zhang, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yinger Zhang, Yu Wan, Yuqiong Liu, Zekun Wang, Zeyu Cui, Zhenru Zhang, Zhipeng Zhou, and Zihan Qiu. *Qwen3 Technical Report*. 2025. arXiv: 2505.09388 [cs.CL]. URL: https://arxiv.org/abs/2505.09388.

[32] Zhilin Yang, Peng Qi, Saizheng Zhang, Yoshua Bengio, William W. Cohen, Ruslan Salakhutdinov, and Christopher D. Manning. "HotpotQA: A Dataset for Diverse, Explainable Multi-hop Question Answering". In: *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*. 2018, pp. 2369–2380. DOI: 10.18653/v1/D18-1259.

[33] Shunyu Yao, Noah Shinn, Pedram Razavi, and Karthik R Narasimhan. "{τ}-bench: A Benchmark for \underline{T}ool-\underline{A}gent-\underline{U}ser Interaction in Real-World Domains". In: *The Thirteenth International Conference on Learning Representations*. 2025. URL: https://openreview.net/forum?id=roNSXZpUDN.

[34] Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. *ReAct: Synergizing Reasoning and Acting in Language Models*. 2023. arXiv: 2210.03629 [cs.CL]. URL: https://arxiv.org/abs/2210.03629.

[35] Junjie Ye, Zhengyin Du, Xuesong Yao, Weijian Lin, Yufei Xu, Zehui Chen, Zaiyuan Wang, Sining Zhu, Zhiheng Xi, Siyu Yuan, Tao Gui, Qi Zhang, Xuanjing Huang, and Jiecao Chen. "ToolHop: A Query-Driven Benchmark for Evaluating Large Language Models in Multi-Hop Tool Use". In: *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Ed. by Wanxiang Che, Joyce Nabende, Ekaterina Shutova, and Mohammad Taher Pilehvar. Vienna, Austria: Association for Computational Linguistics, July 2025, pp. 2995–3021. ISBN: 979-8-89176-251-0. DOI: 10.18653/v1/2025.acl-long.150. URL: https://aclanthology.org/2025.acl-long.150/.

[36] Siyu Yuan, Zehui Chen, Zhiheng Xi, Junjie Ye, Zhengyin Du, and Jiecao Chen. *Agent-R: Training Language Model Agents to Reflect via Iterative Self-Training*. 2025. arXiv: 2501.11425 [cs.AI]. URL: https://arxiv.org/abs/2501.11425.

[37] Zheng Yuan, Hongyi Yuan, Chengpeng Li, Guanting Dong, Keming Lu, Chuanqi Tan, Chang Zhou, and Jingren Zhou. *Scaling Relationship on Learning Mathematical Reasoning with Large Language Models*. 2023. arXiv: 2308.01825 [cs.CL]. URL: https://arxiv.org/abs/2308.01825.

[38] Eric Zelikman, Yuhuai Wu, Jesse Mu, and Noah D. Goodman. *STaR: Bootstrapping Reasoning With Reasoning*. 2022. arXiv: 2203.14465 [cs.LG]. URL: https://arxiv.org/abs/2203.14465.

[39] Aohan Zeng, Mingdao Liu, Rui Lu, Bowen Wang, Xiao Liu, Dong Yuxiao, and Jie Tang. "AgentTuning: Enabling Generalized Agent Abilities for LLMs". In: Jan. 2024, pp. 3053–3077. DOI: 10.18653/v1/2024.findings-acl.181.

[40] Qizheng Zhang, Changran Hu, Shubhangi Upasani, Boyuan Ma, Fenglu Hong, Vamsidhar Kamanuru, Jay Rainton, Chen Wu, Mengmeng Ji, Hanchen Li, Urmish Thakker, James Zou, and Kunle Olukotun. *Agentic Context Engineering: Evolving Contexts for Self-Improving Language Models*. 2026. arXiv: 2510.04618 [cs.LG]. URL: https://arxiv.org/abs/2510.04618.

[41] Andrew Zhao, Daniel Huang, Quentin Xu, Matthieu Lin, Yong-Jin Liu, and Gao Huang. "ExpeL: LLM agents are experiential learners". In: *Proceedings of the Thirty-Eighth AAAI Conference on Artificial Intelligence and Thirty-Sixth Conference on Innovative Applications of Artificial Intelligence and Fourteenth Symposium on Educational Advances in Artificial Intelligence*. AAAI'24/IAAI'24/EAAI'24. AAAI Press, 2024. ISBN: 978-1-57735-887-9. DOI: 10.1609/aaai.v38i17.29936. URL: https://doi.org/10.1609/aaai.v38i17.29936.

[42] Andrew Zhao, Daniel Huang, Quentin Xu, Matthieu Lin, Yong-Jin Liu, and Gao Huang. "ExpeL: LLM agents are experiential learners". In: *Proceedings of the Thirty-Eighth AAAI Conference on Artificial Intelligence*. AAAI'24/IAAI'24/EAAI'24. AAAI Press, 2024. ISBN: 978-1-57735-887-9. DOI: 10.1609/aaai.v38i17.29936.

[43] Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric P. Xing, Hao Zhang, Joseph E. Gonzalez, and Ion Stoica. *Judging LLM-as-a-Judge with MT-Bench and Chatbot Arena*. 2023. arXiv: 2306.05685 [cs.CL].