What Is Prompt Learning?

Prompts, like models, should improve with feedback — not stay static.

Author: Aparna Dhinakaran | 11 min read | Aug 1, 2025
URL: https://medium.com/arize-ai/what-is-prompt-learning-b2019e8d4154

Co-Authored by: Jason Lopatecki, Co-founder and CEO; Priyan Jindal, AI Engineer; & Aman Khan, Group Product Manager.

Recently, we started experimenting with a new approach to optimizing LLM prompts that we're calling "Prompt Learning" (PL). Unlike traditional optimization methods that rely on numerical scores, PL uses natural language feedback to iteratively improve prompts. The roots of this approach are in the Voyager paper by Jim Fan's team at NVIDIA. It is also alluded to by Andrej Karpathy in several recent tweets, where he argues prompt-centric learning will be a key technique.

Despite these early inklings, to our knowledge no one has yet rigorously researched, characterized, and measured a full implementation of a reinforcement learning based approach to prompt tuning. That's exactly what we set out to do.

This implementation is inspired by an idea introduced in the original Voyager paper. The iterative prompting mechanism used in the original Voyager paper as the agent acquires and refines forms the basis for our prompt learning approach.

Here is the repo for this research those who want to dive in deeper or run their own experiments!

What Is Prompt Learning?

Distinct from prompt tuning and not chain-of-thought, **prompt learning** is an approach that leverages natural language evals and critiques (rather than just scalar metrics like previously done by the industry) to drive targeted prompt updates.

Prompt learning differs from MetaPrompt prompt optimization in a couple major ways.

First and foremost, the error term is in English and is not a score. The English error term allows for English feedback that is used directly to tune instructions. An explanation from an eval tells you exactly why the evaluation failed and prompt learning then adds instructions to help fix the problem to the system prompt. The English error term allows us to solve a set of problems that are unsolvable by current pure prompt optimization techniques.

Secondly, prompt learning is an online approach to manage your system instructions that is designed to be run continually against your prompt — tuning instructions back into the context. LLM-based systems can assist with context engineering your system instructions.

The English instructions in the prompt context allow for management of instructions, such as how to deal with competing instructions or expiring instructions or human review of instructions, all in English. In our prompt learning meta prompt we even allow keywords where it will only make edits to a specific instructions-based area of the prompt. In “weights” and “gradient”-based prompt optimization approaches, this is nearly impossible.

This implementation of prompt learning uses evaluations, explanations, and annotations on runs of an application to automatically improve your prompt.

The results are promising: prompt learning can make significant levels of improvements, with only one-tenth or one-hundredth the number of labeled examples.

Let’s dive into the mechanics of prompt learning and examine exactly why it’s working.

What Is Prompt Learning?

Prompt learning differs from MetaPrompt prompt optimization in a couple major ways.

First and foremost, the error term is in English and is not a score. The English error term allows for English feedback that is used directly to tune instructions. An explanation from an eval tells you exactly why the evaluation failed and prompt learning then adds instructions to help fix the problem to the system prompt. The English error term allows us to solve a set of problems that are unsolvable by current pure prompt optimization techniques.

Secondly, prompt learning is an online approach to manage your system instructions that is designed to be run continually against your prompt — tuning instructions back into the context. LLM-based systems can assist with context engineering your system instructions.

The English instructions in the prompt context allow for management of instructions, such as how to deal with competing instructions or expiring instructions or human review of instructions, all in English. In our prompt learning meta prompt we even allow keywords where it will only make edits to a specific instructions-based area of the prompt. In “weights” and “gradient”-based prompt optimization approaches, this is nearly impossible.

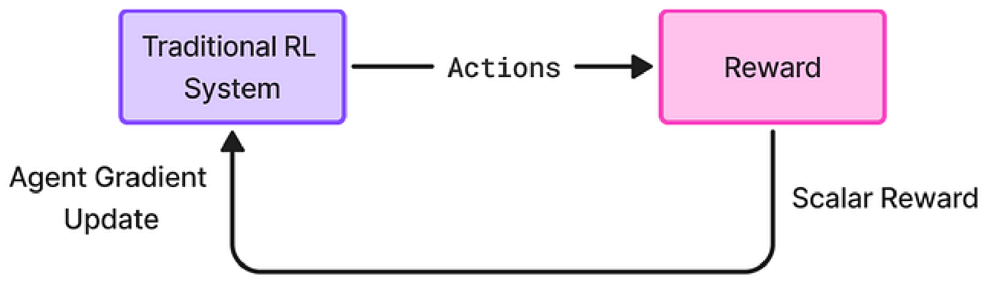
This implementation of prompt learning uses evaluations, explanations, and annotations on runs of an application to automatically improve your prompt.

The results are promising: prompt learning can make significant levels of improvements, with only one-tenth or one-hundredth the number of labeled examples.

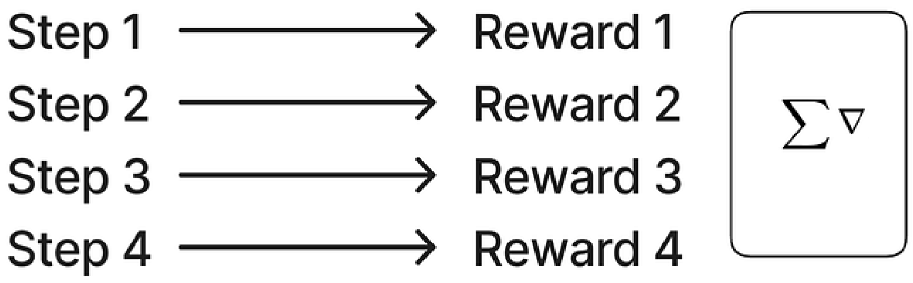
Let’s dive into the mechanics of prompt learning and examine exactly why it’s working.

What’s the Difference Between Reinforcement Learning and Prompt Learning?

Traditional reinforcement learning relies on using scores or errors to generate gradient error terms, which then update your original model. Each gradient error term pushes your model slightly closer to optimal performance.



The key here is that you need many, many examples to align your model. Over time, these myriad examples push your model towards outputting the correct values across your possible inputs. It works by accumulating error gradients and nudging your model in a certain direction.

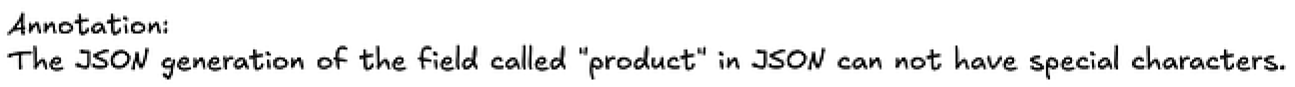


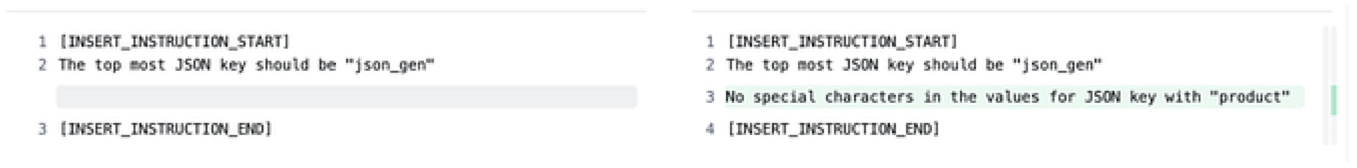
Reinforcement learning is a very powerful technique. But what if you don’t have thousands of examples? What if you have a complex set of goals and those goals don’t easily express as a score? Lastly, what if someone, an annotator or human expert, has relayed to you in English what the problem actually is and how to fix it?

Prompt learning allows you to make powerful changes using *individual* examples. Instead of gradient error terms calculated for each example, you calculate full text explanations of why an example was scored a certain way. These examples are then fed back into the optimization flow and incorporated into the prompt.

The key idea is:

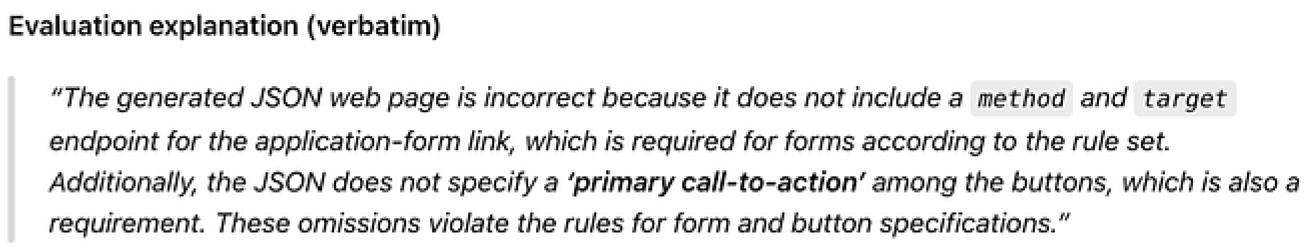
1. The “error”, an Eval explanation OR annotation term is in English
2. The modification that changes your actions are done in the prompt context, not weights
3. The reward function is an evaluation or human annotation
4. The instructions are maintained and managed in the prompt context, allowing instruction management


Annotation:
The JSON generation of the field called "product" in JSON can not have special characters.

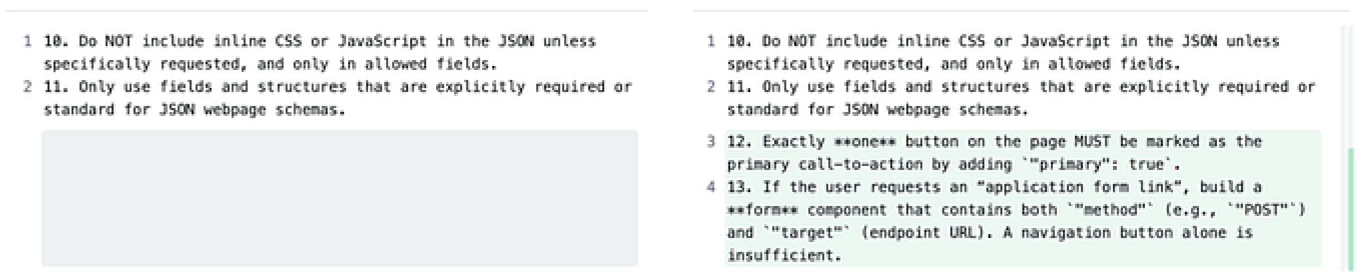

```
[INSERT_INSTRUCTION_START]
The top most JSON key should be "json_gen"

[INSERT_INSTRUCTION_END]
```

```
[INSERT_INSTRUCTION_START]
The top most JSON key should be "json_gen"
No special characters in the values for JSON key with "product"
[INSERT_INSTRUCTION_END]
```

Annotation Example: The above shows an example of a human annotation and a metaprompt added instruction.


Evaluation explanation (verbatim)

> *"The generated JSON web page is incorrect because it does not include a `method` and `target` endpoint for the application-form link, which is required for forms according to the rule set. Additionally, the JSON does not specify a* ***'primary call-to-action'*** *among the buttons, which is also a requirement. These omissions violate the rules for form and button specifications."*

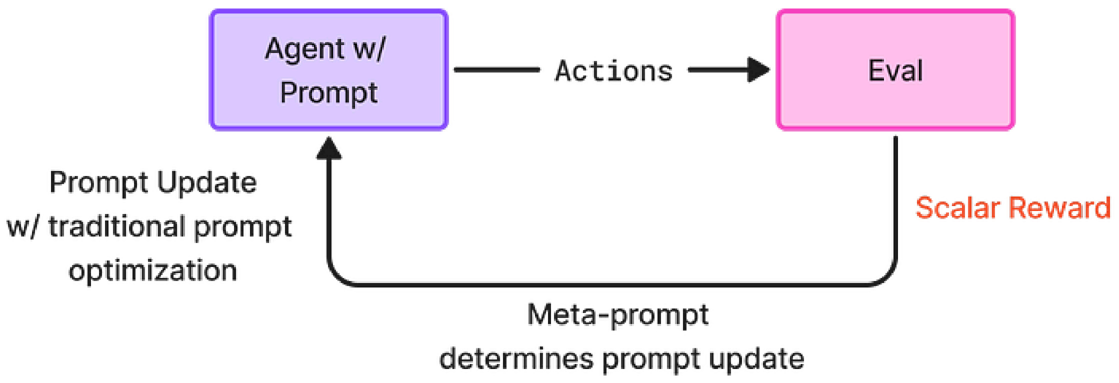

```
10. Do NOT include inline CSS or JavaScript in the JSON unless
specifically requested, and only in allowed fields.
11. Only use fields and structures that are explicitly required or
standard for JSON webpage schemas.
```

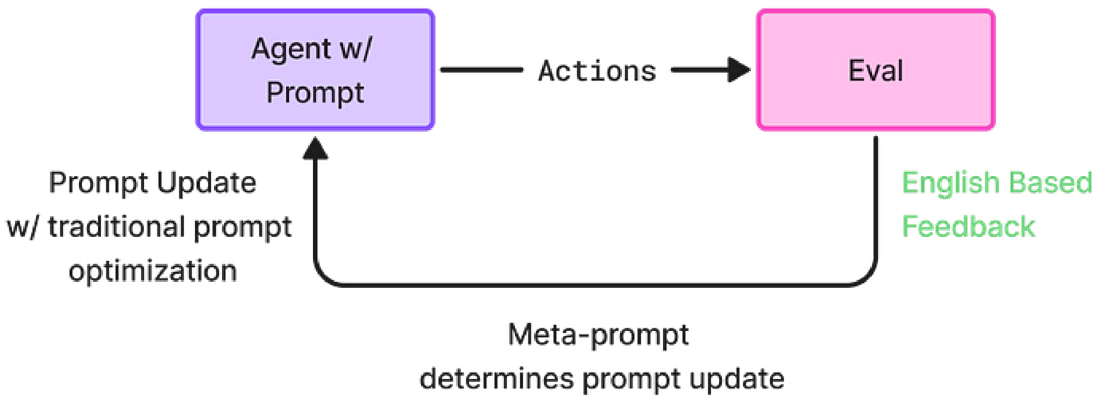
```
10. Do NOT include inline CSS or JavaScript in the JSON unless
specifically requested, and only in allowed fields.
11. Only use fields and structures that are explicitly required or
standard for JSON webpage schemas.
12. Exactly **one** button on the page MUST be marked as the
primary call-to-action by adding `"primary": true`.
13. If the user requests an "application form link", build a
**form** component that contains both `"method"` (e.g., `"POST"`)
and `"target"` (endpoint URL). A navigation button alone is
insufficient.
```

Evaluation Example: The above shows an example of an evaluation and a metaprompt created instruction to fix.

Our research data shows examples where well known optimization libraries fall short today. Namely, where evals with critiques or annotations contain information not available in the training set on how to fix a failure. There is not an easy way to take information-rich feedback in English and easily feed it back into a gradient update. In general you might not want to do gradient updates at all. Having all of your instructions in English allows you to deal with things that are not easy to do in "weight land," such as what to do with competing instructions, removal of instructions, compaction of instructions and managing when to expire an instruction — essentially what we call **instruction management.**

One other advantage of prompt learning over gradient based updates is instead of using tens of thousands of examples, you can make changes to your system prompt with a single annotation example.


Agent w/ Prompt
Actions
Eval
Scalar Reward
Prompt Update w/ traditional prompt optimization
Meta-prompt determines prompt update

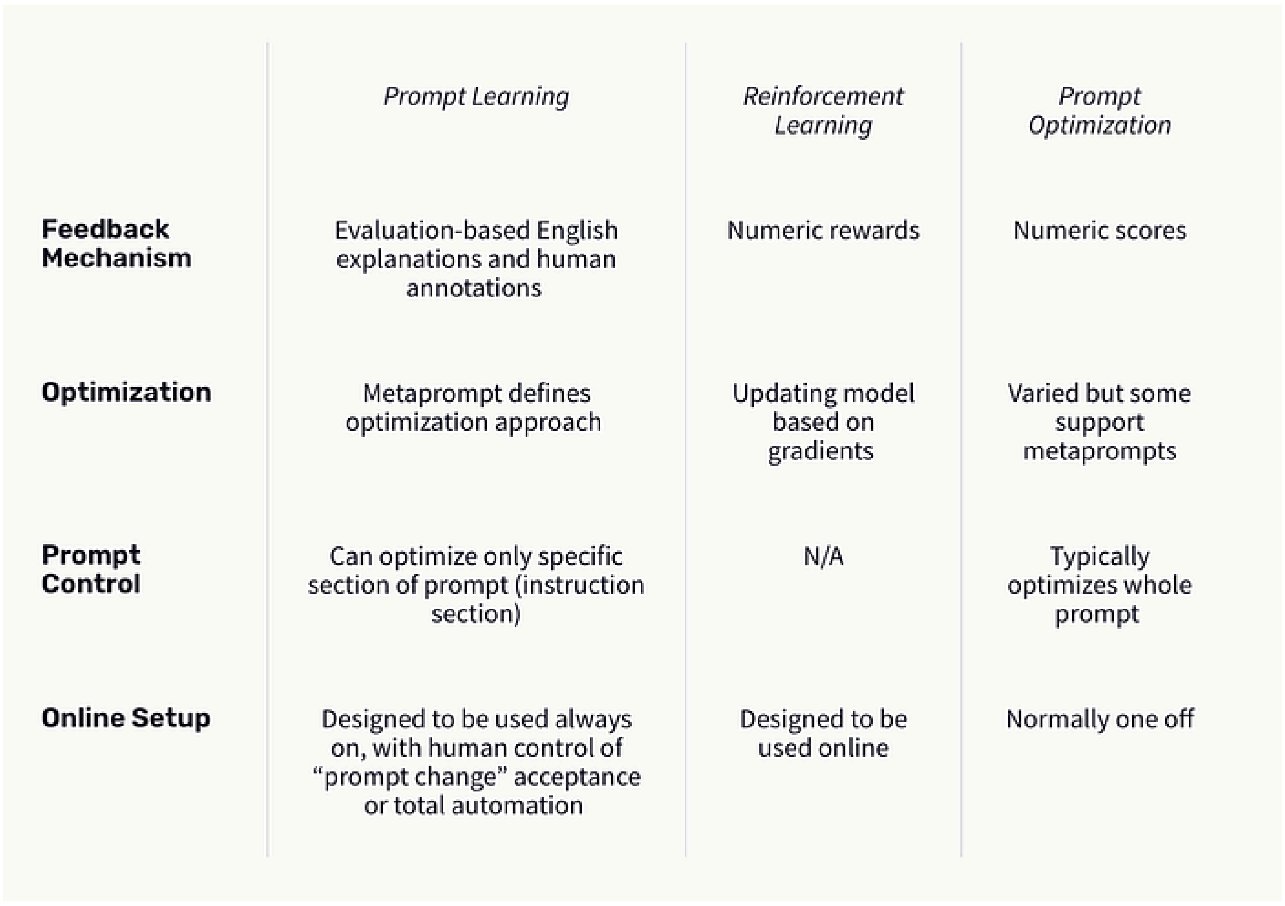

Agent w/ Prompt
Actions
Eval
English Based Feedback
Prompt Update w/ traditional prompt optimization
Meta-prompt determines prompt update

How Is This Different from Prompt Optimization?

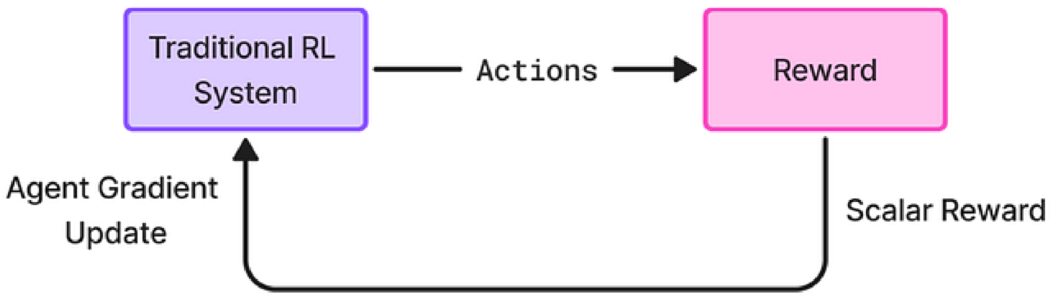
There are a lot of techniques out there for prompt optimization. Prompt optimization applies more traditional machine learning train and test approaches to optimizing prompts by gathering examples and attempting to find similarities with those examples.

The seed of the failure of all prompt optimization approaches comes from the focus on scores as the means of propagating failure errors. As you think about failures, not every failure expresses itself easily as a numeric number and a numeric value hides the reason for a failure.

Using a score as your main approach for propagating a failure disconnects the optimization fix from the reason it failed.

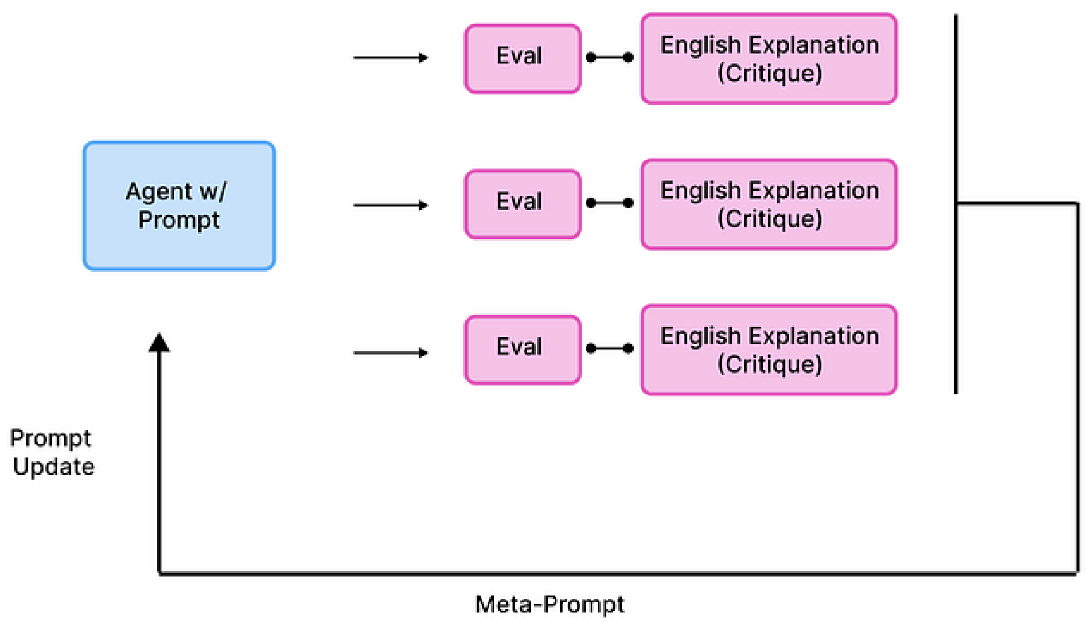


	Prompt Learning	*Reinforcement Learning*	*Prompt Optimization*
Feedback Mechanism	Evaluation-based English explanations and human annotations	Numeric rewards	Numeric scores
Optimization	Metaprompt defines optimization approach	Updating model based on gradients	Varied but some support metaprompts
Prompt Control	Can optimize only specific section of prompt (instruction section)	N/A	Typically optimizes whole prompt
Online Setup	Designed to be used always on, with human control of “prompt change” acceptance or total automation	Designed to be used online	Normally one off



How Does the Optimization Loop Work?

In many real world use cases, as we tested with customers on real data, a single optimization run with a single shot output worked great. In cases where you need multiple loops over the optimization to improve performance, the English explanation (or critique) output of an Evaluator can improve performance.


Agent w/ Prompt
Eval
English Explanation (Critique)
Eval
English Explanation (Critique)
Eval
English Explanation (Critique)
Prompt Update
Meta-Prompt

The English explanation (Critique) is an important feature of our evaluation library, generating an explanation then allows the results to be used in a feedback loop.

In our testing, as the model was required to add more instructions back into the context window to fix the prompt, the iterative loop became more important. In cases where only 1–10 instructions needed to be added a single meta-prompt improvement loop was sufficient.

How Did We Test Prompt Learning?

We ran a series of optimization experiments using prompt learning in order to benchmark its efficacy. To date, this has been run across a sizable production set of AI application and agent use cases:

For our demo data application, we chose a JSON generation problem where models had to generate JSON for a webpage based on natural language prompts.

We additionally generated a set of **latent rules** that the responses needed to follow. Things like:

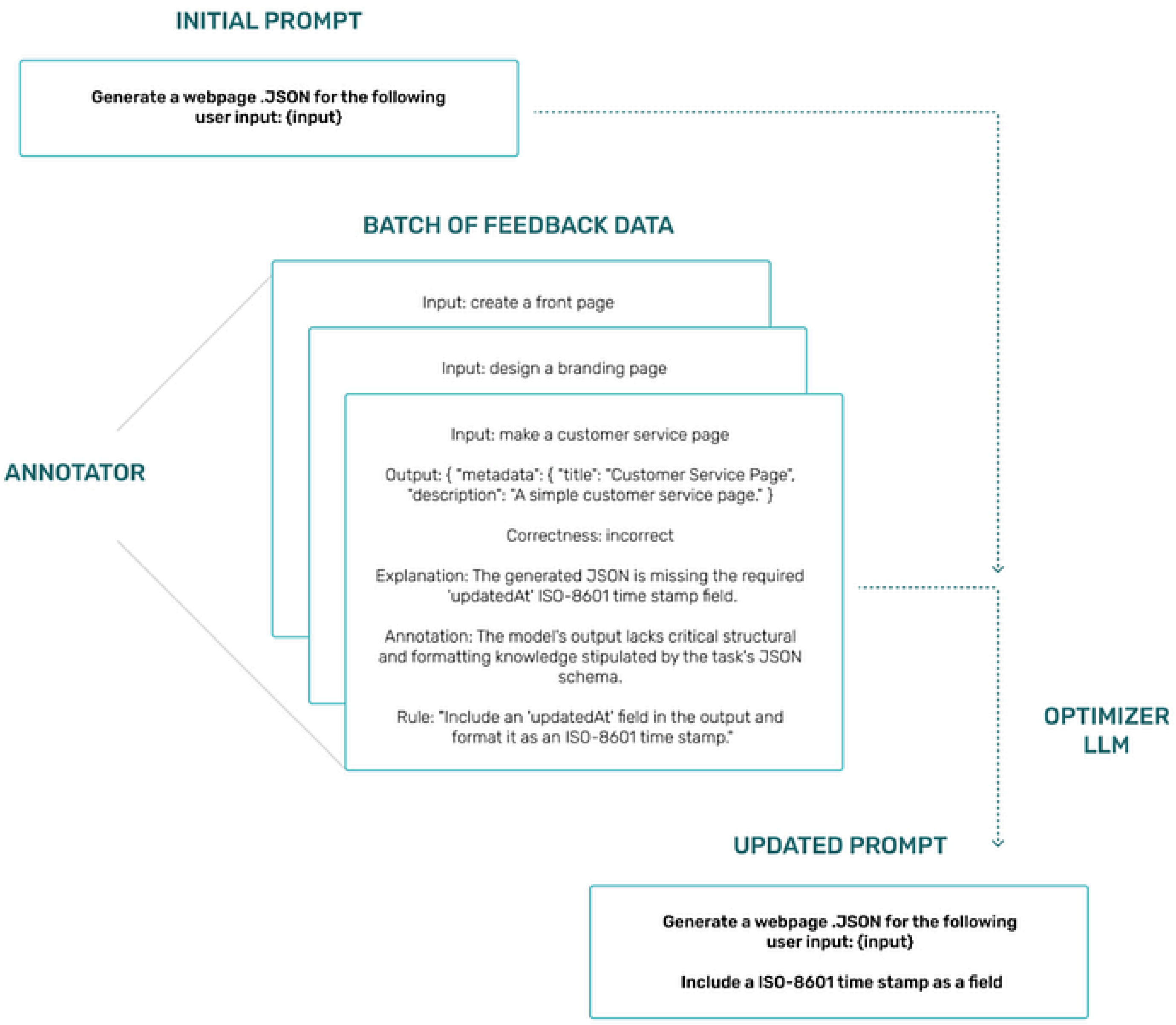
1. Every section needs a type value from a predefined list
2. All images must include alt text
3. All external asset links must use https

These rules were implicitly represented in feedback and explanations attached to a set of traces of our application.

We designed this test to mimic a typical evaluation cycle of an agent. Evaluation was done using a mixture of LLM-as-a-judge with human review, again to mimic real world patterns.

All of this data (the application traces, feedback, and explanations) was then fed into the optimization stage.

To perform the optimization itself, we used a modified version of meta-prompting that we later dubbed **prompt learning**.


INITIAL PROMPT
Generate a webpage .JSON for the following user input: (input)
BATCH OF FEEDBACK DATA
Input: create a front page
Input: design a branding page
Input: make a customer service page
Output: { "metadata": { "title": "Customer Service Page", "description": "A simple customer service page." }
Correctness: incorrect
Explanation: The generated JSON is missing the required 'updatedAt' ISO-8601 time stamp field.
Annotation: The model's output lacks critical structural and formatting knowledge stipulated by the task's JSON schema.
Rule: "Include an 'updatedAt' field in the output and format it as an ISO-8601 time stamp."
ANNOTATOR
OPTIMIZER LLM
UPDATED PROMPT
Generate a webpage .JSON for the following user input: (input)
Include a ISO-8601 time stamp as a field

How Does Prompt Learning Perform?

We tested prompt learning on real world customer deployments, internal synthetic data instruction learning tests, and well known benchmarks like Big Bench Hard.

Instruction Learning

The following are results from our rules based instruction learning data set. The dataset was designed to test tasks that are impossible to solve without learning from information provided from outside of the training data.

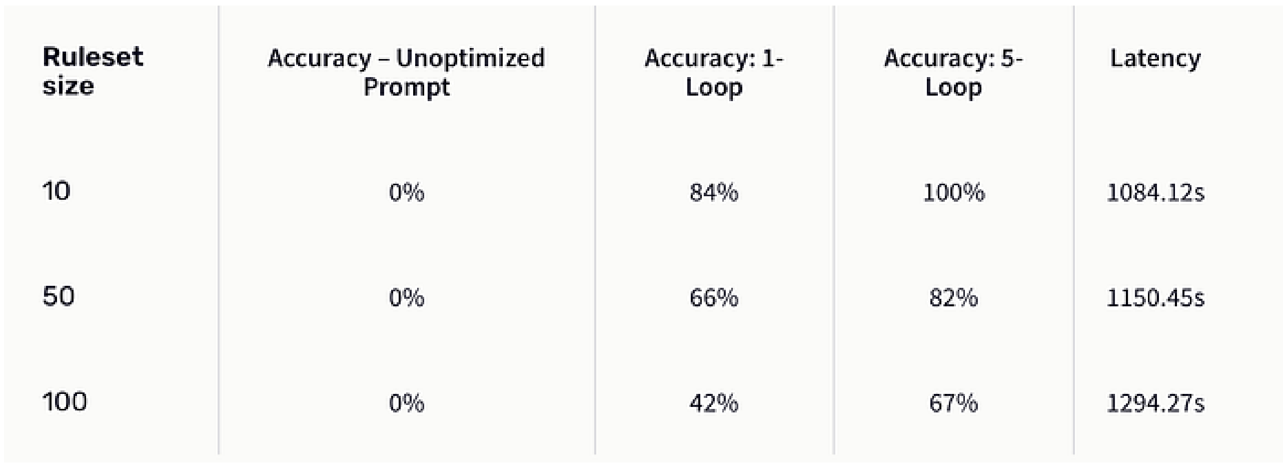
The task design is as follows:

- There is a JSON Generation where the JSON is designed to control the rendering of a website (real world customer use case)
- There are a number of business rules unknown to the LLM that it must follow
- There are annotators that annotate mistakes, where it did not follow the rule
- The Prompt Learning loop must pick up the rules and modify the prop to capture enough rules to pass the Evaluation
- We test with a number of rules and all rules must pass to get a “full pass”, a single rule fails the test

This test attempts to assess how well the system can pick up information from feedback and formalize them into a new system prompt. Most of these tests will have 0 for accuracy prior to optimization.

Feedback can be human driven, evaluation driven or real world feedback driven.

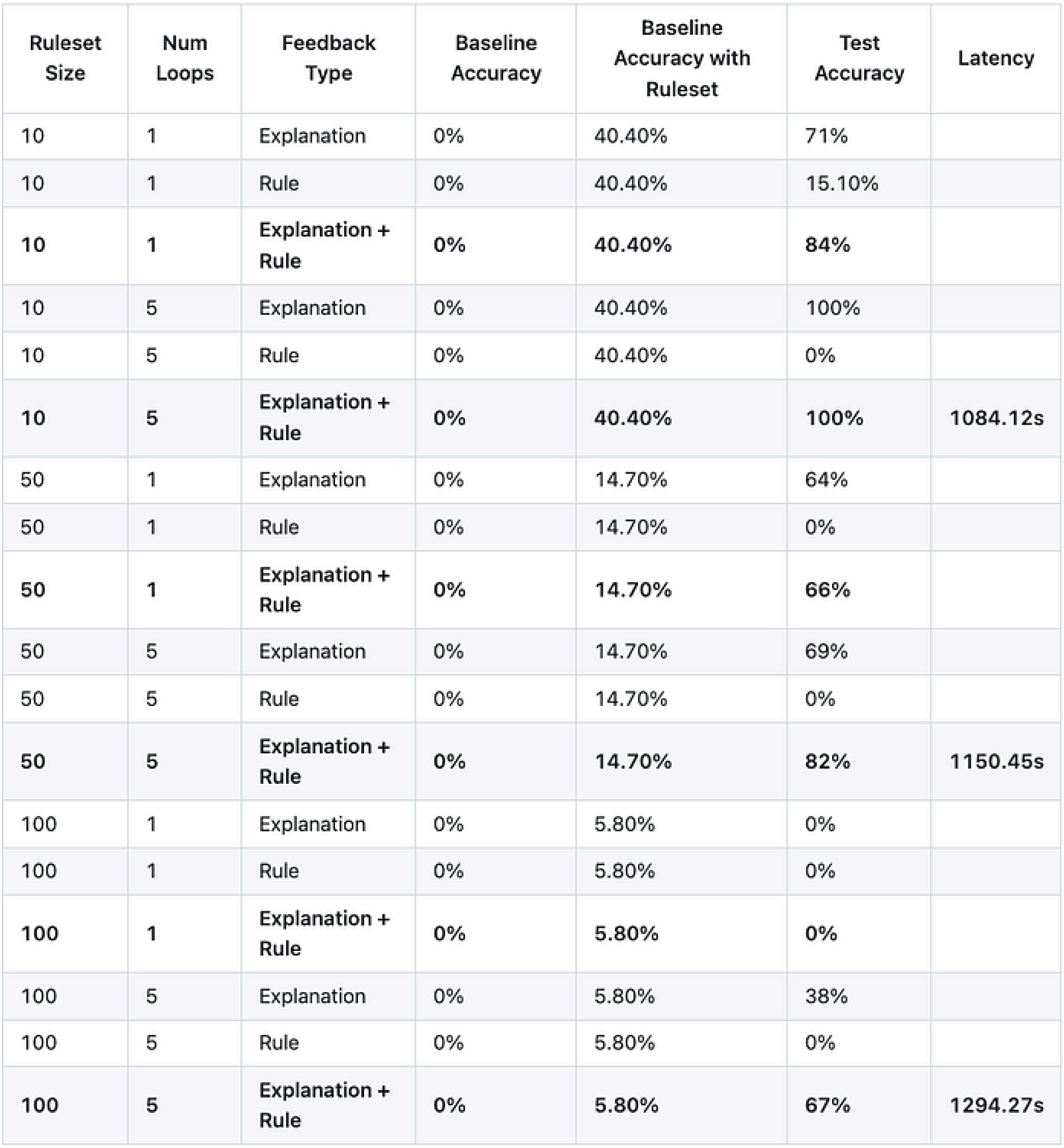
These rules based critiques also challenge most prompt optimization systems whose only feedback mechanism is a scalar error term. Almost all prompt optimization techniques fail on this test today.



Ruleset size	Accuracy – Unoptimized Prompt	Accuracy: 1-Loop	Accuracy: 5-Loop	Latency
10	0%	84%	100%	1084.12s
50	0%	66%	82%	1150.45s
100	0%	42%	67%	1294.27s

Table 1: Initial Results Rules Task

The above shows that as you grow rules the test gets harder, the prompt needs to learn 100s of rules to generate correct JSON. It takes multiple iterations to get the prompt right to pass the rules.



Ruleset Size	Num Loops	Feedback Type	Baseline Accuracy	Baseline Accuracy with Ruleset	Test Accuracy	Latency
10	1	Explanation	0%	40.40%	71%	
10	1	Rule	0%	40.40%	15.10%	
10	**1**	**Explanation + Rule**	**0%**	**40.40%**	**84%**	
10	5	Explanation	0%	40.40%	100%	
10	5	Rule	0%	40.40%	0%	
10	**5**	**Explanation + Rule**	**0%**	**40.40%**	**100%**	**1084.12s**
50	1	Explanation	0%	14.70%	64%	
50	1	Rule	0%	14.70%	0%	
50	**1**	**Explanation + Rule**	**0%**	**14.70%**	**66%**	
50	5	Explanation	0%	14.70%	69%	
50	5	Rule	0%	14.70%	0%	
50	**5**	**Explanation + Rule**	**0%**	**14.70%**	**82%**	**1150.45s**
100	1	Explanation	0%	5.80%	0%	
100	1	Rule	0%	5.80%	0%	
100	**1**	**Explanation + Rule**	**0%**	**5.80%**	**0%**	
100	5	Explanation	0%	5.80%	38%	
100	5	Rule	0%	5.80%	0%	
100	**5**	**Explanation + Rule**	**0%**	**5.80%**	**67%**	**1294.27s**

Tests of Prompt Learning

The rules determine how many instructions the AI Agent must adhere to in order to pass, all these rules need to be picked up through the feedback paths.

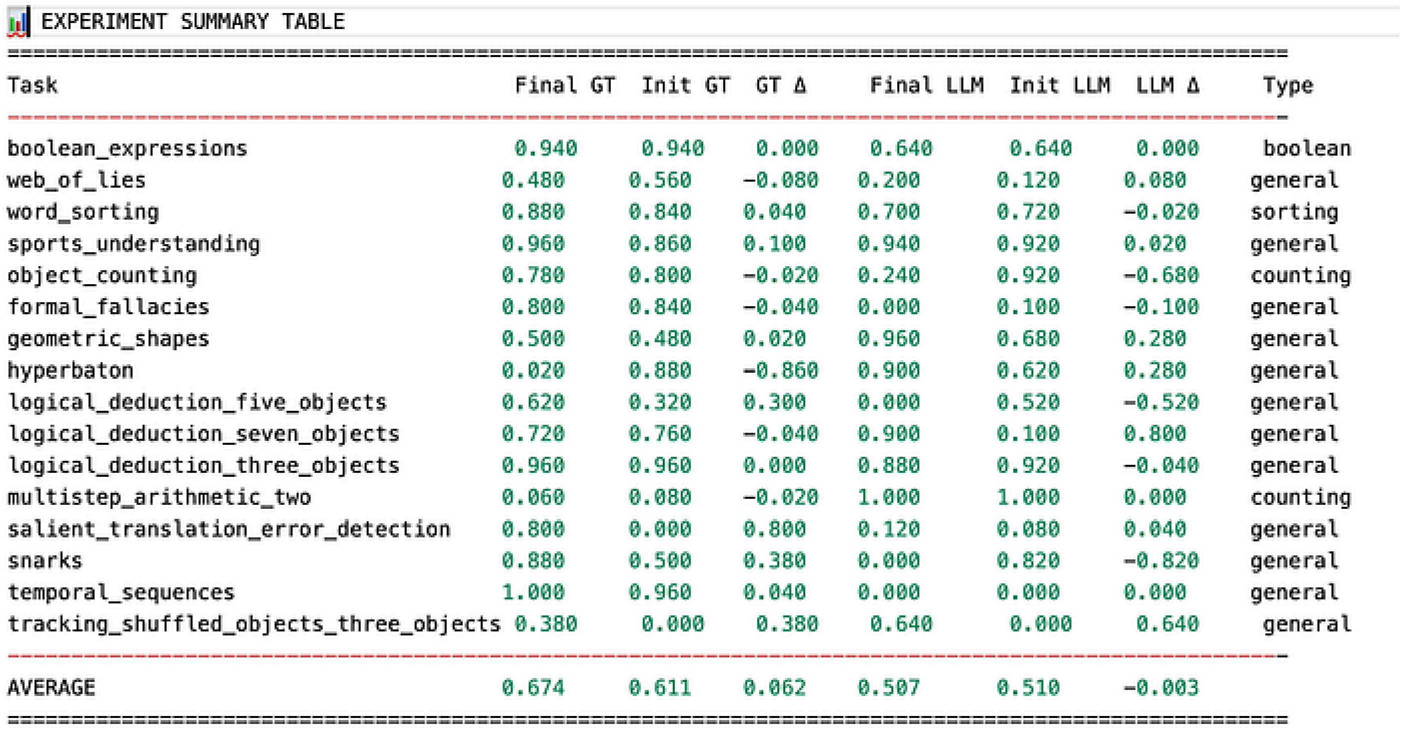
Big Bench Hard

Prompt learning was tested against Big Bench Hard with GPT-4.1 as the model under test (gpt-4.1–2025–04–14) and GPT-4o as the evaluation model.

There is no handcrafted prompt for each test, only a simple evaluation prompt; the actual prompt is learned from the evaluation results in an iterative fashion.

The results consistently show an improvement over baseline with the result below showing a 10% improvement on what is a highly saturated benchmark.

The BBH was run with 50 random samples, with 1 loop of iteration. We ran a selection of the BBH tests, a number of tests had issues that were not fixed by publication time and are not included.



```
📊 EXPERIMENT SUMMARY TABLE
====================================================================================================
Task                                       Final GT   Init GT   GT Δ     Final LLM   Init LLM   LLM Δ     Type
----------------------------------------------------------------------------------------------------
boolean_expressions                        0.940      0.940     0.000    0.640       0.640      0.000     boolean
web_of_lies                                0.480      0.560     -0.080   0.200       0.120      0.080     general
word_sorting                               0.880      0.840     0.040    0.700       0.720      -0.020    sorting
sports_understanding                       0.960      0.860     0.100    0.940       0.920      0.020     general
object_counting                            0.780      0.800     -0.020   0.240       0.920      -0.680    counting
formal_fallacies                           0.800      0.840     -0.040   0.000       0.100      -0.100    general
geometric_shapes                           0.500      0.480     0.020    0.960       0.680      0.280     general
hyperbaton                                 0.020      0.880     -0.860   0.900       0.620      0.280     general
logical_deduction_five_objects             0.620      0.320     0.300    0.000       0.520      -0.520    general
logical_deduction_seven_objects            0.720      0.760     -0.040   0.900       0.100      0.800     general
logical_deduction_three_objects            0.960      0.960     0.000    0.880       0.920      -0.040    general
multistep_arithmetic_two                   0.060      0.080     -0.020   1.000       1.000      0.000     counting
salient_translation_error_detection        0.800      0.000     0.800    0.120       0.080      0.040     general
snarks                                     0.880      0.500     0.380    0.000       0.820      -0.820    general
temporal_sequences                         1.000      0.960     0.040    0.000       0.000      0.000     general
tracking_shuffled_objects_three_objects    0.380      0.000     0.380    0.640       0.000      0.640     general
----------------------------------------------------------------------------------------------------
AVERAGE                                    0.674      0.611     0.062    0.507       0.510      -0.003
====================================================================================================
```

Big Bench Results

The Final GT is the final prediction measured relative to the ground truth. The ground truth is not used in the prompt learning loop, in the spirit of Big Bench, only to test against in the final check.

The Init GT is the initial measurement against ground truth with no prompt.

The data is split 50 / 50 split testing 50 samples on each. The initial test data is a different random sample from the final test data.

What we found in our testing of the current public models is that they are so attuned to the benchmark that many times any small prompt change at all actually lowered the benchmark. This is a well known benchmark but the question of saturation of results is very real.

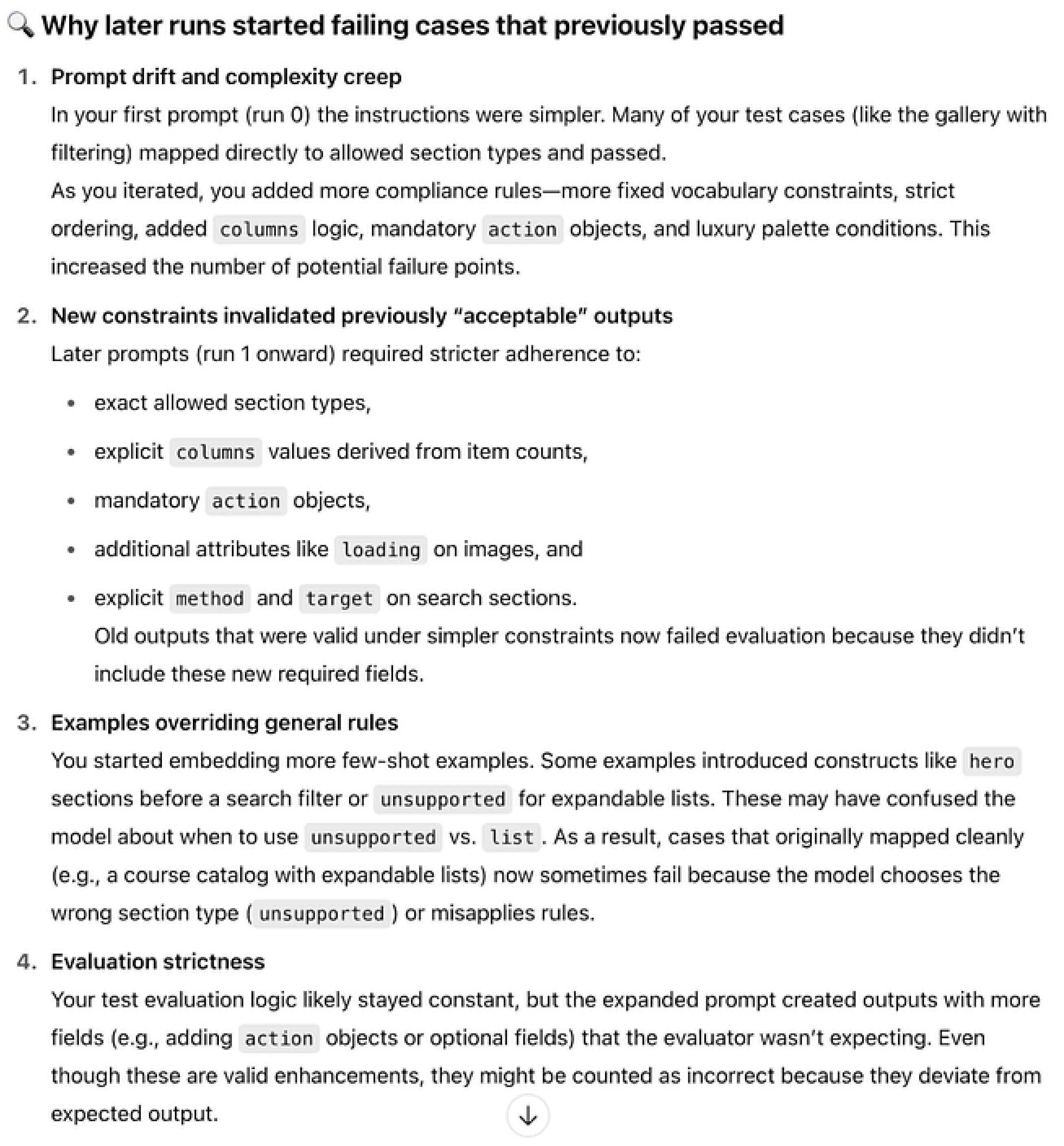
Differences from RL Training

In typical gradient based training most gradient based updates will improve your training scores. In prompt learning, that is not guaranteed, it does in practice for most loops but we have found running for long enough can affect your training scores. The reason is training is not a gradient update but a best attempt to fix a prompt problem, that best attempt might or might not work.

This is not the typical case but the following can happen:

- Training results can drop after a run
- Test results can be higher than training

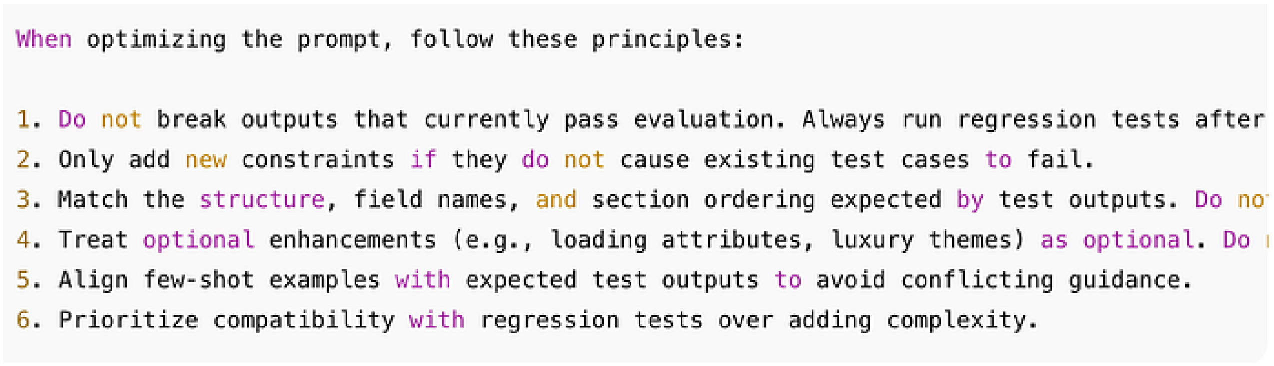
We leveraged o3 to analyze the failures of the prompt optimization runs looking at raw data and prompt changes.



🔍 Why later runs started failing cases that previously passed

1. **Prompt drift and complexity creep**
   In your first prompt (run 0) the instructions were simpler. Many of your test cases (like the gallery with filtering) mapped directly to allowed section types and passed.
   As you iterated, you added more compliance rules—more fixed vocabulary constraints, strict ordering, added `columns` logic, mandatory `action` objects, and luxury palette conditions. This increased the number of potential failure points.
2. **New constraints invalidated previously "acceptable" outputs**
   Later prompts (run 1 onward) required stricter adherence to:
   - exact allowed section types,
   - explicit `columns` values derived from item counts,
   - mandatory `action` objects,
   - additional attributes like `loading` on images, and
   - explicit `method` and `target` on search sections.
     Old outputs that were valid under simpler constraints now failed evaluation because they didn't include these new required fields.
3. **Examples overriding general rules**
   You started embedding more few-shot examples. Some examples introduced constructs like `hero` sections before a search filter or `unsupported` for expandable lists. These may have confused the model about when to use `unsupported` vs. `list`. As a result, cases that originally mapped cleanly (e.g., a course catalog with expandable lists) now sometimes fail because the model chooses the wrong section type (`unsupported`) or misapplies rules.
4. **Evaluation strictness**
   Your test evaluation logic likely stayed constant, but the expanded prompt created outputs with more fields (e.g., adding `action` objects or optional fields) that the evaluator wasn't expecting. Even though these are valid enhancements, they might be counted as incorrect because they deviate from expected output.

The above gave us insights to improve the Metaprompt and evaluation prompts.



```
When optimizing the prompt, follow these principles:

1. Do not break outputs that currently pass evaluation. Always run regression tests after
2. Only add new constraints if they do not cause existing test cases to fail.
3. Match the structure, field names, and section ordering expected by test outputs. Do no
4. Treat optional enhancements (e.g., loading attributes, luxury themes) as optional. Do
5. Align few-shot examples with expected test outputs to avoid conflicting guidance.
6. Prioritize compatibility with regression tests over adding complexity.
```

Example of improvement to MetaPrompt based on Eval data and failure cases. It feels like a future area of research is how to improve the evaluation prompts and metaprompts based on evaluation and prompt data.

Speed of Library

In our tests, the major benchmark harness runs took over +24 hours. Where a run through Big Bench Hard of Arize Phoenix is around 30minutes. The library design makes prompt iteration runs about 10–100x faster than the current ecosystem.

More On Prompt Learning

Prompt learning presents a compelling approach for continuous improvement of AI applications, and its ability to drive results with relatively few examples make it suitable for both early stage and production applications.

Mentions by Author

- Promptbreeder (DeepMind)
- OPRO — "LLMs as Optimizers"
- PromptAgent
- StablePrompt
- Meta-Prompting: Task-Agnostic Scaffolding
- Critic-RM — Self-Generated Critiques Boost Reward Modeling
- Self-Refine: Iterative Refinement with Self-Feedback
- Self-Generated Critiques Boost Reward Modeling for Language Models