AI RESEARCH

Which Action Deserved the Reward? Exploring Credit Assignment in Reinforcement Learning

Anand Murugan • August 27, 2026 • Research retrospective • Action-level credit in language-model and synthetic tasks

IN THIS ARTICLE

Abstract

Reinforcement learning often gives a model one score for a complete attempt and applies that same score to every action in the attempt. This treats helpful and harmful actions alike. We asked whether training on the effect of each action could work better than training on the final result alone.

We found two limited benefits. On an interactive arithmetic task, action labels raised Qwen's success from 29 to 37 of 192 episodes; training on the final result reduced it to 23. But action training also caused many malformed commands. On a small synthetic task, exact action labels improved average success during training by 8.60 percentage points when both methods used the same attempts. The advantage disappeared when we also counted the extra work needed to calculate those labels.

The broader result was negative. Written reasoning steps were not reliable units of action.

Exact action values did not automatically improve Qwen. Putting task states into graphs did not help across tasks that lacked shared meaning. A separate small model scored well only after code had already calculated the consequences of each arithmetic choice and, in later tests, exposed much of the short search. We therefore do not count it as evidence for a general solution.
The study supports a modest conclusion: feedback about individual actions can help, but only if the actions are well defined, the labels are affordable, the training method uses them correctly and the model can still produce valid output. We do not yet have a method that meets all four requirements for general language models.

Introduction

In reinforcement learning, a model takes actions and receives a reward based on how well the attempt went. When training uses only this final outcome, every action in the episode receives the same reward. If the episode succeeds, all of its actions receive positive feedback. If it fails, all receive negative feedback.

This becomes especially awkward for language-model agents. They generate text one token at a time, while their meaningful effects often occur at a larger scale. A model may make a tool call, edit a file, issue a command or take several turns to complete a task. Applying one final reward uniformly can reward mistakes inside a successful attempt and punish useful actions inside a failed one.

The idea behind this project was to train on actions rather than spread one final result across all generated tokens or turns. For each action, we wanted to estimate how it changed the state and the chance of eventual success. A helpful action should receive positive credit, a harmful one negative credit, and an irrelevant one little or none.

That proposal raises several separate questions:

1. What should count as an action in language-model output?
2. Can we measure the effect of an action reliably?
3. Does action-level feedback train a better model?
4. Is calculating that feedback worth its cost?
5. Can one method work across different tasks?

We used small tasks with exact rules so that these questions could be tested directly. The tasks were not intended to imitate all uses of language models. They were controlled settings in which every action, state change and final result could be checked.

The tasks and model interfaces

We used small tasks with exact rules. This let us check every action without asking a person or another model to judge it.

1. Select the useful tape positions

In the MAD selective-copy task, the model saw a short tape and had to select every non-blank position.

```
position:  0    1    2    3    4    5    6    7
tape:      __   t04  __   __   t09  __   __   __
```

The correct selection is positions 1 and 4.

The model's input token set was:

```
{
  unselected blank, selected blank,
  unselected t00, ..., unselected t11,
  selected t00, ..., selected t11,
  choose an action
}
```

Its output set was:

```
{
  select position 0, ..., select position 7,
  unselect position 0, ..., unselect position 7,
  submit
}
```

The model scored every allowed output and chose one. The program carried out that action, updated the tape and called the model again.

```
TURN 1
model input:  tape with no positions selected
model output: select position 1
new state:    selected positions [1]

TURN 2
model input:  tape with position 1 selected
model output: select position 4
new state:    selected positions [1, 4]

TURN 3
model input:  tape with positions 1 and 4 selected
model output: submit
program:      correct; reward 1; episode ends
```

An incorrect submission did not end the episode if turns remained. The model could repair a bad selection by unselecting it.

2. Let Qwen act one command at a time

Qwen received four numbers and a target. It had to combine the numbers until only the target remained. The program accepted three command forms:

```
<ACT>COMBINE number operator number = result</ACT>
<ACT>UNDO</ACT>
<FINAL>number</FINAL>
```

Qwen still used its normal language-model output set. The three lines above were required forms, not a restricted vocabulary. Qwen could produce ordinary prose or a malformed command, which the program rejected.

A successful episode looked like this:

```
TURN 1
model input:  target 14; available [2, 3, 4, 5]; 7 turns left
model output: <ACT>COMBINE 2 + 3 = 5</ACT>
program:      accepted; available [4, 5, 5]

TURN 2
model input:  full conversation; available [4, 5, 5]; 6 turns left
model output: <ACT>COMBINE 4 + 5 = 9</ACT>
program:      accepted; available [5, 9]

TURN 3
model input:  full conversation; available [5, 9]; 5 turns left
model output: <ACT>COMBINE 5 + 9 = 14</ACT>
program:      accepted; available [14]

TURN 4
model input:  full conversation; available [14]; 4 turns left
model output: <FINAL>14</FINAL>
program:      correct; reward 1; episode ends
```

A rejected command used a turn but left the numbers unchanged. `UNDO` reversed the latest accepted combination. The episode ended when Qwen submitted the correct final value or ran out of turns.

3. Ask Qwen for one written solution

Our first Qwen experiment used a different design. Qwen wrote a complete answer in one response:

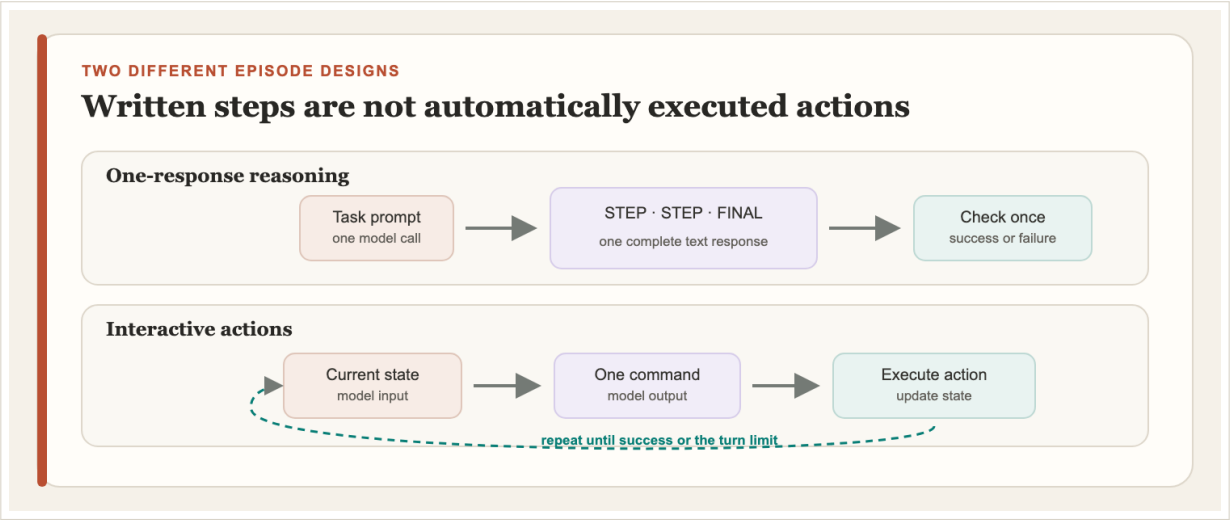
```
MODEL INPUT
Numbers: 2, 3, 5
Target: 10
Use + or −. Use each number at most once.
Output STEP lines followed by one FINAL line.

MODEL OUTPUT
<STEP>2 + 3 = 5</STEP>
<STEP>5 + 5 = 10</STEP>
<FINAL>10</FINAL>

PROGRAM
check the complete response once; reward 1 if it is valid
```

Qwen again used its normal output set. `STEP` and `FINAL` were requested tags, not the only text it could generate.

This task had one model response and one final check. The program did not execute the first `STEP`, return a new state and ask Qwen what to do next. The two `STEP` lines were parts of one written answer, not actions in an interactive episode.



This distinction shaped the rest of the study. We first tested written steps, then moved to commands whose effects were explicit.

What the main terms mean

The examples above separate several terms that are easy to blur together:

- A **token** is one piece of Qwen's generated text. A command such as `<FINAL>14</FINAL>` contains several tokens.
- A **reasoning step** is a written unit such as `<STEP>2 + 3 = 5</STEP>`. It may describe a calculation without causing the task to execute it.
- An **action** is one choice carried out by the task program. `select position 1` in MAD and `<ACT>COMBINE 2 + 3 = 5</ACT>` in interactive arithmetic are actions.
- The **state** is the current task situation. In MAD it includes the tape and the selected positions. In arithmetic it includes the available numbers, action history and turns left.
- An **episode** is one complete attempt, from the initial state until success or the turn limit.
- The **reward** is the score for the episode. Our tasks usually used 1 for success and 0 for failure.
- **Credit** is an estimate of how much one earlier action helped or hurt the final result.

The important distinction is between written reasoning and executed action. They can match, but they need not. In the one-response task, a `STEP` was only text. In the interactive task, a `COMBINE` command changed the numbers available on the next turn.

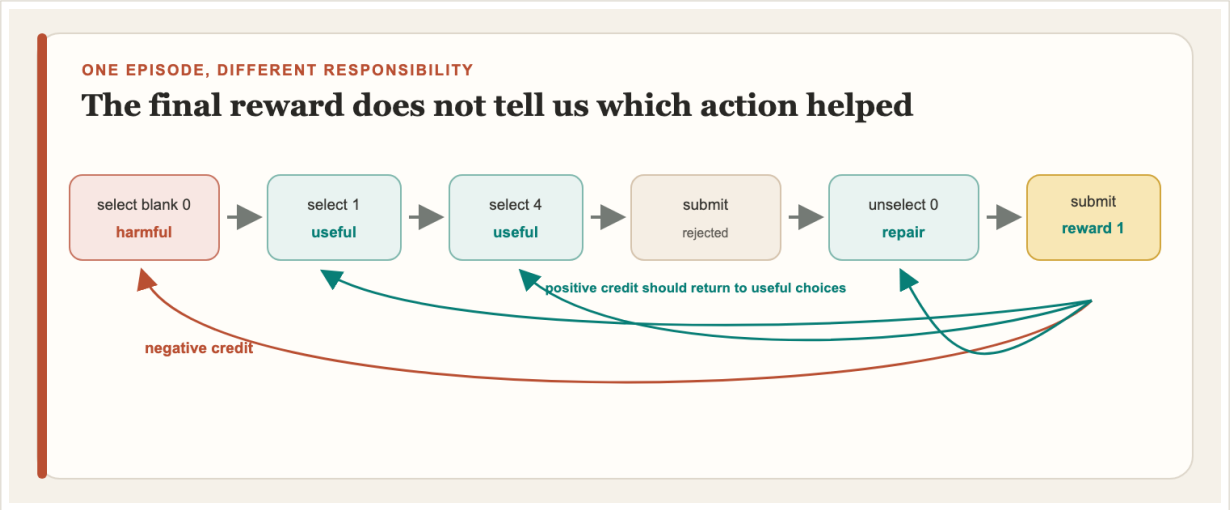
The credit-assignment problem

Now consider a MAD episode in which the model made a mistake and later repaired it:

```
select blank position 0
select useful position 1
select useful position 4
submit — rejected
unselect position 0
submit — success
```

The final reward is 1. Yet the first selection was harmful, the next two were useful, the first submission wasted a turn and the unselection repaired the mistake. Giving every action the same positive label hides these differences.

Credit assignment asks which earlier choices deserve credit or blame. For a language-model agent, this also requires choosing the right unit: individual tokens, written steps, executed commands or something else.



What we tested

Written reasoning steps were not reliable actions

We began with written Qwen arithmetic solutions. At each `STEP` boundary, we generated many possible endings and measured how the step changed the chance of a correct final answer.

Most of the measured change occurred before the final answer, which initially looked promising. But the result was not stable. On fresh data, whole-step boundaries preserved only 68.7% of the finer changes inside the response. Only 22.4% of the largest changes reliably kept the same positive or negative sign.

We stopped this line of work. A sentence that looks like a reasoning step is not automatically a useful unit for learning. The stronger boundary is an action that the task actually executes.

Exact action labels helped with fixed experience, but cost too much here

We next used the MAD tape task with a 102,000-parameter Transformer. Because the task was small, we could calculate how each action changed the chance of eventual success under the current model.

Before training, we checked that the labels were measurable. Across 605 states and 34,624 sampled completions, the positive or negative signs of the largest action values agreed between independent samples 89.3% of the time. Almost all of the measured credit appeared before the final `submit` action.

Across five paired runs, exact action labels improved average learning by 8.60 points when both methods used the same main attempts. Final success reached 82.4%, compared with 67.2% for training only on the final result.

Calculating the action labels required many extra trial runs. When the ordinary method received

additional attempts equal to that work, it reached 90.0% and the action-label method lost by 6.42 points averaged across training.

COMPARISON	RESULT FOR EXACT ACTION LABELS
same main attempts	**+8.60 points**
same total trial budget	**-6.42 points**

These points measure average success across training, not only the last saved model.

Both results matter. Detailed labels made better use of a fixed set of attempts, but producing those labels was not the best use of computation in this cheap simulator.

We also trained a model to predict action values so that they would not need to be recalculated every time. The predictor was accurate on nearby tasks, but its training data cost 161,464 environment steps. After counting that cost, it still lost to ordinary outcome training.

Action labels helped Qwen choose better actions, but damaged its commands

We then trained Qwen3-1.7B on the interactive arithmetic task. The original model generated one fixed training set. Every trained model used those same episodes and the same update procedure. Only the labels changed:

- final-result training gave each turn the episode's final result;
- action training rewarded useful accepted actions and penalized rejected ones;
- grammar-protected action training also tried to preserve command syntax.

The results across 192 evaluation episodes were:

MODEL	SUCCESS	VALID FORMAT	ACCEPTED TURNS	UNDO	INVALID TURNS
original Qwen	29	99.37%	56.03%	223	8
final-result training	23	**99.85%**	44.20%	400	**2**
action training	**37**	91.95%	**72.81%**	**172**	100
action training with grammar protection	32	99.05%	60.55%	208	12

Action training improved the choices Qwen was trying to make. It accepted more actions, used `UNDO` less and gained ten tasks that the original model missed, while losing two that the original had solved. Training on the final result made Qwen much more likely to use `UNDO` and solved six fewer episodes than the original.

But action training damaged the command format. A common error was:

```
<ACT>FINAL 22</ACT>
```

instead of:

```
<FINAL>22</FINAL>
```

The intended action was clear to a reader, but the program rejected it. Protecting the grammar reduced invalid actions from 100 to 12, while retaining only part of the success gain. The stronger protection setting was chosen partly in response to earlier evaluations, so this repair is useful evidence rather than an independent confirmation.

This is our clearest positive language-model result, and its limitation is equally clear: better action choice is not enough if the model can no longer express the choice correctly.

Exact values did not automatically improve Qwen

To remove command-format errors, we ran another experiment in which code listed all legal actions and assigned each a temporary letter. Qwen chose a letter; the program executed the corresponding action. We also replaced simple action labels with exact values calculated from the complete graph of possible future states.

This failed:

LEARNING RATE	ORIGINAL	FINAL-RESULT TRAINING	EXACT ACTION VALUES
1e−6	13/32	13/32	12/32
3e−6	13/32	12/32	11/32

A simple program that always chose the highest exact value solved every development task. The labels therefore contained enough information. Qwen did not learn to use that information well.

The main problem was context. A final action is excellent when the target has been reached and harmful when it has not. The update pushed Qwen toward choosing final actions more often instead of learning that distinction.

The values had been calculated for a reference rule that chose uniformly from the menu. Qwen chose final actions 51.9% of the time, compared with 24.4% for that reference. Under the reference rule, a premature final action received only a small penalty because later random choices could still repair the episode. That made the values poorly matched to Qwen's actual behavior.

Giving Qwen all action values at once did not solve the mismatch. One dense target reduced success from 13/32 to 2/32 because the model spread its probability across many letters and usually ran out of turns. Hard ranking and pairwise comparisons also failed.

An exact label is not a complete learning method. The model still has to learn the right comparison in the right state.

A shared graph format did not create shared meaning

We also tried to describe states as graphs. Objects became points, and relations such as “inside” or “selected” became links. The hope was that one credit model could then work across several tasks.

Graphs helped within one small workspace. Sharing one graph model between two related predictions improved one of them by 7.03 points. A model later rebuilt the graphs from controlled language with 93.23% complete accuracy on new tasks.

The graph did not transfer well from the workspace to MAD. Shared training was worse than MAD-only training in four of five runs. The two environments had no shared action names, object types or task relations. They used the same container, but the contents meant different things.

The lesson is straightforward: a graph can record connections, but it does not decide which connections in different tasks have the same meaning.

We also tested a graph-based safety check that could reject actions predicted to be harmful. In one fresh test it kept all 224 useful corrections and reduced harmful ones from 19 to 2; across 4,032 episodes it added ten successes and caused no losses. Later tests found a false rejection, while a more cautious version became so conservative that it never acted. The safety result was promising but not reliable enough to generalize.

A high-scoring side result that should not be overstated

One branch produced much larger numbers. We trained a small model to score legal arithmetic actions. Before the model saw an action, code:

1. executed the action exactly;
2. built the resulting state; and
3. listed every legal calculation available one step later.

On new states, the model's action rankings compared as follows:

MODEL	CHOSE AN EXACTLY BEST ACTION	CORRECT PAIR ORDERING
flat model	70.3%	74.1%
same model trained with shuffled labels	68.8%	53.9%
model given program-generated future calculations	**96.0%**	**93.8%**

When used to control complete episodes, five trained models had a median of 118 successes out of 128, compared with 54 for Qwen.

The model did learn something: the same model trained with shuffled labels failed, and a simpler version performed much worse. But the comparison is not fair evidence of general reasoning or credit assignment. Qwen had to read the original task and produce commands. The small scoring model received legal choices, exact state changes and program-generated future calculations.

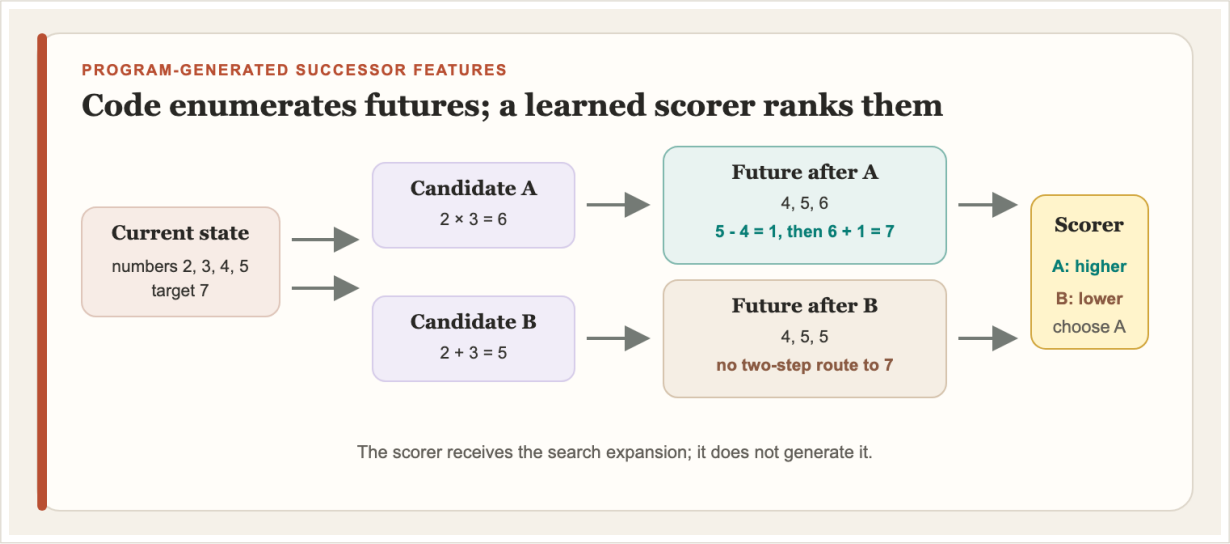
The 118/128 model received the state after each candidate and every calculation available one step later. It did not receive the complete route shown below. We added that stronger two-operation input later, after harder tests exposed failures on multiplication, larger values and small targets. On these small arithmetic problems, the new input exposed much of the short solution search directly.

```
task: numbers [2, 3, 4, 5], target 7

candidate A: 2 × 3 = 6
state after A: [4, 5, 6]
route supplied by code: 5 − 4 = 1, then 6 + 1 = 7

candidate B: 2 + 3 = 5
state after B: [4, 5, 5]
route supplied by code: no two-step route reaches 7

model output: one score for A and one score for B
program choice: whichever candidate has the higher score
```



The later system scored 64/64, 64/64, 60/64 and 63/64 across four arithmetic test families. A learned text-processing model also reproduced all 1,355 decisions made from the exact inputs.

These results show that a learned model can use a well-prepared search record. They do not show that it can discover the consequences or the search from a normal language-model prompt. We include this branch because it clarified which part was easy and which part we had moved into code. We do not treat its high score as the main result of the project.

Learning values from final outcomes was still too hard

The assisted scoring model above was trained with exact answers. We next tried to learn action values using only final success or failure from random completions. Each candidate action received 64 completions. In total, the dataset contained about 1.81 million completions.

The measured values were stable: two independent halves of the data had a correlation of 0.982. Exact analysis also showed that the underlying values were sufficient to solve every audited task. The trained models still failed too often on the first decision of the harder arithmetic problems. Their five scores were 49, 49, 48, 50 and 50 out of 64; the combined model scored 49. Removing actions that the environment would reject raised the median only to 50.

Once the first choice was correct, the model usually finished the task. The main failure was telling apart several first actions whose values were close. More data from the model's own states and different ranking losses did not fix it.

This result narrows the remaining problem. The labels were not obviously noisy, language parsing was no longer involved and later planning was not the main bottleneck. The learned model could not reproduce the close comparisons that mattered at the first step.

What the evidence supports

The experiments support four conclusions.

1. **Use real actions when possible.** A command or tool call has a clear effect. A written reasoning step may not.
2. **Action labels can help with fixed experience.** We saw this in MAD and, more narrowly, in Qwen's action choices.
3. **Count the full cost.** In MAD, calculating detailed labels cost more than collecting enough ordinary outcomes to win.
4. **Judge the whole agent.** Better choices are not useful if command syntax, safety or execution reliability gets worse.

The experiments also rule out several easy answers:

- exact action values do not automatically train a better language model;
- a common graph format does not create common meaning across tasks;
- a model that ranks program-generated search records is not a general planner;
- stable sampled values can still be difficult for a learned model to compare.

Limits

The tasks were synthetic arithmetic, selective copy and a small controlled workspace. They are useful for measurement, but they do not represent open-ended language-model use.

The direct language-model experiments used Qwen models, small offline datasets and small local updates to their weights. We did not run large, repeated online reinforcement-learning experiments.

Many later experiments were chosen after seeing earlier failures. We used fresh test sets and repeated runs where practical, but later questions were chosen after we saw earlier results. This was not one study planned completely in advance.

Conclusion

The original idea was simple: if an agent takes several meaningful actions, train it on what each action did instead of copying the final reward onto every token.

The study found a real but narrow benefit. Action labels improved Qwen's arithmetic choices and helped a small model learn more from fixed experience. Neither result became a complete improvement: Qwen's command format degraded, and the synthetic benefit vanished after counting the

cost of the labels.

The failed branches were useful. They showed that choosing the action boundary, calculating accurate values, training a model on those values and preserving valid commands are separate problems. Solving one does not solve the others.

A convincing general method must improve an actual language model, preserve valid behavior, beat ordinary outcome training after all costs are counted and work beyond the tasks used to create its labels. We have not reached that point.

Project record

The public code and experiment records are available in the reinforcement-learning credit-assignment repository[1]. The artifact guide[2] links each result in this article to the relevant scripts, configurations and machine-readable summaries.

[1] https://github.com/kupilikula/rl-credit-assignment
[2] https://github.com/kupilikula/rl-credit-assignment/blob/main/ARTIFACTS.md