



Where should a looped model spend its compute on a decision?

We had one week and one RTX 4070 Ti to find out, while building wyat-2-1.4b. This brief gives what we learned, whether it fits you, how to start, and what we would do next with more compute.

1.76×

faster than full depth
234 vs 412 ms per question

1/4

of the question cache
11.5 vs 46.1 MiB kept per
question

−2 pts

of accuracy
91.2% vs 93.3%

4+2 means four loops reading the question, then two loops per candidate answer; full depth (4+4) gives each answer four. These numbers compare the two using adapter arms on the same 1.4B base, trained only on 3,370 ARC science questions, one seed each, built to compare schedules. Accuracy on 2,524 ARC questions not used in their training or selection; speed and cache from a separate timing run of these same trained arms. The released wyat-2-1.4b uses the same 4+2 schedule but was trained on 136,104 decision rows; it was not measured on ARC.

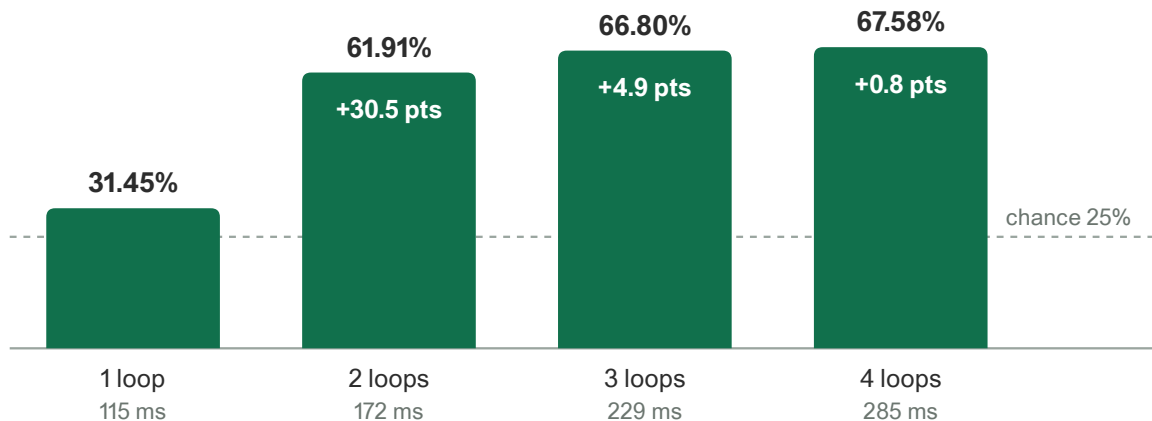
The short version

- **Looping helps, with steep diminishing returns.** The second loop does most of the work; the fourth adds almost nothing for a quarter more time.
- **Small trained adapters look like what changes the economics.** In our test they let the model keep one memory (the cached copy of the question from one loop) instead of one per loop with no loss seen (tested against two memories; one seed: a strong lead, not proof). Keeping one of four is where the 1/4 comes from.
- **Our schedule is a trade, not a free lunch:** 1.76× faster, 1/4 of the question cache, about 2 points of accuracy. We had set a 2-point limit before testing and missed it: the loss could be as large as about 3 points (95% interval: −3.24 to −1.14 points).
- **The question that matters most is still open:** does a looped model beat a normal model of the same cost? It is cheap to settle, and page 4 says how.

What we learned

1 · The second loop does most of the work

Ouro-1.4B reuses the same layers in loops. Before any training, we varied how many loops each candidate answer gets, with the question read four times.



Untouched Ouro-1.4B, before any adapter training; 512 ARC questions already used in earlier tests; one run. Accuracy by loops per answer, with median time per question. This is a different run and model from the trained arms on page 1, so its times differ from the 234 and 412 ms there.

One loop is near chance. Two loops double it. The third adds 4.9 points; the fourth adds 0.8 for 25% more time. **Spend loops where the curve is steep, not where it is flat.**

2 · Training looks like what lets you keep one memory

The cheap way to save memory is to keep only the last loop's memory of the question instead of one per loop. We tested it with two answer loops, so one memory against two. Without adapters (only a small scoring head trained), that costs 4 points (86.4% → 82.4%, over 10 training seeds). With small trained adapters on the attention layers, we saw no loss (+1.2 points, but one seed and an interval of -0.8 to +3.3 that includes zero). **A short adapter training run (under two hours per arm here) is the strongest lead we have for the ¼ cache; it is not yet proven.**

3 · The 4+2 package costs about 2 points

Going from 4+4 to 4+2 costs 2.15 points, and 3.2 on the hardest questions. That change also keeps one memory instead of four and halves the active adapters, so this is the cost of the whole package, not of loop count alone; each arm is one training seed. Also cutting the question's loops from four to two (2+2) costs a further 0.94, which our test cannot tell apart from zero. **The question side may be cheaper to cut than the answer side;** we have not isolated it yet.

4 · Why the built-in early exit is switched off

Ouro has a learned exit gate that can stop looping early. **wyat-2 switches it off and runs a fixed 4+2 schedule**, for three reasons. It decides after all four question loops have run, so it cannot cut that cost. It was trained to predict the next word, not to score decisions. And on 384 decision questions we had already used, at thresholds 0.5, 0.75 and 0.9, it never stopped before loop four. Retrained on decision scores, it could let easy answers stop early.

5 · Re-used test sets flatter you

On the 512 questions we had used to pick the model, 4+2 looked only 0.98 points worse. On 2,524 fresh questions it is 2.15 points worse. A 512-question set can only resolve about ± 2 points. **Test on questions nobody has looked at, and enough of them.**

Is it a good idea for you?

Good fit

- You score many candidates (choices, rankings, yes/no) against one shared context.
- Latency, or memory for cached questions, is what limits you.
- A loss of about 2 points is acceptable, or you can measure it on your task.

Not yet

- You need every point of accuracy: run full depth (4+4).
- You route tools: our tool-use score dropped.
- You need free text: the model only scores candidates.
- You need an SLA today: 1 in 20 requests takes about 1.6 s or longer.

What else we measured

- **Public benchmark:** on the public Jev Decision Index (higher is better), the released model scores 27.58, against 25.69 for our previous model. One run, also used to pick the model: a direction, not a leaderboard claim.
- **Tool use got worse:** the benchmark's tool-use score fell by 8.57 points.
- **Speed of the released model** on that benchmark, on our RTX 4070 Ti: the typical request takes 0.14 s; 1 in 20 takes about 1.6 s or longer, against the benchmark's 1.0 s limit.

How to start

Fastest start: try the released model on your own questions (page 5). To adopt the recipe on your own data, train both arms yourself:

1. Fix one decision task and a fresh test set of at least 1,000 questions.

Why: 512 questions only resolve about ± 2 points, and re-used sets flatter you.

2. Train small per-loop adapters on a frozen base for two arms: full depth (4+4) as your reference, and 4+2 with one kept memory.

Why: the reference must be trained the same way, or the comparison flatters 4+2. In our test the adapters let one memory replace one per loop with no loss seen (tested against two memories, one seed). The released 4+2 model carries 13.6M adapter and head parameters on a 1.4B base; each trained arm took under two hours on one RTX 4070 Ti.

3. Price 4+2 against the 4+4 reference on your fresh set.

Why: that is the trade, and only your task can price it.

4. Use the exact mode.

The exact mode computes attention in full and is the default; the fast mode uses chunked Flash attention. Why: the fast mode is 2.9× faster on long inputs but changes some scores. Validate it on your task before using it.

What is still open, and how to settle it fast

OPEN QUESTION	HOW TO SETTLE IT	ROUGH COST
Does looping beat a normal model of the same cost?	Adapt a pretrained non-looped model matched on parameters or compute with the same recipe, and compare on a fresh set.	A few 1–2 h training runs plus evaluation (our estimate)
Is the 2-point cost stable?	Two more training seeds per arm.	About 1 h per seed for 4+2, 2 h for 4+4
Are three answer loops the sweet spot?	Train a 4+3 arm; none exists yet.	About 1.5 h (estimate)
Can depth adapt per question?	Retrain the exit gate on the decision score instead of next-word prediction.	To be scoped
Does long context keep its quality?	Train the long-context path; it fits in 8 GB today, with almost no headroom, but is untrained.	To be scoped

If you have more compute

Our bets, ranked. These are hypotheses from the results above, not results.

1. Settle the matched comparison first.

Everything else depends on whether looping beats a normal model of the same cost.

2. Make depth adaptive.

The 4+2 package costs 3.2 points on hard questions but 1.1 on easy ones. If loop count drives that gap, four answer loops for hard questions and two for easy ones could keep most of the accuracy at most of the saving.

3. Scale the base, keep the recipe.

The cache saving is a ratio of the question cache, so it grows in absolute terms with longer contexts.

4. Add seeds and tasks.

Three seeds per arm and a real decision task are what turn a proxy result into a product decision.

What's next

Try it on your own decision task. The model is available on Hugging Face at **teilmillet/wyat-2-1.4b** and runs the 4+2 schedule. Give it a question, its context and your candidate answers; it scores each one and returns the best:

```
uv add wyat --extra inference

import wyat
model = wyat.load("teilmillet/wyat-2-1.4b", backend="torch", device="cuda")
decision = model.decide(wyat.Request(
    query="Choose the requested booking action.",
    context="Please cancel my booking.",
    candidates=[wyat.Candidate(id="create", text="Create a booking"),
                wyat.Candidate(id="cancel", text="Cancel the booking")]
))
decision.selected  # "cancel"
```

Adapters, scoring head and runtime are Apache-2.0; the Ouro-1.4B base keeps its own terms. Score a fresh set of your own questions and compare with what you use today.

Our next experiments. First, the matched comparison on page 4: everything else depends on whether looping beats a normal model of the same cost. After that, two levers we have not pulled yet: **more and better data**, by repeating the loop comparison on decision data like the released model's own 136,104 training rows, and **a retrained exit gate** that learns from decision scores, so easy answers can stop before using all their loops. If any interest, please contact me (teilmillet almost everywhere).