ARC Prize 2026 ARC-AGI-2: Qwen3.8-Flash-Next agent on 4 L4 GPUs

amn
Kaggle team "I have an overfitting problem"
https://www.kaggle.com/nisshokuitsuki

Abstract

A public finetuned checkpoint of Qwen3.8-Flash-Next is used in an agentic configuration. We let it reason about each task in a harness with two tools: a Python executor including predefined helpers for testing code on the training pairs, and a submitting tool for the output grids, with which submission can be done by naming a session function or variable, or by giving a grid. The model runs on Kaggle's 4*L4 in 12 hours in virtue of Intel's W4A16 quantization of the routed experts, REAP pruning to 448 of 512 routed experts per layer, FP8 storage of the n-gram embedding table, the model's own multi-token-prediction (MTP) head as the drafter for speculative decoding with 3 draft tokens per step, and CPU offloading for the KV cache of the sparse-attention layers. The final notebook scored 77.22 (rank 10) on the public leaderboard when the competition closed.

1 Competition overview

In each ARC-AGI-2 task [1], a few training pairs of input and output grids are given and the outputs of one or more test inputs are asked for.

A prediction for a test output is correct only when it matches the ground truth exactly, and two predictions are allowed for each test output.

The submission is a notebook that Kaggle runs on 4 NVIDIA L4 GPUs within 12 hours, without network access, over 240 hidden tasks, i.e. 120 for the public and 120 for the private leaderboard. A task can have more than one test input; for example, the 120 public evaluation tasks have 172 test inputs overall.

2 Inference

Part	Configuration
Model	Qwen3.8-Flash-Next in W4A16, 448 of 512 experts per layer, AZ-Focus weight override, FP8 n-gram table
Drafter	MTP head with INT4 routed experts, 3 draft tokens per step, drafting over a 16340-token vocabulary
Engine	Merlin, a fork of vLLM 0.30 [2], built for sm89 and run as one server across the four GPUs with tensor parallelism 2 and pipeline parallelism 2

Preprint.

Part	Configuration
Agent	reasoning with a stateful Python tool and a submit tool, up to 64 concurrent trajectories, each with a 196608-token context
Attempts	two per test output; extra attempts with a vote if time permits

2.1 Engineering

The notebook installs the engine from the arc-merlin-wheels dataset, assembles the pruned model from the checkpoints mounted as Kaggle models (listed under Reproduction), starts one server across all four GPUs, and runs the agent (the harness and the model) from scratch on every task until the deadline.

2.2 Model & Customization

We use Qwen3.8-Flash-Next, which is a 48-layer mixture-of-experts (MoE) model [3]. Each layer of it routes a token to 10 of 512 experts, plus a shared expert.

Every fourth layer (layers 4, 8, . . . , 48) is a Qwen Sparse Attention (QSA) layer, and the other 36 are Gated DeltaNet (GDN) layers [4] as a form of linear attention. An n-gram embedding table with 20M bigram and trigram entries (51B parameters) is also added at layer 2.

We sample at temperature 0.7, top-p 0.95 and top-k 20.

We customize the model as follows.

Quantization. Instead of the original bf16 model we use Intel's W4A16 AutoRound checkpoint [5], which stores the routed experts in INT4 with group size 128 and keeps the other weights in bf16.

Expert Pruning. We keep 448 of the 512 routed experts in every layer, ranked by REAP [6].

Public Finetuned Weight. We use AZ-Focus, AxonZeta's fine-tuned version of the model, where 75 tensors in layers 20 to 44 differs from the original model. And we load its 50 bf16 tensors, which are shared-expert down projections and output projections of certain 19 Gated DeltaNet and 6 QSA layers. Loaded weights are an override over the corresponding tensors of Intel's checkpoint. We leave 25 routed-expert down projections since their changes are mostly lost in INT4 rounding.

N-gram table. We use the FP8 copy of the n-gram embedding table that RadixArk made and albucino assembled (albucino/Qwen3.8-Flash-Next-W4A16-FP8PLE), which halves occupied memory into 51 GB.

Speculative Decoding [7].

- For drafting we use the model's own MTP head, whose routed experts are quantized by albucino to INT4 with group size 32. Besides the main model's LM head adding 1 token each step, the single-layer MTP head is applied 3 times per step to draft 3 tokens, so each step produces 1 to 4 tokens.
- Following FR-Spec [8], our patch restricts the drafter to a 16340-token vocabulary, which covers every token in 43M tokens of our agent trajectories.
- Verification still uses the full vocabulary, so the output distribution is unchanged, and the main model can still produce tokens outside the drafter's vocabulary.

2.3 Throughput optimization

Kaggle's four L4 GPUs are connected over PCIe Gen3. GPUs 0 and 1 share one PCIe switch, and GPUs 2 and 3 share another.

Since when both GPUs behind one switch read CPU memory (including offloaded KV) at the same time, each of them reads at about an abnormally low 5.5 GB/s, we therefore start the server with `CUDA_VISIBLE_DEVICES=0,2,1,3`. It maps ranks 0, 1, 2 and 3 to the physical GPUs 0, 2, 1 and 3.

We also change the engine as follows.

KV offloading for QSA.

- vLLM reserves a fixed 3.9 GB of each GPU's memory for the KV cache at startup, but the number of tokens whose keys and values fit in it limits concurrency.
- We move keys and values, of the main model's & MTP drafter's QSA layers, out of this cache into 76 GB of CPU memory in total shared by all requests.
- The GPU KV cache now keeps only the QSA index and the GDN states, so the cache can hold several times more less-consuming tokens.
- A small number of 5344 most recent entries are still kept in reserved GPU space for KV cache since successive decoding steps of a request mostly select the same positions.

GDN states across turns. Only the state until previous turn's input and before tool output or model output is cached, where the actually latest is not. We keep the newest GDN state that generation reaches at a 4096-token block boundary, so at most one block of that output is recomputed in the next turn. While a trajectory waits for its tool reply, we copy its idle GDN states from the GPU KV cache to CPU memory and copy them back when its next turn starts.

Chunked verification. To verify draft tokens, vLLM copies the logits of a chunk of requests to fp32, and by default a chunk may hold up to 1 GiB of logits causing congestion. We lower that limit to 32 MiB to avoid running out of GPU memory at 64 requests on a nearly full L4.

Hierarchical concurrency fallback. If the engine fails to start or runs out of GPU memory, we restart it at the next lower level of concurrency, first 48, then 36, then 20 requests, each with a 131072-token context instead of 196608 in original.

As a result of throughput optimization, in a Kaggle commit run with concurrency 64 and ctx length 131072 as an earlier config, on the 120 public evaluation tasks (172 test outputs), the system took 240 trajectories and finished in 5.37 hours. Generation in this run had a one-minute peak of 1081 tokens/s and an average of 677 tokens/s.

3 Harness

All test inputs of a task are answered in one trajectory. The agent can use 2 tools introduced in the TL;DR.

After each python executor call, we check the validity (backtest) by running every `transform` or `solve` function in the code on every training pair. An invalid function would cause a reply with the problem causing pairs.

If a submitted function fails a training pair or the submitted grids repeat the first attempt's answer in later attempts, we refuse the submission once and accept the same call when it is repeated. On the final turn, we accept it at once.

Time limit. Each program run is limited to 5 seconds.

Answering order. Every task in the first attempt is arranged before later attempts. After first attempts, for minimizing performance loss when we cannot meet the running deadline, we categorize and set a priority for tasks to arrange later attempts as follows (highest to lowest):

1. Tasks whose first attempt hit the token, turn or time limit, or ended without a grid.
2. Tasks whose first answer was not from a function that reproduces every training pair, or broke an induced rule.
3. Tasks whose first attempt is still running.
4. The rest: tasks whose first answer came from a function that reproduces every training pair and keeps the induced rules.

Induced rule checks. If some induced rule among the following 3 applies to every training pair, then we check for that rule on the test outputs of functions that reproduce every training pair and warn the model upon violation; after a violation, the task's later attempts are ranked as if the function had failed a training pair:

1. The output uses exactly the input's colours.

2. An output of the input's shape keeps every background cell of the input. The background of a grid is its most common colour.
3. The output has as many non-background cells as the input.

Extra attempts. While time remains after the first two attempts, we run up to 3 more attempts for each task with fewer than 2 answers from functions that reproduce every training pair and pass the induced rule checks. When a test output has three or more distinct candidate grids, we fill its two answer slots first with grids from such functions and then with the grids that the most attempts produced.

Deadline. We set a deadline of 42300s (11.75 hours) after the notebook starts, with a cleanup reserve of 300s, so we stop generation at 42000s and write `submission.json`.

4 Comparison

We compare settings on the 120 public evaluation tasks.

Setting	Comparison on the same machine	Public evaluation performance
Experts per layer	448 against 512 without AZ-Focus, first attempts	73.61 against 64.86
AZ-Focus	with against without at 448 experts, first attempts	75.28 against 73.61
Context	recorded first attempts of an earlier configuration, scored as if cut at 65536 instead of 131072 tokens	31 to 36 points lower (37.5 to 43.5 of 120 tasks)
Temperature	0.7 against 1.0 at 448 experts with AZ-Focus & first attempts for 105 tasks	82.70 against 81.90 (standard error 3.75)

We compare repeated runs of one version.

Task set	Version and hardware	Scores
Public evaluation tasks	final configuration, two seeds, on 4 RTX 6000 Ada GPUs limited to the L4's memory	89.17 and 83.89 with two attempts; 79.17 and 76.94 with first attempts
Public leaderboard	final notebook, submitted twice on Kaggle's 4*L4	77.22 and 78.06

5 Compute and implementation

We run the final notebook on Kaggle's 4*L4. During development we measured throughput on rented 4*L4 machines and accuracy on 4 RTX 6000 Ada GPUs. A large portion of the code was written by Claude Opus 5.5 agents with human in the loop for every direction & decision.

6 Reproduction

Code and the competition notebook: itsuitsuki/arc-prize-2026-arc-agi-2.

Kaggle notebook: nisshokuitsuki/arc-prize-2026-solution.

Models: W4A16 checkpoint with the FP8 n-gram table, MTP drafter, AZ-Focus override.

Engine wheels: arc-merlin-wheels.

Engine patches, sandbox, kept experts and drafter vocabulary: arc-solution-files.

Third-party licences: `THIRD_PARTY_NOTICES.md` in the repository.

Acknowledgements

We thank the ARC Prize Foundation and Kaggle for hosting ARC Prize 2026.

We also thank the authors of the public work we build on:

- Adarsh's notebook arc-qwen-3-8-27b-agent: the `objects()` and `diff()` helpers, and a submission tool, which the model can call to record the answer of a program that reproduces the training pairs.
- Russell Kirk's notebook arcagi2-qwen38-freeform-direct and Tufa Labs' Duck harness: a harness that leaves the choice between reasoning directly and running Python to the model.
- NVIDIA's Nemotron-SFT-ARC-AGI-v1 dataset: the name and call format of our Python tool (we have not trained on the dataset).
- The ARC Prize Foundation's arc-agi-benchmarking: the wording of our task prompt and the parsing of grids from the model's answer.

The model stack is the work of the Qwen team (Qwen3.8-Flash-Next), Intel (W4A16 AutoRound), RadixArk and albucino (the FP8 n-gram table and the INT4 MTP drafter), AxonZeta (AZ-Focus) and dfranzen (the Kaggle copies). We serve it with halt95's Merlin engine, a fork of vLLM.

References

[1] Francois Chollet, Mike Knoop, Gregory Kamradt, Bryan Landers, Henry Pinkard. ARC-AGI-2: A New Challenge for Frontier AI Reasoning Systems. 2025. arXiv:2505.11831.

[2] Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E. Gonzalez, Hao Zhang, Ion Stoica. Efficient Memory Management for Large Language Model Serving with PagedAttention. In SOSP, 2023. arXiv:2309.06180.

[3] Zihan Qiu, Zekun Wang, Xiao Li, Yanpeng Li, Yang Xu, Yixuan Wang, Huaqing Zhang, Rui Men, Bochao Mao, Chengruidong Zhang, Fan Zhou, Hao Luo, Haofeng Huang, Haoran Lian, Haoyan Huang, Hongqing Chen, Jianwei Zhang, Jing Xu, Junjie Wang, Langshi Chen, Liangyu Wang, Linlang Jiang, Man Yuan, Minmin Sun, Peng Jin, Siqi Zhang, Siyu Wang, Xingzhang Ren, Yakai Wang, Yi Zhang, Yiming Dong, Yizhong Cao, Yubo Ma, Yunfei Mao, Bo Zheng, Dayiheng Liu. On the Design of Qwen3.8-Next Architecture: Evaluation, Efficiency, and Training Stability. 2026. arXiv:2608.30320.

[4] Songlin Yang, Jan Kautz, Ali Hatamizadeh. Gated Delta Networks: Improving Mamba2 with Delta Rule. 2024. arXiv:2412.06464.

[5] Wenhua Cheng, Weiwei Zhang, Haihao Shen, Yiyang Cai, Xin He, Kaokao Lv, Yi Liu. Optimize Weight Rounding via Signed Gradient Descent for the Quantization of LLMs. 2023. arXiv:2309.05516.

[6] Mike Lasby, Ivan Lazarevich, Nish Sinnadurai, Sean Lie, Yani Ioannou, Vithursan Thangarasa. REAP the Experts: Why Pruning Prevails for One-Shot MoE compression. 2025. arXiv:2510.13999.

[7] Yaniv Leviathan, Matan Kalman, Yossi Matias. Fast Inference from Transformers via Speculative Decoding. 2022. arXiv:2211.17192.

[8] Weilin Zhao, Tengyu Pan, Xu Han, Yudi Zhang, Ao Sun, Yuxiang Huang, Kaihuo Zhang, Weilun Zhao, Yuxuan Li, Jianyong Wang, Zhiyuan Liu, Maosong Sun. FR-Spec: Accelerating Large-Vocabulary Language Models via Frequency-Ranked Speculative Sampling. 2025. arXiv:2502.14856.