FACE-OF-AGI Report

1. Introduction

This report summarizes the research carried out around the ARC-AGI-3 challenge.

We explored various agentic approaches and at the time of the June 30 milestone cutoff we were in the top 10 on the public leaderboard out of more than 1500 participants, our team name: Face of AGI. This is a step towards more human-like problem-solving skills in AI agents. Our final solution can be found here: https://github.com/cocomette/artificial-agency.

The project started from a theoretical solution based on several AI-powered modules: one model to understand the world mechanics, one model to estimate progress toward the goal, and one agent to orchestrate decisions. The main idea was to let the agent reason through predictions before acting. Each of the models was implemented as an open-source VLM.

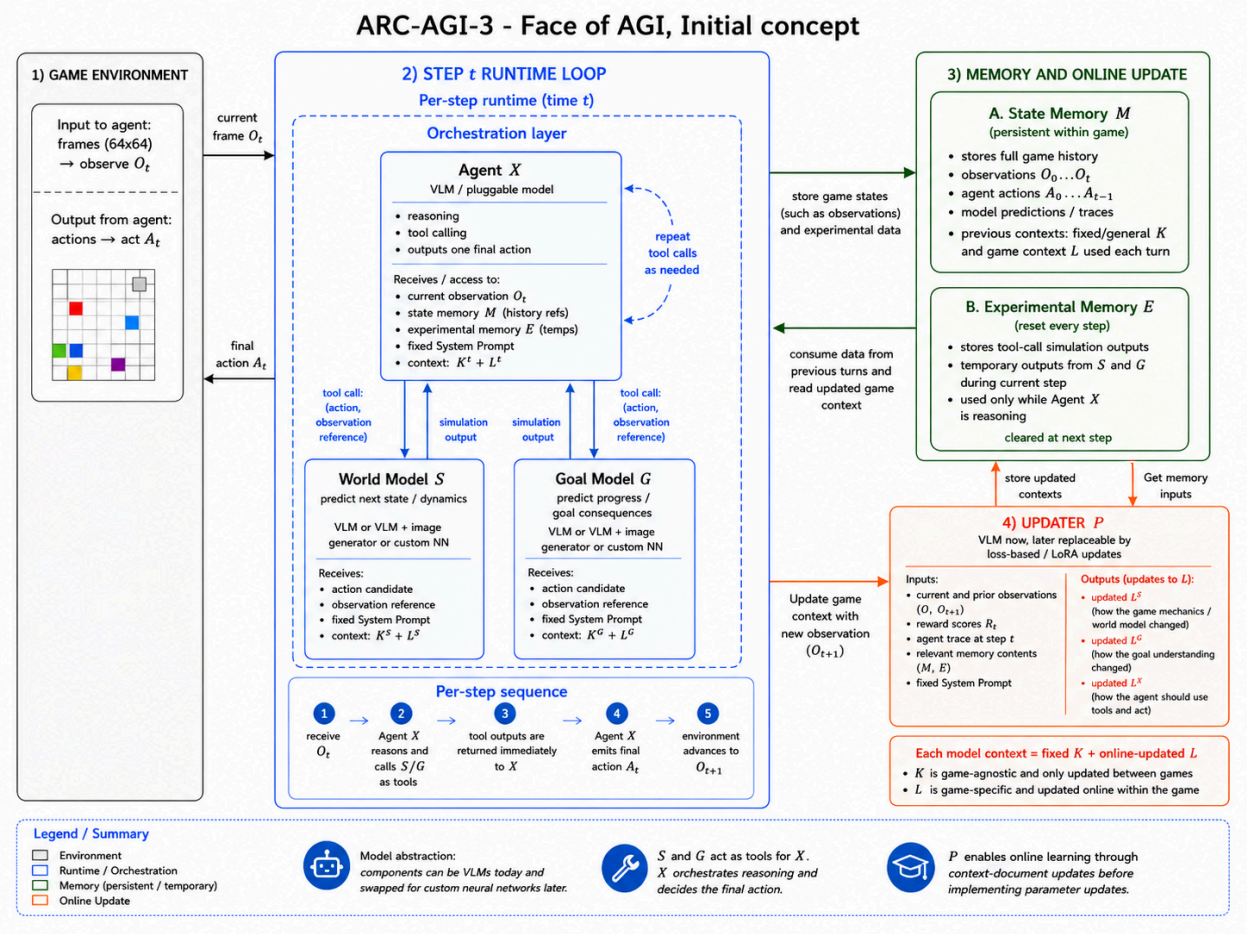
Over time, this theory had to be simplified. The cost of predicted frame generation, the limited resources available for the challenge, and the project timeline made the full version difficult to optimize. The work therefore evolved into a more practical agentic system focused on structured perception, memory, history compaction, and action selection.

We evaluated several models, including Qwen 3.6 27B, Qwen 3.6 35B, Gemma 4 variants, and Nemotron. In our experiments, the differences between models were relatively small. Since no model clearly outperformed the others in our harness, we decided to rely on available performance benchmarks and selected **Qwen 3.6 35B** as the main model.

2. Initial Architecture

The initial theory was built around three model-driven components: the **World Model**, the **Goal Model**, and the **Agent**. All three used Qwen 3.6 35B.

The Agent was the orchestrator. The World Model and the Goal Model were exposed to it as tools. At each turn, the Agent could call up to three tools, reason about the returned predictions, and choose one action from the fixed action space provided by the ARC environment.



2.1 World Model

The World Model was designed to capture game mechanics.

It received the previous observations, previous actions, the current game context, and a candidate action. It then predicted the next observation, either as an image or as a grid depending on the experiment.

After each turn, a model updater compared the previous observation, the action taken, the World Model prediction, and the real observation returned by the environment. From this comparison, it inferred what mechanic had been observed and encoded it back into the World Model context.

2.2 Goal Model

The Goal Model was designed to estimate useful progress.

It also predicted the next observation, but from the perspective of what a useful next step should look like. The goal was to estimate a plausible next state aligned with the game objective.

After each turn, a model updater compared the Goal Model prediction with the real observation returned by the environment. It used this comparison to refine the Goal Model context and improve its representation of the current game objective. The goal model was continuously aiming at finding the goal based on the observation data and the taken agent steps.

2.3 Agent

The Agent orchestrated the decision process. It could call the World Model and Goal Model as tools, compare their outputs, and choose one action from the fixed action space provided by the ARC environment.

The Agent context was updated after each turn using the latest observation, the action taken, the model predictions, and the resulting feedback signals.

The Agent received the loss/reward signal because it was responsible for selecting the action that produced the observed transition. Four signals were considered:

- **Goal Model loss:** the difference between the Goal Model predicted frame and the observed frame at a given turn.
- **World Model loss derivative:** the change in World Model prediction error across turns, used to encourage actions that exposed unknown mechanics rather than repeating already predictable behavior.
- **Action-count pressure:** a penalty based on the number of actions taken, compared with a baseline for solving a level efficiently.
- **Tool-call pressure:** a penalty based on the number of model/tool calls used per turn, added to encourage faster and more efficient decision-making.

These signals were not used for gradient-based training. They were used as contextual feedback to update the Agent's future reasoning.

2.4 Practical Simplification

The initial theory was too expensive to run efficiently under the challenge constraints.

The main bottleneck was next-frame prediction. Asking the World Model and Goal Model to produce image or grid predictions consumed too many resources, especially when repeated several times per turn. We considered more compact prediction formats, such as segmented regions or reduced grid outputs, but the expected optimization cost was too high for the timeline.

This forced the project to move away from explicit prediction and toward structured support for the Agent.

The World Model and Goal Model gradually stopped being full predictive modules. Instead, they became processors that helped the Agent understand the current state, recent changes, possible action effects, and past behavior.

3. Software Framework

The software framework was built in Python and designed to keep experiments modular. The goal was to make it easy to change models, prompts, memory strategies, and orchestration patterns without rewriting the full system.

3.1 Memory

The memory layer used SQLite. Each game had its own memory file, and each turn was stored as a new row.

The memory stored:

- observations and frames;
- actions taken;
- full model outputs;
- model contexts;
- token usage;
- intermediate summaries;
- game metadata.

The memory system also enabled replay and state restoration. This was useful for returning to a previously reached state, debugging a specific failure, or analysing how a prompt change would work in a specific, already reached situation.

3.2 Model Harness

The model harness supported several backends:

- OpenAI API;
- Ollama;
- vLLM.

This allowed us to test different models under a shared interface. Each model module had a clear input/output contract, a task-specific system prompt, and optional evolving context.

This made experimentation faster because modules could be swapped without changing the orchestration layer.

3.3 Orchestration Layer

The orchestration layer connected the ARC environment, the model modules, and the memory system.

Its role was to:

- receive the current observation;
- preprocess the frames;
- call the required model modules;
- pass structured context between modules;
- submit the selected action;
- store the full turn data in memory.

This layer acted as a simple state machine around the game loop.

3.4 Execution Environments

The system supported both local and remote execution.

Local execution was used for single-game analysis and fast debugging. Remote execution was used to run games in parallel on Modal GPUs. We also used Kaggle's available competition GPUs.

This gave us two complementary workflows: detailed inspection on one game, and broader parallel evaluation across multiple games.

3.5 Debug Dashboard

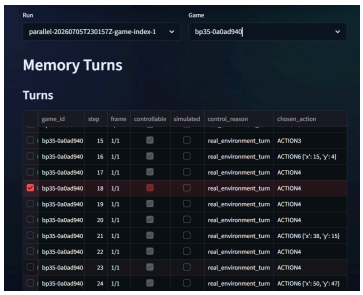
A Streamlit dashboard was built on top of the SQLite memory files.

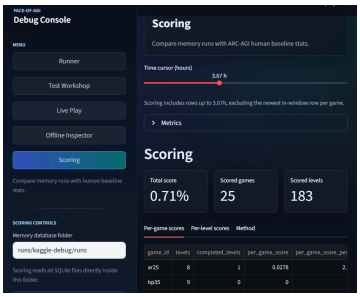
It allowed us to inspect:

- frame-by-frame evolution;
- actions selected by the Agent;
- model outputs;
- token consumption;
- context updates;
- repeated states;
- failure modes;
- arc-agi-3 benchmark scores on public games.

The dashboard became essential because many failures were only understandable through the full turn-by-turn trace.

3.6 Dashboard Overview





4. Architecture Experiments

The experiments followed a chain of simplification. We started with the full theoretical solution, then progressively replaced expensive prediction with cheaper intermediate representations.

4.1 Full Predictive Theory

The first experiment implemented the initial architecture directly.

The World Model and Goal Model were asked to predict the next observation as an image or grid. The Agent then used these predictions to reason about the next action.

This approach failed mainly because of resource consumption. The cost of producing predicted observations was too high, especially when several tool calls were allowed per turn and multiple games had to run in parallel.

Outcome: abandoned as a practical solution, but it defined the core research direction.

4.2 Text-Based Prediction

The next experiment replaced image and grid prediction with text prediction.

Instead of producing the next observation, the World Model and Goal Model described what they expected to happen. This reduced resource usage, but it also weakened the

loss mechanism. Comparing a textual prediction to an actual observation was less direct and less reliable than comparing two grids or images.

This made the original reward/loss loop much less meaningful.

Outcome: partially useful, but not enough to preserve the original theory.

4.3 Human-Like Task Decomposition

To keep the core idea of a main Agent assisted by specialized models, we shifted toward a decomposition closer to how a human would analyze an ARC game.

Instead of asking models to fully simulate the next frame, we asked them to extract intermediate information useful for decision-making.

This led to several modules: a Change Summarizer, an action-effect version of the World Model, history compactors, exploration/exploitation agents, and dynamic role experiments.

The purpose was to keep the system general enough to apply beyond one specific environment while making it practical enough to run under the challenge constraints.

4.3.1 Change-summarizer

The Change-summarizer became one of the most important modules.

Its role was to compare consecutive frames and explain what changed after an action. This exposed a major failure mode: the VLM could miss small but obvious changes between ARC frames, even when those changes were essential to understanding the game.

We tested several ways to guide the model, including bounding boxes, targeted questions about specific regions, and explicit prompts focused on changed areas. These were not reliable enough and did not bring significant improvements.

The biggest practical improvement came from adding a compressed symbolic representation of the ARC grid, especially through connected components and neighbouring cells. This gave the model structured information that was easier to compare than raw pixels alone.

We also experimented with an additional memory module which takes the full sequence.

Outcome: very useful, it was kept as a core direction.

4.3.2 World Model as Action-Effect Explainer

The World Model was then repurposed.

Instead of predicting the full next observation, it described the likely effect of available actions. This preserved part of the original world-model idea while removing the cost of full frame generation. It was working nicely in pair with a history of past the change summarizer output with the corresponding actions taken for each transition.

Outcome: partially useful as a lightweight compactor.

4.3.3 History Compaction

As runs became longer, the Agent could not receive the full raw history efficiently. We introduced history modules to compact previous actions with associated change summaries (from change-summarizer), failed strategies, observed changes, and current hypotheses.

The goal was to preserve the useful parts of the trajectory while reducing context size and avoiding unnecessary repetition.

Outcome: very useful, this was kept as a core module for our final solution.

4.3.4 Dual-Agent Exploration and Exploitation

Another recurring failure mode was behavioural repetition. The Agent could get stuck in loops, repeat failed actions, or continue a strategy even when the history showed it was not working.

To address this, we tested a dual-agent setup. One Agent was configured for exploration, and the other for exploitation. A history module decided which mode was more appropriate based on recent behaviour.

In practice, this did not clearly outperform a well-configured single Agent. It helped us understand the loop problem, but the added complexity was not justified.

Outcome: abandoned as a main solution.

4.3.5 Dynamic Agent Roles

We also experimented with dynamic role creation across games.

A supervising Agent observed gameplay and proposed specialized roles such as navigation, colour reasoning, object tracking, strategy refinement, or failure-mode

detection. The goal was to encourage cross-game knowledge transfer, the supervising agent was fully free to create any role he wanted and was not constraint. The supervising agent was allowed to create, update and delete roles freely.

Learnt roles were reusable for later game runs, we did not observe a cross-game knowledge transfer nor the usefulness of the learnt roles.

This produced results similar to the single-agent setup.

Outcome: abandoned as a main solution, but useful as a research direction.

5. VLM-level Experiments

In parallel with the system-level experiments, we also tested how the VLMs behaved under different input representations, prompt structures, and reasoning constraints. These experiments were important because many failures did not come from the orchestration logic itself, but from how the model perceived and processed ARC frames.

5.1 Image Input

We experimented with different image sizes and different visual representations of the ARC grid.

The most reliable results appeared when the image scale gave the VLM token patches a meaningful relationship with the ARC grid structure. In practice, images where one VLM token roughly covered around one to two ARC grid cells seemed to work best.

The useful range seemed to be the one where the model could still see grid-level structure clearly without wasting too much context on redundant visual detail.

5.2 History/Context Length

One key aspect that influences performance and behaviour is how much previous history the agent and updater has access to. Since we were not providing actual observation frames, history here means the actions taken and the transition text produced by the change summary VLM. Longer history increases context length and can confuse these smaller VLMs like Qwen 3.6, while shorter history omits context about what happened in the past that can be useful for future strategies or hypotheses. While we achieved good results with a fairly short history length (past 10-20 environment steps), our best results (in section 6) were observed when increasing this

to the full history, i.e. during a 9-hour run in the range of 100-200 steps. This essentially means perfect memory.

5.3 Instruction Tuning Through Prompting

We spent significant effort refining system prompts and task instructions.

More precise instructions helped at first. They improved the model's ability to reason like we would when we observed failure modes.

However, this improvement eventually reached a plateau. Past a certain point, adding more instructions did not make the model more reliable, other failure modes appeared and in some cases, the model became worse because it had too many constraints to track and started failing to respect parts of the prompt.

This suggested that prompt refinement was useful until the role of the model and the task are defined enough and the goal is clear. In order to solve the game efficiently, exploring efficiently and strategizing to find probable sub-goals become the core limit of today's LLMs / VLMs.

5.4 Multi-Frame and Animation Processing

Some ARC-AGI-3 environments return animation frames rather than only a single before/after transition.

We tested whether the model could process several frames in sequence without losing track of the relevant changes. The result depended heavily on the representation given to the model.

When the model only received raw frames, it could miss important transitions or collapse the sequence into a vague summary. However, when it also received a good intermediate representation, especially the ARC grid connected components representation, it could process multiple frames more reliably.

With the right representation, the model was able to follow several frames in a row and still produce detailed observations about what changed, what persisted, and what the transition suggested about the game mechanics.

5.5 Reasoning

Reasoning significantly improved model behaviour.

When the model was allowed or encouraged to reason explicitly, it followed system prompts better, understood the game state more accurately, and produced more useful action analysis. Reasoning helped the model connect visual changes, action history, and possible objectives.

However, reasoning also had a limit. Past a certain token budget (1-2k), the model could overthink, introduce unnecessary hypotheses, or become less decisive. In some cases, longer reasoning made performance worse. The best results came from controlled reasoning: enough to force the model to inspect the state carefully, but not so much that it drifted away from the observable evidence.

6. Solution and results on the ARC-AGI-3 June 30 competition milestone

6.1 Solution

As described before our solution is an online-learning ARC-AGI agent built around a small set of VLM roles. Here, we abandoned explicit world and goal models. Agent X chooses an action, other VLM roles explain what changed, and updater P rewrites the agent's context so the next decision can use what was learned.

All prompt-backed roles receive immutable role instructions plus mutable context. The mutable context is represented as a game-agnostic document plus a game-specific document. The immutable system prompt tells a role how to behave; the documents carry accumulated knowledge.

Observed frames are converted into two aligned model inputs. We serialize the ARC grid into exact symbols, coordinates, components, and deterministic change evidence. Cropped PNG images provide matching visual context for frame-consuming VLM roles. The text representation is authoritative when text and image appearance disagree, while the image helps the VLM inspect spatial patterns.

Agent X is the decision model. On controllable frames, orchestration gives X the composed agent context, the current observation evidence, the current valid action space, recent action history, and deterministic evidence about which actions recently changed the visible state. X returns a final action to submit to the environment.

After a frame transition is observed, the change-summary model turns raw before and after frame evidence into compact transition knowledge. It receives serialized observations, matching images, action context, component-level deltas, and deterministic changed-cell evidence. It returns a transition summary plus structured change fields. Orchestration uses that output as action-history evidence and as part of the updater input.

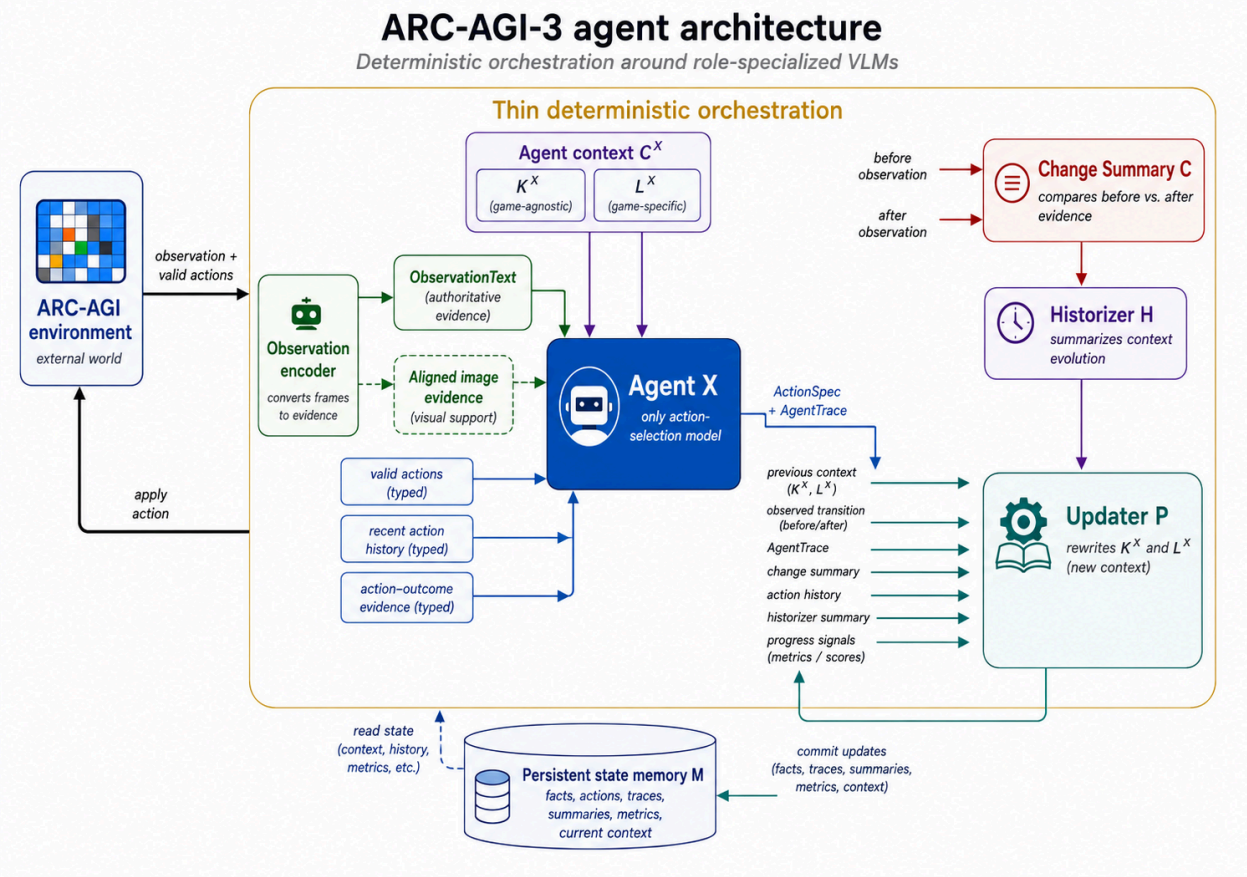
The agent context historizer is a text-only compression role. It receives recent revisions of the agent game context and summarizes how fields such as goals, game mechanics, policy, history, and extras have evolved. This prevents the updater from reasoning only from the latest context document when useful history is available.

Updater P is the online (context-level) learning mechanism in the current VLM architecture. During the game loop, the agent-game updater revises the game-context from the observed transition, change summary, recent action history, historizer summary, progress signals, and the previous agent context. Orchestration applies the returned context to the live working state before later model calls use it.

The resulting model loop is:

- encode the current observation into exact text plus visual evidence
- call Agent X to choose an action on controllable frames
- observe the real transition from the game environment
- summarize the transition with the change model
- summarize recent context evolution with the historizer
- call updater P to rewrite the agent context
- commit the new context and evidence so the next Agent X call uses it next turn

Architecture diagram



6.2 Public games and competition results

We ran this solution in parallel on the 25 public games. While other solutions often produced variable results this one was fairly stable. We observed that there were always games and levels that the framework could easily solve, while others it never managed to do so. Overall, when running for an extended period (3-4 hours on a single GPU) it was able to solve 10-11 levels, mostly 1st levels of different games with 1-2 2nd levels. The corresponding score on these games was between 0.8 and 1.0. Since the system required heavy reasoning and several models, even after running for 3-4 hours it only got to about 120 average actions taken per game. This means that it did not have enough time to progress further to later levels. This is a fundamental resource-budget imposed limit in our approach, but we do not think it is the most important one. We observed that most level completions happen early (below 70 actions on average), and then a plateau is hit, where even running for a very long time does not produce more levels solved. We believe this is a fundamental limit imposed by our core architecture

and the Qwen3.6 VLM in terms of perception/understanding and abstract reasoning. Observed failure modes seem to be centered around getting stuck in repetitive loops, not revising strategy even when new evidence and information is available.

ARC-AGI-3 scoring rewards both solving levels and doing so with action efficiency relative to human baselines. In practice, this means that solving only early levels gives a limited score ceiling, and taking many actions reduces the value of each solved level. Our agent usually solved around 10-11 public levels, mostly first levels across different games, with only a few second levels. This produced meaningful leaderboard progress, but the score remained bounded because later levels are weighted more heavily and the system's slow VLM loop often required many actions before plateauing.

Our final score on the 55 (hidden) evaluation games on Kaggle was 0.7, which put us at 8th place on the leaderboard just before the June 30 milestone cutoff. The best result at the time was 1.21, and the second best was 0.86. We also ran a few experiments on the evaluation games by lowering / increasing the reasoning budget, and adjusting history length, but these produced worse results.

7. Discussion

7.1 Negative Results

Several directions were useful scientifically but did not become the final solution. Full next-frame prediction with explicit World and Goal models was too expensive to run repeatedly under Kaggle constraints, and text-based prediction made the original loss/reward signal too indirect to be reliable. Bounding boxes, visual overlays, SAM-style preprocessing, and region-focused prompts helped debug perception errors but did not consistently fix the VLM's tendency to miss small ARC changes. Dual exploration/exploitation agents, dynamic role creation, graph-controller/belief-patcher variants, and LoRA/PPO learning-progress experiments all added complexity without clear leaderboard gains in the available time. Prompt tuning produced early improvements, but eventually plateaued: more instructions often made models less decisive or less consistent.

7.2 Lessons Learned

The most robust direction was not asking the VLM to simulate the world directly, but giving it exact structured evidence and using it as a reasoning component inside a deterministic loop. Symbolic ARC-grid serialization, component summaries, changed-cell evidence, and cropped images worked better together than visual images alone. The change summarizer and history/context compaction became central because the agent needed a compact record of what actions actually changed the environment. Online "learning" through mutable context documents was far more practical than weight updates during the project timeline. Reasoning helped, but only with controlled budgets; too little reasoning missed important structure, while too much caused drift and overcomplication. The dashboard and SQLite traces were essential to understand and iterate on key failures.

7.3 Limitations

The agent often returned to the same local behaviors even when history showed they were ineffective, such as moving back and forth, retrying no-op actions, or reusing failed strategies. The updater could add this evidence to the agent context, but the agent model did not always treat the revised context as binding, so context-level online learning influenced behavior without reliably forcing a clean change of plan. Perception also remained brittle: small cell changes, tiny object movements, color shifts, or interaction markers were easy for raw VLM image inputs to miss, and although symbolic grid text, changed-cell counts, components, and cropped images helped, the model could still underweight deterministic evidence. ACTION6 (grid clicks) added precision problems because the model had to choose exact coordinates and a target description, which led to near-miss clicks, crop/original-coordinate confusion, repeated failed coordinates, or over-avoidance of coordinate actions after failures. Finally, throughput constrained learning and evaluation: each step could require several large-VLM calls with reasoning, so the system gathered environment experience slowly and often plateaued before it could test enough alternative strategies, recover from bad hypotheses, or reach later levels.

Overall, the results are evidence that structured VLM agents can perform useful online adaptation in ARC-AGI-3, but not yet that they can robustly infer and exploit new game mechanics with human-like efficiency.

7.4 Future directions

The current ARC-AGI-3 system shows that a thin deterministic coordinator around specialized VLM roles can produce useful online adaptation, but the results also expose the main remaining bottleneck: the agent often reaches an early plateau, repeats ineffective strategies, and fails to revise its policy strongly enough from accumulated evidence. Future work should therefore focus less on additional prompt engineering and more on mechanisms that improve perception, exploration, memory use, and action efficiency.

A first direction is to improve the agent's world representation by learning compact text or program-like world models that can simulate action effects without generating full frames.

A second direction is to restore the original learning-progress idea in a cheaper and more robust form. Instead of using curiosity only as a loose contextual signal, the agent should explicitly choose actions that maximize expected improvement in prediction or compression while being penalized for time, action count, memory use, and tool calls. This would align exploration with efficient problem solving rather than with novelty alone. The broader research hypothesis is that learning progress becomes more useful when combined with realistic constraints on sensing, acting, memory, compute, and self-maintenance.

A third direction is simulation and replay. After a level or failed trajectory, the system could use a learned or program-synthesized world model to replay alternative strategies, generate auxiliary tasks, and fine-tune lightweight adapters for better action efficiency. Frontier models could also be used offline as teachers: they can play, annotate transitions, propose abstractions, and generate training traces that are distilled into smaller models through LoRA or similar adaptation methods.

A fourth direction is to move beyond fixed language-reasoning loops toward flexible token-stream agents. Instead of forcing a model to follow a rigid observe → reason → act format, the agent should be able to interleave observation tokens, action tokens, language tokens, and silent deliberation tokens. This would let the model decide when to inspect more input, when to act, and when to spend computation on internal reasoning, with each emitted token carrying an explicit time or energy cost.

A fifth direction is cross-game improvement through generated environments. Instead of relying only on the 25 public games, a coding agent could generate large numbers of ARC-AGI-like environments with known latent mechanics, increasing difficulty, and controlled variation. This would allow continuous training on action efficiency, causal

discovery, and transfer across related mechanics. A self-play or “hyperagent” setup could generate harder environments as the agent improves, producing an open-ended curriculum for fast learning rather than a fixed benchmark-tuning loop.

Finally, ARC-AGI-3 should be used as one evaluation signal inside a broader suite of constrained interactive environments. The next version of the research program should build partially observed environments with controllable bottlenecks, train agents under intrinsic learning-progress rewards, and test whether these agents become more human-like in their efficiency, exploration, and ability to generalize. Success would mean not only higher ARC-AGI-3 scores, but a clearer empirical account of which environmental constraints produce more robust, interpretable, and aligned problem-solving behavior.