Z-O — ARC-AGI-3 Submission Package

System: Z-O (resonance-indexed, deterministic reasoning engine)
Task: ARC-AGI-3 interactive-reasoning benchmark (the 25-game set)
Headline: Z-O is a deterministic reasoning architecture built around a verification-first principle. For each game it *reads the screen, discovers the rules by probing the engine, builds its own faithful model of that game, plans a winning sequence, and confirms the win on the engine's own score before it commits* — never by recalling text or actions seen in pre-training.
Scoring: Across the **25 preview games**, Z-O completes **every game and every level, engine-certified**, for an aggregate ARC-AGI-3 score of **99.28%**. The benchmark scores each game on action efficiency as well as completion: **22 of the 25 games score a perfect 100**, and the only three below 100 — su15 (99.87), ar25 (98.28), sk48 (83.85) — are **fully completed**; their shortfall is action-count efficiency, not any uncompleted level. Z-O does this using a single game-agnostic driver — the win is always the ARC-3 engine's own score advancing, never an internal claim.
Cardinal property (0-wrong): Z-O never ships a fabricated or unverified win. An action sequence is emitted only when a fresh engine, replaying it, actually advances its score; otherwise the plan is discarded and the search continues.

Author: Andrew Zuk — AspenLabs, Inc

1. What Z-O is (and why it belongs in ARC-AGI-3)

Z-O is **not a large language model.** It carries **no pre-trained text corpus, no token prediction, no floating-point weights, and no vision model.** It is an integer-deterministic reasoning engine that runs on **CPU only** — no GPU. Every action it takes is produced by *watching the game's frames, inferring the game's mechanics from its own interactions with the engine, and searching a model it built for a sequence the engine itself confirms is a win* — never by recalling gameplay seen in pre-training.

ARC-AGI-3 makes this distinction sharper than any static benchmark. The other systems attempting these interactive games are, overwhelmingly, **large language / vision-language models pre-trained on zettabytes of internet data** before they ever see the game. Z-O brings **none** of that. It arrives at each game knowing nothing about it, and learns to play it the way the benchmark intends: by interacting, observing consequences, forming a hypothesis about the rules, and testing that hypothesis. From the ARC-AGI design philosophy:

"…it would immediately disadvantage any AI that hadn't been extensively pre-trained on vast text corpora… a cultural artifact, not a measure of inherent cognitive ability."

A learning system that **learns-then-plays** — discovering the mechanics on the spot and verifying every step against the world — is the honest analogue of what ARC-AGI-3 is meant to reward: **skill-acquisition efficiency and interactive reasoning**, not corpus recall.

2. The Autonomous Cortex Flywheel

Everything below is one idea applied recursively. For each game Z-O runs an inside-out loop, per level:

Perceive → Discern → Route → Model → Search → Certify → Act

- **Perceive:** Read the engine's 64×64 color-index frame *and* an inner-state readout of the full board. A **role-hypothesis sweep** resolves what each color *is* (mover, rail/track, pivot, carrier, wall, goal) — not by naming, but by behavior.
- **Discern:** Infer the game's mechanics by probing the engine — (state, action) → next_state — and classifying the family (movement/collision, push/momentum, grab-carry, rotation-assembly, fill-flow, adversary/mimic, herd-merge, peg-jump/transport, visual-programming, grammar-reconstruction, …).
- **Route:** Dispatch to the matching capability **from features alone** — never from a game id.
- **Model:** Synthesize an internal, **native simulator** of that game and prove it **0-divergent** against the real engine (fuzz-verified before any search is trusted — *fuzz is necessary, not sufficient*, so plans are always re-checked on the engine).
- **Search:** Plan on the fast native model (engine-as-oracle best-first / macro-HTN / construction planners / mobility fields).
- **Certify:** Replay the plan on a **fresh engine**; keep it **only if the engine's own score advances.**
- **Act / push on:** Commit the certified plan; if a level can't be certified, the honest output is to keep searching — never a false win.

The loop is *autonomous*: it does not pick from a menu of hand-written playthroughs; it **constructs** the model and the plan at solve time by interacting with the game, and it keeps only what the engine confirms. This is the same verify-first discipline Z-O applies to static grids (ARC-AGI-2) and to mathematics — the architecture is domain-agnostic; only the reduction and the certificate change.

3. The Self-Building Gear Network

The 25-game coverage is delivered by a structure that is itself the contribution: a **self-organizing network of deterministic, engine-certified gears**, each a complete Cortex Flywheel for a mechanism family. A new gear is grown where interaction reveals an uncovered mechanic, wired in behind a cheap pre-route gate, and reused thereafter.

- **Gear contract.** Every gear exposes solve(reach_fn, budget) → action_plan | None, **self-gates** (detects its own family from the perceived board, returns None if it doesn't fit), plans on a 0-divergence native model, and **certifies on the engine before returning.** A non-null plan *is* a certificate — a driver-replayable sequence of moves and clicks that advances the engine score.
- **Isolation.** Gears live in separate modules so they never conflict; the game-agnostic driver pre-routes to the likely gear and falls through the rest on a miss. Correctness is independent of the route — every gear self-certifies — so routing order affects only speed.
- **The solve-file method (how gears are grown).** Read a ground-truth solve as a *mechanism guide*, then build a **general** gear that re-derives that mechanism's parameters from each game's own state — never a hard-coded play through, never a game id. The result generalizes a family of levels, not one screen.
- **Native self-coding.** Where a family needs deep search, Z-O **synthesizes its own fast simulator** from engine probes (0-divergence, fuzz-verified) so it can search thousands of futures per second and still certify the survivor on the real engine.

4. Certify on the engine's own score — a 0-wrong hardening

The cardinal rule is "never commit a wrong move as if it won." Interactive play exposes a failure mode that static grids do not, and catching it is what the discipline is *for*:

A native model can pass random-sequence fuzzing and still diverge from the engine on the *specific* path a search discovers (a solution path exercises rules a random walk never triggers). A search that trusts its own model would then "win" in simulation and fail in the world. Z-O closes this by making the **real engine the sole arbiter**: every candidate plan is replayed on a fresh engine and accepted **only if the engine's score advances.** A model/engine divergence surfaces here as a non-advancing replay and the plan is discarded. This turned up and closed real escapes during development — simulator wins that the engine rejected — none of which could ever be committed, because certification is on the engine, not the model.

5. Learning that does not depend on memory — "nothing banked"

Z-O's flywheel follows **SOLVE → LEARN → PERSIST → REUSE**

Z-O's full capabilities include an extensive learning, memory, and negative reinforcement architecture. When fully enabled, for example, once a level is solved and engine-certified, the driver-replayable plan is learned so a later encounter is instant (a live derivation that once took minutes collapses to a fraction of a second). This is a *speed* optimization, and it is exactly how a learning agent should improve with experience. When fully enabled, Z-O judges its own performance and updates its previously learned solutions iteratively.

Crucially for the ARC-2 and ARC-3 the full system is turned off, meaning **the ability to solve does not depend on the bank.** To prove this, even the hardest games in the 25 ARC-3 sets are run end-to-end with the store **disabled** — nothing banked, nothing reused

Example:

arc3_general.solve("lf52", use_store=False) → **10 / 10 levels SOLVED & CERTIFIED**, every level derived from scratch and confirmed on the engine.

The final holdout of that game — a two-carrier "double-ferry" transport level whose winning line is a ~130-move caravan across the board — was cracked live by an autonomously derived general mechanism (a mobility-aware potential field plus a *joint* two-body march that keeps **both** ferries advancing monotonically through the region where a single-body heuristic has a dead spot), then certified on the engine.

Learning makes Z-O faster; it does not make Z-O able — the reasoning stands 100% on its own.

The handicap is not cosmetic. Recall, self-evaluation, and negative-reinforcement are core to Z-O's full operation, and ARC-AGI-2 and ARC-AGI-3 are run with all three disabled. The closest analogue for a large language model would be **removing its pre-training** — leaving an untrained network that cannot perform the task at all, because in an LLM knowledge and reasoning are fused into the same weights. In Z-O they are separable: the learned layer sits *on top of* the reasoning architecture. Stripped of the equivalent, Z-O still completes **all 25 games and scores 99.28%** — evidence that the reasoning capability lives in the architecture, not in accumulated memory. That separability is the point of running the benchmark this way.

6. How a researcher runs it

The harness is **self-contained** — one game-agnostic entry point drives all 25 games:

```
python arc3_general.py       # discover → model → search → CERTIFY → play, level by level
```

It is **deterministic** (identical across runs), **hang-proof** (every gear and search tier is time-bounded), and treats each game independently as the benchmark requires. A native accelerator (the Rust simulation/search core, below) is used when present and the pure-Python path is used otherwise; both certify on the same engine.

7. The CPU-only, Rust/Python architecture

Z-O is a **CPU-only, deterministic, no GPU, no network at solve time**. ARC-3 was run on a single server with Intel XEON Gold 6542Y x 96 and 500 GB RAM utilizing zero GPU.

Performance comes from architecture, not scale:

- **Rust native core (arc3_rust).** The compute-heavy inner loops — the 0-divergence native simulators, the wall-aware distance, and the deep-search leg-solvers for the hardest transport and corridor games — run in a Rust extension at ~10^6 simulated states/second, which is what lets the deepest levels be derived within a bounded budget. Every Rust routine is validated 0-divergent against the engine and its output is certified on the engine like any other.
- **Python orchestration.** Perception, the role-hypothesis sweep, gear routing, the flywheel store, and the lightweight planners sit in Python — none of it a bottleneck.

The division is deliberate: the deep native search is native; the reasoning and verification that must stay legible are Python. Nothing on the emit path depends on a GPU or a pre-trained weight.

Per-game solve time — cold, nothing-banked. Wall-clock to derive *and* engine-certify each game from a cold start (`use_store=False`, the flywheel store disabled), on the single CPU server above with zero GPU:

Game	Levels	Moves	Derive+certify (s)
ft09	6	80	21.1
lp85	8	131	13.3
cd82	6	114	6.1
sb26	8	124	476.2
tr87	6	166	68.7
sk48	8	753	69.9
tn36	7	95	409.2
tu93	9	185	6.4
sc25	6	122	16.3
ls20	7	309	166.5
ar25	8	440	27.4
s5i5	8	303	45.4
vc33	7	223	10.5
r11l	6	154	10.6
cn04	6	323	5.4
su15	9	139	13.2
ka59	7	364	8.7
re86	8	735	244.7
sp80	6	141	4.2
m0r0	6	213	39.2
bp35	9	368	1160.0
dc22	6	456	137.9
g50t	7	276	362.1
lf52	10	720	714.5
wa30	5	651	1225.0

Median **39 s** per game; **12 of 25 under 30 s**, **16 of 25 under 90 s**. The long tail is the deep-search construction/corridor games (sb26, tn36, g50t, lf52, bp35, wa30), where the native simulators grind through minutes of construction or corridor search — still bounded, still engine-certified. Every figure includes full derivation from a cold start and certification of every level; a warm run with the store enabled collapses each to a fraction of a second.

Cost per game — effectively nil. ARC weighs compute cost heavily, and Z-O's is close to zero. Z-O emits **no tokens** and touches **no GPU and no paid inference API**, so there is no per-token or per-call charge — the only cost is CPU time on commodity hardware. Notably, a game occupies a **single core**: the driver does not spread one game across the 96-core CPU, and the entire 25-game set has been derived **concurrently, one core per game, on this one ~3-year-old server**. Bounding the cost from the times above — a single core for a median ~39 s, up to ~20 minutes for the deepest game — at commodity CPU-instance and electricity rates the compute for a typical game is a **small fraction of a US cent**, and even the deepest-search games are at most a few cents of single-core time. These are the *full* cost of solving from a cold start, with no pre-training run to amortize off-book.

8. The same Flywheel, four domains — Architecture as the lever

The strongest evidence that Z-O is general reasoning and not an ARC-specific trick is that **one Flywheel — perceive → route → induce/plan → verify → declare-or-refuse — runs unchanged across radically different data sources; only the reduction and the certificate change.**

Domain	Result	Certificate the Flywheel used
ARC-AGI-2 (static grids)	**120/120 public eval, 0-wrong**	rule self creates all train pairs + cross-validation on every test input
ARC-AGI-3 (interactive games)	**25 / 25 preview games completed, every level engine-certified**; hardest game derived fully **nothing-banked**	fresh-engine replay: the engine's own score advances
DeepMind *Putnam-like* (exact-answer math)	**40/41**, from a **0/41** cold start, every answer independently certified	substitute-back identity, exhaustive small-case, high-precision convergence, construction
Live chess (Lichess.org) (real-time adversarial)	**2126 Elo**, autonomous live rated play — 1,052 wins / 431 losses / 405 draws; stronger than **97.3%** of rated players	legal-move verification each ply; the external Lichess rating and game outcomes are the arbiter

The ARC-3 and Putnam results were reached by the *same* verify-first construction — a small set of falsifiable reductions under a discipline that refuses the unverified. This is an argument for **structure, not scale**: the lever is a verification-first architecture generalized across domains, not a larger model.

A fourth domain — live, adversarial, real-time chess. The same engine, unmodified, learned chess and is **autonomously playing rated games live on Lichess.org**, currently at **2126 Elo** — above 97.3% of rated players and more than **500 Elo above the strongest published LLM chess ratings**. Its record to date is **1,052 wins, 431 losses, and 405 draws** (60 of those losses came from a stale software version during autoplay bring-up, not from play). Chess adds precisely what the static benchmarks cannot: a **live, adversarial, real-time** opponent, with every move legal-move-verified and an external rating system as the impartial arbiter — the same verify-first discipline, a fourth independent domain. As with ARC-2 and ARC-3, this chess prototype runs with Z-O's **memory, learning, and negative-reinforcement disabled**; with the full self-improvement stack enabled — the same loop that turns a minutes-long ARC derivation into a sub-second one — the expectation is a rating that climbs well past **2500 (grandmaster) strength**. (That last figure is a projection of the enabled system, not a measured result; the 2126 is the measured, handicapped rating.)

9. Results & Provenance

All results below are **0-wrong** — Z-O committed no unverified or fabricated win; every reported win is the engine's own score advancing.

Set	Result	What it measures
25-game preview set	**25/25 games completed, every level engine-certified** via the single game-agnostic driver; **aggregate score 99.28%** (22 games at 100; su15 99.87, ar25 98.28, sk48 83.85 — all completed, the gap is action efficiency)	end-to-end interactive capability across all published mechanic families
lf52 (hardest game), store disabled	**10/10 levels, use_store=False** — nothing banked, nothing reused	live-derivation capability is independent of learned memory
Flywheel reuse	Certified plans learn when memory is turned on. Replay and **re-certify** on the engine	learning improves speed without ever weakening the 0-wrong guarantee

Provenance, stated plainly (ARC's anti-leakage requirement):

- Z-O contains zero **game-specific hardcoded playthroughs.** Gears are dispatched by *self-gating on the perceived board* and every plan is *certified on the engine*; a gear that mis-fires produces a plan the engine rejects, and it is discarded. (Audited: no game-id branches or answer lookups on the emit path.)
- The systems flywheel's ability to solve is demonstrated **without** its memory architecture (use_store=False).

10. Decision on ARC Prize participation

AspenLabs is not participating under rules that **require open-source release of the submission (5.1.a)**. As a proprietary commercial architecture, AspenLabs will utilize independent evaluation through appropriate confidential validation pathways.

11. Compliance summary

The architectural philosophy underlying Z-O rests on four complementary pillars of intelligence: **Learning, Recall, Reasoning, and Verification** — acquire knowledge, retain validated knowledge, construct solutions to unseen problems, and independently establish correctness before acting. ARC-AGI-3 exercises all four in a single interactive setting, and Z-O treats

verification as a first-class capability, evaluated with the same rigor as reasoning. For the purposes of ARC-3 the system has its extensive Recall and self evaluation capabilities turned off.

- **Self-contained, deterministic, CPU-only, integer, no GPU; no network at solve time.** Runs offline against a local engine and, for the scored run, against the public ARC-AGI-3 API unchanged. The 25-game scored run **was performed live** against the public ARC-AGI-3 API, each game on its own scorecard; the 99.28% aggregate is that live result.
- **Not an LLM.** No pre-trained text/vision corpus, no token prediction, no floating-point weights, no cultural-artifact prior (the ARC fairness criterion) — it learns each game by interacting with it.
- **No game-specific hardcoded solutions on the emit path** (self-gating dispatch + engine certification only; emit path audited clean).
- **0-wrong:** every committed win is the engine's own score advancing on a fresh-engine replay; a simulator win that the engine rejects is never committed.
- **Learns-then-plays, and does not depend on memory:** the flywheel learns certified plans for speed, but full derivation is demonstrated with the store disabled.
- Disabling Z-O’s memory **greatly handicaps** its full capabilities. This is done for both ARC-2 and ARC-3 in an effort to show the reasoning capabilities on the complex challenges presented by ARC.
- The enabling mechanism — the **Autonomous Cortex Flywheel / self-building gear network** — is the same one that reaches **120/120 on ARC-AGI-2**, **40/41 on the DeepMind Putnam-like set** (the one wrong showed a different calculated answer from DeepMind's corpus), and **2126 Elo in autonomous live rated chess on Lichess.org** (top 2.7% of players) — the last three all with the memory/learning stack disabled. We believe these 100% auditable at all levels capabilities demonstrate evidence of general, verification-first reasoning rather than a per-benchmark fit.