Ralpha: Parallel LLM-Orchestrated Scene Optimization Through a Ground-Truth World Model

James Tagg

Ralpha, 2026

Abstract

We present Ralpha, a system that uses large language models (LLMs) to iteratively optimize 3D scenes in Unreal Engine 5 (UE5) to match photographic references. Rather than learning a world model — the central challenge of Joint Embedding Predictive Architectures (JEPA) — Ralpha uses the physics engine itself as a deterministic, zero-error world model, transforming the planning problem from representation learning into orchestration. The system implements a six-phase state machine (PLAN → ACT → OBSERVE → SCORE → REVISE → repeat) driven by 50+ domain-specialist LLM agents that collectively control 2,300+ rendering parameters through 218 registered command handlers. We describe four contributions: (1) a **composite perceptual-semantic scoring pipeline** combining color histograms, CLIP/SigLIP embeddings, LPIPS perceptual distance, and region-aware decomposition; (2) a **convergence guard** that adapts command budgets, revert thresholds, and parameter isolation based on optimization trajectory; (3) a **parallel hypothesis exploration** framework that fans out competing parameter strategies across N UE5 instances, scoring and selecting winners in parallel; and (4) a **shared learning bus** — a lock-free, append-only knowledge transfer mechanism that enables concurrent brain loops working on different visual domains (sky, fog, lighting, post-processing) to cross-pollinate discoveries without coordination. The system further includes a meta-cognitive evolution loop that uses structured multi-perspective deliberation to evolve specialist prompts through A/B-tested revisions validated on benchmark suites. We describe the architecture in sufficient detail for reproduction, analyze the design tradeoffs, and situate the work relative to JEPA, TextGrad, Reflexion, and the broader landscape of LLM-driven optimization.

---

1. Introduction

The dominant paradigm for AI-driven 3D content creation is generative: diffusion models, NeRFs, and Gaussian Splatting produce novel views or geometries from learned priors. These approaches excel at creation from nothing but struggle with *precision* — matching a specific reference photograph, respecting physical constraints, and producing outputs that are editable, composable, and production-ready.

An alternative paradigm, which we call **orchestrated simulation**, uses AI not to generate content directly but to *control an existing physics simulator*. The simulator provides ground-truth rendering; the AI's role is to find the simulator parameters that produce a desired visual result. This reframes the problem from generation to search — specifically, search through a high-dimensional parameter space guided by perceptual similarity to a reference.

Ralpha implements this paradigm using Unreal Engine 5 as the simulator and LLM agents as the search controller. The core loop is simple:

1. **PLAN**: An LLM specialist examines the current render and reference, then proposes parameter changes
2. **ACT**: Changes are applied to UE5 via MCP (Model Context Protocol) commands
3. **OBSERVE**: A screenshot is captured from the engine
4. **SCORE**: The screenshot is compared to the reference using composite metrics
5. **REVISE**: The system decides whether to keep changes, revert, or escalate

This loop repeats until the score converges, the iteration budget is exhausted, or cost limits are reached. The simplicity is deceptive — the engineering challenges lie in scoring, convergence control, parallelization, and the organization of domain knowledge across dozens of specialist agents.

1.1 Relationship to JEPA

Yann LeCun's Joint Embedding Predictive Architecture [1] proposes that intelligent systems require a *world model* that predicts future states from current observations and proposed actions. The key insight of Ralpha is that for any domain where a physics simulator exists, the world model is already built. UE5 computes exact future states — no prediction error, no representation collapse, no hallucination. The open problems of JEPA (representation collapse, grounding, cost function bootstrap) are eliminated by construction when the world model is a deterministic simulator. A companion paper [Tagg, 2026] explores this mapping in detail; here we focus on the systems architecture.

1.2 Contributions

This paper describes the Ralpha system in sufficient detail for reproduction, with emphasis on four areas that we believe represent novel contributions:

1. **Multi-metric scoring with adaptive weighting** (§3): A four-component scoring pipeline (traditional perceptual, CLIP semantic, LPIPS learned perceptual, region-aware spatial) with weights that adapt to component availability and user preferences.
2. **Convergence guard** (§4): A trajectory-aware control layer that modulates command budgets, revert sensitivity, and parameter isolation based

on the optimization's momentum, preventing oscillation and protecting gains.

3. **Parallel hypothesis exploration** (§5): A multi-instance architecture that distributes competing hypotheses across N UE5 instances for simultaneous rendering and scoring, with strategy diversity (conservative, creative, control, contrarian).
4. **Cross-domain shared learning** (§6): A lock-free append-only bus that enables concurrent optimization loops working on different visual domains to share discoveries, creating emergent cross-pollination without explicit coordination.

---

2. System Architecture

2.1 Overview

Ralpha consists of three layers:

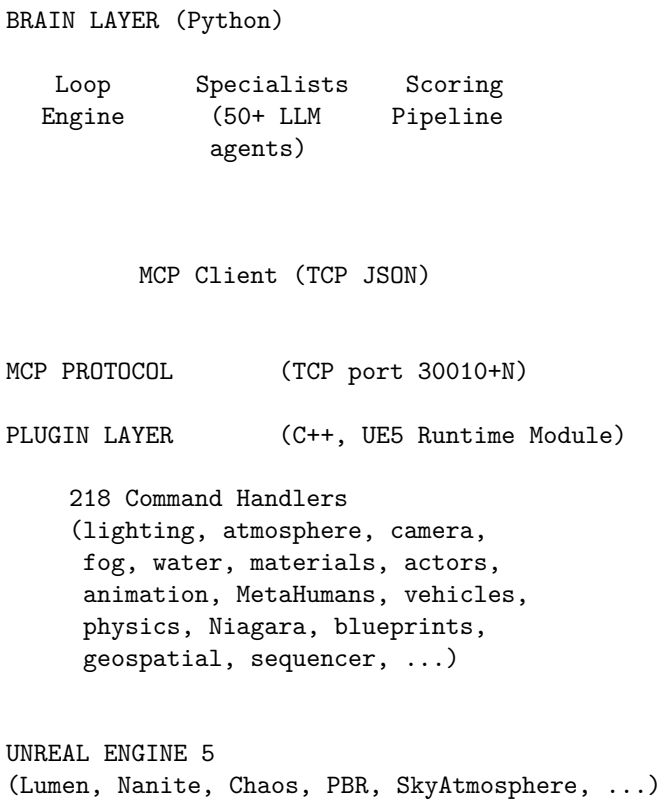

Figure 1. Three-layer architecture. The brain layer runs in Python and communicates with the UE5 plugin via TCP JSON commands. The plugin is a "dumb pipe" — it applies parameter changes without clamping, filtering, or second-guessing the brain.

2.2 The MCP Protocol

Communication between brain and engine uses a custom TCP-based JSON protocol. Each command is a JSON object with a `type` field identifying the handler and additional fields for parameters:

```
{"type": "set_directional_light", "pitch": -15.0, "yaw": 220.0, "intensity": 8.0}
```

The plugin processes commands synchronously and returns JSON responses. A critical design principle is that **the plugin is a dumb pipe**: all intelligence, constraints, and parameter decisions reside in the brain layer. The plugin never clamps values, overrides ranges, or applies heuristics — this ensures that the brain has full, unmediated control of the simulation, and that behavioral changes need only be made in one place.

The system exposes 218 registered command handlers spanning: directional/sky/point/spot lighting, SkyAtmosphere, ExponentialHeightFog, VolumetricClouds, camera (position, rotation, lens, depth of field, motion blur), post-processing (exposure, color grading, bloom, tone mapping), water bodies (ocean, lake, river), materials, actor management (spawn, delete, transform, visibility), animation and sequencer, MetaHuman characters (appearance, clothing, expressions), vehicles, physics (Chaos), Niagara particle effects, blueprints, foliage, Cesium geospatial tiles, level streaming, and media playback.

2.3 The Brain State Machine

The brain operates as a state machine with six phases:

```
INIT → DECOMPOSE → [PLAN → ACT → OBSERVE → SCORE → REVISE]* → DONE
```

INIT establishes the scene: loads the appropriate level, applies geo-location if specified, sets up camera position from reference analysis, and retrieves warm-start parameters from scene memory.

DECOMPOSE analyzes the reference image using vision models to identify constituent elements (sky type, cloud coverage, weather conditions, time of day, key objects) and produces a structured decomposition that guides specialist selection.

PLAN is the core intelligence phase. The orchestrator selects 1–2 specialist agents based on the current score breakdown, injects evidence from prior iterations and peer sessions, and calls the LLM to propose MCP commands. The specialist sees: the reference image, the current render, the score breakdown, evidence of what worked and what regressed, deterministic tool outputs (sun position, exposure calculation), and any user directives.

ACT sends the proposed MCP commands to UE5. Commands are validated but not filtered — if the specialist proposes it, the engine executes it.

OBSERVE captures a screenshot and queries scene state.

SCORE computes the composite similarity score between the capture and the reference (detailed in §3).

REVISE evaluates whether to keep changes, revert to the best-known state, enter parameter isolation mode, or escalate to a different specialist. The convergence guard (§4) drives this decision.

2.4 Specialist Agent Architecture

Rather than prompting a single LLM with all 2,300+ parameters, Ralpha decomposes action selection into domain-specialist agents. Each specialist is an LLM (Claude, Gemini, GPT-4, or Grok — configurable per deployment) with a structured markdown prompt containing:

1. **Domain physics**: The physical principles governing its domain (e.g., the sunset specialist knows that low sun angles increase Rayleigh path length, shifting sky color from blue to orange-red)
2. **Command vocabulary**: The 10–50 MCP commands relevant to its domain, with parameter names, types, and typical ranges
3. **Pitfall registry**: Known failure modes accumulated from past runs (e.g., "fog density > 0.005 obscures the horizon in most scenes")
4. **Evidence format**: A template for how iteration history is injected

A **router agent** examines the per-dimension score breakdown and selects the specialist whose domain has the greatest improvement potential. The routing follows a phased strategy:

- **Early iterations**: Specialists are activated in a domain-defined order (e.g., atmosphere before post-processing)
- **Mid iterations**: Score-driven routing targets the weakest dimension
- **Stagnation**: Escalation to reasoning specialists, then meta-debuggers, then contrarian agents that challenge assumptions

The system currently maintains 50+ specialist agents. New specialists can be generated automatically by the meta-cognition system (§7) when gap detection identifies dimensions with no coverage.

2.5 Deterministic Tool Registry

A key architectural decision is the separation of **deterministic computation** from **LLM reasoning**. The tool registry catalogues 26+ physics-based tools that provide exact answers for quantities an LLM would otherwise guess:

Tool	Category	What It Computes
`exposure`	physics	EV100, aperture, shutter speed from sun elevation
`sun_position`	physics	Solar azimuth/elevation from lat/lon/datetime
`horizon`	physics	Horizon line position from camera height and pitch
`lighting`	lookup	Color temperature, intensity recommendations
`tides`	api	Tide height from location and time
`terrain_elevation`	api	Ground elevation from coordinates
`cloud_presets`	lookup	Volumetric cloud parameters by cloud type
`camera_solver`	vision	Camera pose estimation from reference image

Tools are lazy-loaded (many depend on numpy/torch), called by name via `call_tool("exposure", sun_elevation=5.0)`, and prioritized by category: physics > lookup > api > vision. The design principle is: **never let an LLM guess a value that can be computed**.

---

3. Composite Perceptual-Semantic Scoring

3.1 Design Goals

The scoring pipeline must satisfy three constraints: (1) capture both low-level fidelity (exposure, color balance) and high-level semantic content (scene type, composition); (2) provide per-dimension breakdowns to guide specialist selection; (3) operate fast enough to score every iteration without bottlenecking the loop.

3.2 Four-Component Pipeline

The scoring pipeline combines four components, weighted by availability:

Traditional perceptual metrics (weight 0.25): - Color histogram intersection (per-channel RGB, 32 bins per channel) - Brightness similarity (mean luminance L1 distance) - Structural gradient correlation (Sobel edge map Pearson correlation)

Semantic similarity (weight 0.30): - SigLIP/CLIP embedding cosine similarity - Encodes reference and render into 512-dimensional vectors - Captures high-level content: “this is a sunset over water” matches even if pixel-level details differ

Learned perceptual similarity (weight 0.25): - LPIPS (Learned Perceptual Image Patch Similarity) [17] - Trained on human perceptual judgments - More aligned with human perception than SSIM or MSE

Region-aware spatial scoring (weight 0.20): - Divides the image into semantic regions: sky, horizon band, foreground, highlights - Scores each region independently - Reports the weakest region, enabling targeted specialist selection

Composite formula:

```
S = w_trad × S_traditional + w_sem × S_semantic + w_perc × S_perceptual + w_reg × S_region
```

Weights are resolved dynamically based on which components are available (e.g., if LPIPS is not installed, traditional and semantic weights are redistributed to 0.60 and 0.40 respectively).

3.3 Cloud-Specific Scoring

For scenes with significant cloud presence, an additional cloud-specific scorer computes: - Coverage IoU (intersection-over-union of cloud masks) - Cloud color similarity (mean color of cloud regions) - Spatial distribution match (where clouds appear in the frame)

This score is used as advisory guidance for the cloud specialist rather than incorporated into the composite score, avoiding double-counting.

3.4 Preference Learning

The scoring system supports user preference overrides via a `PreferenceLearner` module. Users can specify dimension priorities (e.g., "sky match matters more than foreground") which manifest as weight multipliers on the region-aware component. Preferences persist across sessions via warm-start profiles keyed by domain.

---

4. Convergence Guard

4.1 The Problem of Score Oscillation

Iterative optimization through an LLM actor is inherently noisy. Unlike gradient descent, which follows a smooth loss surface, LLM-proposed changes can be large, multi-dimensional, and unpredictable. A specialist that improves sky color may simultaneously degrade exposure. Two specialists alternating focus can cause the score to oscillate without net progress.

4.2 Trajectory-Aware Control

The convergence guard sits between PLAN and ACT as a filter, and between SCORE and REVISE as a state tracker. It enforces five mechanisms:

1. Command Budget Scaling. The number of MCP commands allowed per iteration scales inversely with the current score:

Score Range	Max Commands	Rationale
< 0.40	12	Far from target: explore freely
0.40–0.70	6	Making progress: moderate changes
0.70–0.85	3	Close: careful adjustments
≥ 0.85	1	Very close: surgical, one-at-a-time

This draws from the explore-exploit tradeoff: early iterations benefit from broad exploration, while late iterations require precision. The chess analogy is deliberate — "before making a move, evaluate the position."

2. Graduated Revert Thresholds. The tolerance for score regression tightens as the best score increases:

Best Score	Max Allowed Regression
< 0.50	10% (tolerate large swings while exploring)
0.50–0.80	5% (protect moderate gains)
≥ 0.80	3% (protect hard-won precision)

If a score drop exceeds the threshold, the system immediately reverts to the best-known state rather than attempting correction.

3. Oscillation Detection. The guard examines the last N score deltas (default N=6). If more than 50% are sign changes (improvement followed by regression, or vice versa), the system is declared oscillating and enters parameter isolation mode.

4. Parameter Isolation. When oscillation or stagnation (3+ iterations without improvement) is detected, the guard forces single-parameter changes. The specialist is instructed to propose exactly one modification, isolating its effect. This is analogous to ablation studies in ML research — varying one thing at a time to determine causal effects.

5. Safe Zone Protection. When the overall score exceeds 0.85, the guard identifies "done" dimensions (individual region scores > 0.90) and applies a penalty to commands that modify those dimensions' parameters. This is soft protection (35% effectiveness penalty) rather than hard freezing, to allow escape from local optima. If stagnation persists for 4+ iterations, all protections are temporarily removed.

4.3 Convergence Strategy Injection

The guard generates tactical advice that is injected into the specialist's prompt context:

```
[CONVERGENCE GUARD] Score trajectory: 0.61 → 0.68 → 0.72 → 0.71 (regression)
Best: 0.72 (iteration 3). Budget: 3 commands max.
Advisory: sky dimension is strong (0.88), horizon is weak (0.52).
Do NOT modify sky parameters - focus exclusively on horizon band.
```

This provides the LLM with optimization-aware context without constraining it algorithmically.

---

5. Parallel Hypothesis Exploration

5.1 Motivation

Sequential iteration — one hypothesis, one render, one score — is the bottleneck of the system. A single brain loop spends most of its time waiting: waiting for the LLM to generate commands (~3–10s), waiting for UE5 to render (~0.5–2s), waiting for scoring (~1–3s). The insight is that these waits can be overlapped across multiple hypotheses if multiple UE5 instances are available.

5.2 Multi-Instance Manager

The `MultiInstanceManager` orchestrates N UE5 instances, each running the MCP plugin on a different port (30010 + N) on the same or different machines. It exposes two exploration modes:

Sequential round-robin: Distributes hypotheses across instances in order. Hypothesis i goes to instance $i \bmod N$. Simple, deterministic, good for even workload distribution.

Parallel fan-out: All hypotheses are dispatched simultaneously using `asyncio.gather()`. A semaphore limits concurrency when hypotheses exceed instances. Each instance renders independently, and results are collected when all complete.

```
Hypotheses [A, B, C, D]



Instance 0  Instance 1  Instance 2  Instance 3
render A    render B    render C    render D
score A     score B     score C     score D
```

```
Pick winner (highest score)
Commit winner's commands
```

Figure 2. Parallel fan-out: N hypotheses rendered and scored simultaneously across N instances.

5.3 Strategy Diversity

When running the parallel evolution coordinator across 30 instances, strategies are assigned with deliberate diversity:

Strategy	Allocation	Behavior
Analytical	60%	Conservative optimization — small, safe parameter changes
Creative	20%	Creative exploration — larger, riskier modifications
Control	10%	Baseline reference — no changes, measures natural variance
Contrarian	10%	Opposite direction — actively moves away from consensus

This distribution draws on the broader tradition of structured parallel thinking — assigning distinct cognitive roles to ensure that the population explores diverse regions of parameter space rather than converging prematurely [18, 29].

5.4 Checkpoint-Based State Restoration

All instances start from the same saved level state. After exploring a hypothesis, the instance restores to the checkpoint before receiving the next hypothesis. This enables rapid iteration: the cost of exploring a bad hypothesis is only the render time plus a state restore, not a full scene rebuild.

5.5 Pipeline Mode: Overlapping SCORE and PLAN

Within a single instance, the system can overlap the SCORE phase of iteration N with the PLAN phase of iteration N+1, using a thread pool with two workers. The PLAN thread uses the score from iteration N-1 (one iteration stale) but the current screenshot. Score drift is monitored: if the actual score differs from the assumed score by more than 5%, a warning is logged but execution continues. This yields approximately 25% speedup per iteration at the cost of occasionally planning from stale information.

6. Cross-Domain Shared Learning

6.1 The Problem

When multiple brain loops work in parallel on different domains of the same scene (e.g., sky, fog, lighting, post-processing), they operate independently by default. But their domains interact: fog affects how sky color appears; lighting affects how materials render; post-processing affects everything. Discoveries in one domain are valuable context for others.

6.2 The Shared Learning Bus

The bus is an append-only JSONL file at `~/ralpha/shared_learning/learning_bus.jsonl`. Each brain loop:

1. **Publishes** findings after every SCORE phase — breakthroughs, regressions, parameter discoveries, stagnation signals
2. **Reads** peer findings before every PLAN phase — findings from all *other* sessions in the last N minutes (default 10)
3. **Injects** peer findings into the specialist prompt as additional context

Each finding is a structured record:

```
{
  "session_id": "ralph_sky_001",
  "domain": "SKY",
  "iteration": 12,
  "timestamp": 1710400000.0,
  "category": "breakthrough",
  "description": "sun pitch -8° with temperature 2800K produced strong golden hour effect",
  "score_delta": 0.12,
  "parameter": "sun_pitch",
  "value": -8.0,
  "confidence": 0.95
}
```

6.3 Design: Chaos as Feature

The bus is deliberately lock-free and uncoordinated. POSIX guarantees that writes smaller than PIPE_BUF (4096 bytes) are atomic, so concurrent appends do not corrupt each other. There is no ordering, no consistency protocol, no leader election. Each loop reads what happens to be there, filtered to its own recency window and excluding its own session.

This design mirrors the communication pattern in biological neural populations, where neurons broadcast signals without point-to-point coordination, and downstream neurons integrate whatever signals happen to arrive. The "chaos" is functional — it ensures that discoveries propagate between domains with low

latency and zero coordination overhead, at the cost of occasional stale or redundant information.

6.4 Active Session Tracking

A separate `active_sessions.json` file (protected by file locks) tracks which sessions are currently running, their domains, iteration counts, and heartbeat timestamps. This enables dashboards and monitoring but does not affect the bus's operation.

---

7. Meta-Cognitive Evolution

7.1 The Evolution Loop

Above the per-scene optimization loop sits a meta-cognitive layer that evolves the *system itself* — specifically, the specialist prompts that drive optimization. The evolution loop operates on a longer timescale (hours to days, vs. seconds per iteration):

`BASELINE → REFLECT → ACT → VALIDATE → COMPARE → [PROMOTE | REVERT]`

BASELINE runs the current system on a benchmark suite of reference images spanning diverse scene types (sunsets, interiors, cityscapes, ocean, weather).

REFLECT analyzes benchmark results using structured multi-perspective deliberation — six sequential cognitive modes, each implemented as a separate LLM call with a dedicated prompt [18, 29, 30]: - **Factual**: Data analysis — score distributions, per-specialist performance - **Intuitive**: Gut-check assessment — what "feels" wrong about the results - **Critical**: Risk assessment — what could go wrong with proposed changes - **Optimistic**: Opportunity assessment — what gains are available - **Creative**: Novel proposals — new approaches and experiments - **Integrative**: Synthesis — combining all perspectives into actionable recommendations

This separation ensures that divergent and convergent thinking alternate systematically rather than collapsing into a single undifferentiated analysis [18].

ACT implements the synthesized recommendations through an action engine. Allowed actions include: revise a specialist prompt, adjust scoring weights, add a constraint, create a new specialist, or modify convergence guard parameters. All actions are logged with before/after diffs.

VALIDATE re-runs the benchmark suite with the modified system.

COMPARE computes the delta between baseline and validation scores. Statistical significance is assessed per-benchmark.

PROMOTE or REVERT: If the modification improves aggregate scores without causing regressions on any individual benchmark beyond a threshold, it is

promoted. Otherwise, it is reverted.

7.2 A/B Testing of Specialist Prompts

Individual specialist prompt revisions are validated through controlled A/B testing. When a revision is proposed:

1. Two variants are registered: A (current) and B (proposed)
2. Incoming optimization jobs are assigned deterministically: odd job IDs get variant A, even get variant B
3. After sufficient sample size, median improvement rates are compared
4. The winner is promoted; the loser is archived with full evidence

Promotion is automatic: the winning prompt replaces the current prompt file, and the previous version is archived with a timestamp.

7.3 Guardrails

The evolution system includes explicit safety mechanisms: - **Kill switch**: A file (`~/ralpha/evolution/KILL`) that immediately halts all evolution activity - **Pause gate**: A file that suspends evolution while allowing manual inspection - **Budget limits**: Maximum LLM spend per cycle, maximum cycles per day - **Rollback archive**: Every modified specialist prompt is archived before replacement - **Escalation**: Repeated failures trigger human notification

---

8. Autonomous Outer Loop

8.1 Retry-Diagnose-Adapt

Above the inner optimization loop sits an **outer loop** (`brain/outer_loop.py`) that removes manual intervention between runs. When the inner loop completes — whether by convergence, stagnation, budget exhaustion, or failure — the outer loop diagnoses the outcome and decides whether to retry with an adapted strategy.

The diagnosis examines the score trajectory to classify the failure mode:

Pattern	Detection	Adaptation
Oscillation	>60% of score deltas are sign changes	Reduce exploration, lock best parameters
Plateau	Last 4 iterations within 0.02 score range	Try new strategy, increase stagnation limit
Early peak regression	Best score occurred in first third of iterations	Restore best checkpoint, conservative changes

Pattern	Detection	Adaptation
Low convergence	Converged but score < 0.85	Raise convergence threshold, retry with harder target

8.2 Strategy Escalation

The outer loop escalates along multiple axes:

1. **LLM tier**: haiku → sonnet → opus (each more capable but more expensive)
2. **Iteration budget**: Increase max_iterations (up to 100)
3. **Stagnation tolerance**: Increase stagnation_limit to allow more exploration
4. **Environment**: Switch level or reload location on retry

Cost tracking is cumulative across retries: if the total spend exceeds the budget, no further retries are attempted regardless of diagnosis. This creates a natural annealing schedule — early retries are cheap (haiku, 30 iterations), later retries are expensive (opus, 100 iterations), and the system self-terminates before unbounded spend.

---

9. Anti-Cheat Validation (Track 3)

9.1 Motivation

An LLM-driven optimizer will exploit any shortcut the scoring function rewards. Without validation, the system could produce billboard scenes (a flat image plane that looks correct from one angle), skybox-only environments (no actual geometry), or photo-textured planes that score high on CLIP similarity but contain no 3D structure. Track 3 prevents this.

9.2 Multi-View Validation

The multi-view validator rotates the camera to five angles (0, +45, -45, +90, -90 degrees) and scores each view against the reference using CLIP embeddings. Legitimate 3D scenes maintain coherence across viewpoints; billboards and backdrops show dramatic score drops at oblique angles.

- **min_score_threshold**: 0.6 — any single angle scoring below this fails validation
- **max_variance_threshold**: 0.2 — high variance across angles indicates view-dependent trickery

9.3 Depth Consistency

The depth validator estimates depth maps using MiDaS (monocular depth estimation) for both the reference and the rendered scene. It computes: - **Pearson correlation** between depth distributions (threshold: 0.7) - **Depth variance** — values below 0.01 indicate a flat skybox with no geometry - **Edge alignment** — IoU of major depth discontinuities (threshold: 0.3)

9.4 Cheat Type Classification

Detected violations are classified into types: `BILLBOARD` (dramatic score drop at angles), `SKYBOX_ONLY` (no depth variance), `FLAT_PLANE` (low depth variance but some geometry), `PHOTO_TEXTURE` (high 2D similarity but no 3D structure), and `SCALE_TRICK` (correct from one distance, wrong at others).

Validation runs every N iterations (default 5) and applies a score penalty (0.8x multiplier) for invalid scenes, steering the optimizer away from degenerate solutions.

---

10. Scene Memory and Warm Starting

10.1 CLIP-Based Scene Retrieval

When a new reference image is provided, the system encodes it using CLIP (ViT-B/32, 512 dimensions) and compares the embedding against a local memory store of previously optimized scenes. The store records, for each completed job:

- The reference image CLIP embedding
- The scene type and style classification
- The final converged parameters (as MCP commands)
- The per-iteration adjustment history (what worked, what regressed)
- The final composite score

The most similar prior scene (by cosine similarity) provides **warm-start initialization**: its converged commands are replayed at the start of the new optimization, potentially skipping 5–15 early iterations that would otherwise rediscover the same parameter regime.

10.2 Cumulative Command Tracking

The system maintains a `cumulative_commands` dictionary that tracks every MCP command that ever improved the score across all iterations. This differs from merely recording the best-scoring iteration's commands: a sky improvement at iteration 3 might be overwritten by a fog adjustment at iteration 7, but both contributions are valuable. The cumulative set represents the union of all beneficial changes and is stored in scene memory for future warm-starting.

---

11. Infrastructure and Scaling

11.1 Single-Machine Multi-Instance

On a single GPU machine, N UE5 instances share the filesystem and are differentiated by port (30010 + N). Instances are launched in headless mode (`-RenderOffScreen`) with staggered startup (1-second delay between instances) to avoid I/O contention. Health is monitored via MCP `list_actors` probes.

11.2 Cloud GPU Droplets

For cloud deployment, GPU droplets are provisioned from a snapshot that includes the full UE5 project with assets pre-baked. Snapshots are ~61GB and cost ~$0.05/GB/month for storage. Droplets are destroyed after 6 hours of inactivity and rebuilt from snapshot in 3–5 minutes on demand, reducing costs from $1.57/hour (running) to pennies (dormant).

11.3 Asset Sharing Without Duplication

The UE5 project assets are stored in a Git repository with LFS (Large File Storage) backed by object storage (~61GB total, ~$5/month). Multiple concurrent sessions avoid duplication through three mechanisms:

1. **Shared base assets**: The `/Content/Shared/` directory is read-only and shared by all instances on the same machine
2. **Per-session overrides**: Each session writes only to `/Content/Worlds/{session_id}/`, creating Material Instances (small override files, ~10KB each) rather than duplicating base materials
3. **Sparse checkout**: Remote sessions use Git sparse checkout to download only the directories they need, reducing per-session bandwidth from 61GB to 500MB–2GB

11.4 Cost Model

Component	Cost	Notes
LLM calls (per iteration)	$0.002–0.05	Depends on provider and tier
GPU render time	$0.001–0.003	Per frame, headless mode
CLIP scoring	negligible	Runs on CPU
Total per iteration	~$0.01–0.06	
Typical job (30 iterations)	$0.30–1.80	
Evolution cycle (5 benchmarks)	$1.50–9.00	Reflects multiple jobs

12. Discussion

12.1 Relationship to JEPA

Ralpha instantiates every module of LeCun's cognitive architecture [1]:

JEPA Module	Ralpha Implementation
Perception	Screenshot capture → CLIP encoding
World Model	UE5 physics engine (deterministic, zero error)
Actor	50+ domain-specialist LLM agents
Cost Module	Composite perceptual-semantic scorer
Short-term Memory	Evidence system + scene memory
Configurator	Router agent + convergence guard

The key difference is that Ralpha's world model is not learned — it is engineered. This eliminates representation collapse, grounding, and the cost function bootstrap problem, but limits the system to domains where a physics simulator exists. We argue this is not a limitation but a feature: for any domain with a simulator, the hard part of JEPA is already solved.

12.2 Relationship to TextGrad

TextGrad [16] proposes using LLMs to compute "textual gradients" — natural language feedback that functions analogously to numerical gradients in backpropagation. Ralpha's specialist agents produce something similar: given a current render and a reference, they output structured parameter adjustments that move the rendered scene toward the reference.

However, Ralpha differs in three important ways: (1) the "gradient" is computed by domain specialists with injected physics knowledge, not by a generic LLM; (2) the "parameter space" is the explicit, physically-grounded command API of UE5, not a latent representation; (3) the convergence guard provides trajectory-aware control that TextGrad lacks.

12.3 Relationship to Reflexion

Reflexion [14] introduced verbal reinforcement for LLM agents: an agent reflects on task feedback and maintains episodic memory. Ralpha's evidence system extends this with structured quantitative records (exact parameter values, score deltas, before/after comparisons) rather than natural language summaries. The convergence guard adds algorithmic control (budget scaling, oscillation detection, automatic revert) that Reflexion's purely verbal reflection does not provide.

12.4 Limitations

1. **Simulator dependency**: Ralpha requires a physics engine. It cannot optimize scenes in domains without simulators (though the brain architec-

ture is engine-agnostic and could target Blender, Unity, or other simulators via different MCP plugins).

2. **Scoring ceiling**: The composite scorer, while multi-metric, is not perfect. CLIP embeddings miss fine details; LPIPS can disagree with human judgment; region-aware scoring requires accurate region segmentation. The system converges to what the scorer rewards, which may diverge from human aesthetic preference.

3. **LLM dependence**: The quality of optimization is bounded by the LLM's ability to reason about visual parameters. When an LLM misunderstands physics (e.g., proposes fog density changes to fix a lighting problem), the iteration is wasted. Deterministic tools mitigate this but do not eliminate it.

4. **Cost**: At $0.01–0.06 per iteration and 30–50 iterations per job, the system costs $0.30–3.00 per scene. This is cheap for production use but expensive for large-scale research experiments requiring thousands of runs.

12.5 Future Work

Monte Carlo Tree Search (MCTS): The current system includes an MCTS implementation (`brain/planning/mcts.py`) that enables multi-step lookahead with UCB1 selection: "if I adjust the sun angle, what cloud modification should follow?" This represents Stage 4 of the five-stage world model planning evolution (Stage 0: sequential, Stage 1: multi-turn refinement, Stage 2: branch-and-commit, Stage 3: parallel rendering, Stage 4: tree search). All five stages are implemented.

Cross-simulator transfer: The brain architecture is simulator-agnostic. We plan to implement MCP plugins for Blender and Unity, enabling knowledge transfer between engines — a specialist prompt learned on UE5 sunsets should be partially applicable to Blender sunsets.

Learned scoring components: While the current scorer uses frozen pretrained models (CLIP, LPIPS), fine-tuning these on domain-specific human preference data could improve scoring accuracy for specific use cases.

---

13. Related Work

AI-driven 3D scene generation: DreamFusion [19] and subsequent work use score distillation to optimize 3D representations guided by 2D diffusion models. These generate novel content rather than matching references, and produce NeRF/Gaussian representations rather than editable engine scenes. 3D-GPT [20] uses LLMs to generate Blender scripts for procedural 3D scene creation. Unlike Ralpha, it produces one-shot scripts rather than iterating toward a visual target.

LLM-driven optimization: OPRO (Yang et al., 2024) [15] uses LLMs as optimizers, generating solutions and using scoring feedback to improve. Ralpha extends this paradigm with domain decomposition (specialists), trajectory control (convergence guard), and parallel exploration (multi-instance).

World models in reinforcement learning: World models [21] learn dynamics from experience and use them for planning. Dreamer [22] learns a latent dynamics model for model-based RL. Ralpha sidesteps the learning problem by using an existing simulator, but the planning-through-simulation paradigm is shared.

Visual grounding of language models: Recent work on vision-language models (GPT-4V, Gemini, Claude) enables LLMs to reason about images. Ralpha leverages this for the PLAN phase — the LLM "sees" both the reference and current render — but adds structured evidence, deterministic tools, and convergence control that raw VLMs lack.

---

14. Conclusion

Ralpha demonstrates that LLM-orchestrated physics simulation is a viable paradigm for visual scene optimization. By using Unreal Engine 5 as a ground-truth world model, the system eliminates the representation learning problem at the heart of JEPA and transforms scene matching into a structured search problem.

The key engineering contributions — composite scoring with adaptive weights, trajectory-aware convergence control, parallel hypothesis exploration across multiple instances, and lock-free cross-domain knowledge sharing — address the practical challenges of running LLM-driven optimization loops at scale. The meta-cognitive evolution layer, which uses structured multi-perspective deliberation to evolve specialist prompts through A/B-tested revisions, demonstrates that the system can improve not just individual scenes but its own optimization capability over time.

The system currently controls 2,300+ parameters through 218 command handlers with 50+ specialist agents. The architecture is modular: new specialists, scoring components, and deterministic tools can be added without modifying the core loop. We believe this modularity — combined with the deliberate separation of deterministic computation from LLM reasoning — points toward a general framework for AI-orchestrated simulation that extends beyond visual rendering to any domain where a physics simulator exists.

References

[1] Y. LeCun, "A Path Towards Autonomous Machine Intelligence," Version

0.9.2, June 27, 2022. OpenReview.

[2] M. Assran et al., "Self-Supervised Learning from Images with a Joint-Embedding Predictive Architecture," CVPR 2023.

[3] A. Bardes et al., "V-JEPA: Revisiting Feature Prediction for Learning Visual Representations from Video," arXiv:2404.16930, 2024.

[4] J. Tagg, "Solving the JEPA Problem: Ground-Truth World Models via AI-Orchestrated Physics Simulation," Ralpha Technical Report, 2026.

[5] J. Tagg, "Recursive Self-Improvement in AI-Driven Visual Rendering: From Specialist Agents to Meta-Cognition," Ralpha Technical Report, 2026.

[6] A. Radford et al., "Learning Transferable Visual Models From Natural Language Supervision," ICML 2021. (CLIP)

[7] A. Bardes, J. Ponce, Y. LeCun, "VICReg: Variance-Invariance-Covariance Regularization for Self-Supervised Learning," ICLR 2022.

[8] "Yann LeCun founds AMI Labs," TechCrunch, December 2025.

[9] "World Labs raises $1B at $5B valuation," Bloomberg, February 2026.

[10] J. Bruce et al., "Genie: Generative Interactive Environments," ICML 2024.

[11] NVIDIA, "Cosmos: World Foundation Models," Technical Report, 2025.

[12] A. Radford et al., "Learning Transferable Visual Models From Natural Language Supervision," ICML 2021.

[13] Z. Zhang et al., "Darwin Godel Machine: Open-Ended Evolution of Self-Improving Agents," arXiv:2505.22954, 2025.

[14] N. Shinn et al., "Reflexion: Language Agents with Verbal Reinforcement Learning," NeurIPS 2023.

[15] C. Yang et al., "Large Language Models as Optimizers," ICLR 2024. (OPRO)

[16] M. Yuksekgonul et al., "TextGrad: Automatic 'Differentiation' via Text," arXiv:2406.07496, 2024.

[17] R. Zhang et al., "The Unreasonable Effectiveness of Deep Features as a Perceptual Metric," CVPR 2018. (LPIPS)

[18] J.P. Guilford, "The Nature of Human Intelligence," McGraw-Hill, 1967.

[19] B. Poole et al., "DreamFusion: Text-to-3D using 2D Diffusion," ICLR 2023.

[20] C. Sun et al., "3D-GPT: Procedural 3D Modeling with Large Language Models," arXiv:2310.12945, 2023.

[21] D. Ha and J. Schmidhuber, "World Models," NeurIPS 2018.

[22] D. Hafner et al., "Dream to Control: Learning Behaviors by Latent Imagination," ICLR 2020.

[23] A. Novikov et al., "AlphaEvolve: A Gemini-Powered Coding Agent for Designing Advanced Algorithms," arXiv:2506.13131, 2025.

[24] M. Assran et al., "V-JEPA 2: Self-Supervised Video Models Enable Understanding, Prediction and Planning," arXiv:2506.09985, 2025.

[25] K. Fernando et al., "Promptbreeder: Self-Referential Self-Improvement Via Prompt Evolution," ICML 2024.

[26] E. Zelikman et al., "Self-Taught Optimizer (STOP): Recursively Self-Improving Code Generation," COLM 2024.

[27] M. Assran et al., "V-JEPA 2," arXiv:2506.09985, June 2025.

[28] X. Chen et al., "VL-JEPA: Vision-Language Joint Embedding Predictive Architecture," arXiv, December 2025.

[29] R.J. Heuer and R.H. Pherson, "Structured Analytic Techniques for Intelligence Analysis," 2nd ed., CQ Press/SAGE, 2015.

[30] H. Mercier and D. Sperber, "Why Do Humans Reason? Arguments for an Argumentative Theory," Behavioral and Brain Sciences, 34(2), 57-74, 2011.

Appendix A: State Machine Phase Details

Phase	Input	Output	Duration
INIT	Reference image, domain, config	Loaded level, camera position, warm-start params	5–30s
DECOMPOSE	Reference image	Structured element list, specialist phase order	3–10s (LLM)
PLAN	Current render, reference, evidence, score	Proposed MCP commands	3–10s (LLM)
ACT	MCP commands	Applied scene changes	0.1–2s
OBSERVE	—	Screenshot capture	0.5–2s
SCORE	Current render, reference	Composite score, per-dimension breakdown	1–3s

Phase	Input	Output	Duration
REVISE	Score, trajectory, guard state	Keep / revert / isolate / escalate	<0.1s

Appendix B: Specialist Domains

Domain Category	Example Specialists	Typical Parameters
Atmosphere	sky, sunset, clouds, fog, weather	SkyAtmosphere, VolumetricCloud, ExponentialHeightFog
Lighting	directional, local, interior, DMX	DirectionalLight, SkyLight, PointLight, SpotLight
Camera	cinematography, lens, exposure, geo	CameraComponent, PostProcess (DOF, motion blur)
Water	ocean, water body, tides	WaterBody, wave params, caustics
Characters	MetaHuman, clothing, face quality, performance	MetaHuman appearance, skeletal mesh, animation
Scene	props, terrain, foliage, Cesium, construction	Static/skeletal meshes, landscape, 3D tiles
Post-process	color grade, bloom, compositing, NPR	PostProcessVolume settings
Meta	debug, solve, reason, deep-dive	Cross-domain analysis and troubleshooting

Appendix C: Scoring Weight Configurations

Available Components	w_traditional	w_semantic	w_perceptual	w_region
All four	0.25	0.30	0.25	0.20
No regions	0.30	0.35	0.35	0.00
No LPIPS	0.35	0.35	0.00	0.30
Semantic only	0.60	0.40	0.00	0.00
Traditional only	1.00	0.00	0.00	0.00