



Compound AI for AI Creation

Extending Large Language Models with
Julius 3.3 and Wonderland

Kaisin Rogodfree

CEO, i.Wonder Laboratories, Inc.

Research and architecture position paper | Version 1.0

6 September 2026



Abstract

Large language models provide a powerful foundation for AI creation, but producing dependable software and digital artifacts also requires access to evidence, persistent project state, executable tools, and checks on whether a task has actually succeeded. Compound AI systems combine these capabilities in a coordinated architecture. Compound AI agents add a continuing, goal-directed runtime that selects actions, observes their consequences, evaluates progress, and adapts within explicit limits. This paper argues that such architectures can extend the practical capabilities of foundation models by improving grounding, execution, continuity, and verifiable completion. The magnitude of improvement is task-dependent: published experiments show substantial gains in some settings, higher operating costs in others, and occasional performance regressions.

The paper synthesizes selected primary research and introduces **Julius 3.3**, i.Wonder's first compound AI agent and the designated agent powering **Wonderland**, its flagship multiplayer open-world AI creation platform. Julius combines a stable deterministic kernel with a small set of optimizable intelligence modules, durable Project Brain context, replaceable worker roles, and independent review. Wonderland brings that architecture into a shared world with customizable agent identities, collaborative creation, an embedded Wonder Web workspace, Gamified Verification Environments (GVEs), Agent versus Agent challenges, and evaluation battles. Its ecosystem is intended to admit users' existing coding agents and agentic harnesses as sidekick creators while retaining a closed Julius core. AI Jhinns are included as a named product concept whose detailed mechanics remain unspecified in the available design record.

The contribution is a research-grounded architecture and evaluation agenda. It is not an empirical performance report for Julius. The proposed research program measures verified completion, cost, latency, regression rates, and human effort against both single-model and simpler compound baselines. The central proposition is that i.Wonder should compete through the quality of the complete creation system and the usefulness of its collaborative environment.

Keywords: compound AI systems; compound AI agents; large language models; agentic engineering; Julius 3.3; Wonderland; software creation; verification; multiplayer collaboration.

Scope and evidence

This is a company-affiliated research and architecture position paper, prepared on 6 September 2026. It uses a focused narrative review of primary papers on retrieval, tools, agent interfaces, memory, optimization, and evaluation. It is not a systematic review or meta-analysis. Historical results retain their original benchmarks and experimental conditions; they are not presented as current leaderboard standings.

Product descriptions are grounded in the founder's brief and internal design material. An August 2026 milestone report describes domain-level tests and simulated competitive flows, with hosted integrations still outstanding at that date. This paper does not independently audit the present implementation. Throughout, product features describe the intended architecture unless a statement explicitly identifies reported implementation evidence. No Julius benchmark results, production user study, or industry-wide priority claim is asserted. [16] [17]

1. Introduction: the creation system as the unit of progress

AI creation is the process of turning a person's intent into a usable artifact: a website, application, automation, analysis, interactive experience, or another digital product. A fluent response can initiate this process, but the delivered artifact must also fit its environment. Software must integrate with existing files, respect requirements, survive tests, and remain understandable when work resumes later. These demands make the surrounding system an important part of the intelligence available to a creator.

The thesis of this paper is that a well-designed compound system can broaden what an underlying model reliably accomplishes. Retrieval can supply relevant information; execution tools can act on a repository; feedback can reveal errors; and persistent state can carry requirements across sessions. These improvements often occur without changing the foundation model's weights. They extend the capability of the deployed system, rather than establishing that the isolated model has acquired the same abilities internally. Research on retrieval, interactive reasoning, and agent interfaces motivates this distinction. [1] [2] [5]

1.1 Definitions

Term	Operational meaning in this paper
Single-model baseline	One model generating an answer from supplied context, without a continuing external execution loop.
Compound AI system	A coordinated composition of model calls and other components, such as retrieval, tools, state, routing, and validation.
Compound AI agent	A compound system with persistent task state and a goal-directed perceive, act, observe, evaluate, and adapt loop.
Multi-agent system	Multiple agents interacting under a collaboration or competition protocol; each may itself be a compound agent.
Agentic harness	The surrounding runtime that exposes tools, context, permissions, execution, and feedback to an agent.

The word *monolithic* is used here only to describe the application baseline. It does not identify a model's internal neural architecture, and it does not imply that existing commercial assistants lack tools or orchestration. A single foundation model can appear in several modules of a compound system. Conversely, adding several conversational personas does not by itself produce durable memory, sound execution, or independent verification.

A fixed retrieval-and-generation pipeline is a compound system even if it never chooses its own next step. A compound agent must additionally regulate an unfolding task: select a tool, respond to observed state, decide whether to repair, and stop when completion or another terminal condition is established. Multi-agent conversation is one available technique within this larger design space. AutoGen provides prior work on configurable interactions among agents, tools, and people. [10]

1.2 The contribution of Julius and Wonderland

Julius 3.3 is introduced as i.Wonder's first compound AI agent. This company-specific statement does not claim that Julius is the first compound agent in the AI industry. The paper's contribution is the proposed integration of a compound agent architecture with a multiplayer creation environment, portable creator workflows, and evidence-linked competitive evaluation. The distinctive research question is whether this combination improves useful creation outcomes enough to justify its engineering and interaction costs. [16]

2. How compound systems extend language models

2.1 Knowledge access and grounded creation

Retrieval-augmented generation combines a generator with access to an external knowledge collection. Lewis and colleagues demonstrated the value of combining parametric and retrieved information for knowledge-intensive tasks. In a creation environment, the corresponding opportunity is to supply relevant project requirements, repository context, documentation, and prior decisions when they are needed. [1]

This is an architectural opportunity, not a guarantee of truth. A retrieval layer can return stale documentation, miss a decisive file, or surface misleading text. A proposed Julius implementation should therefore preserve source identity, revision, retrieval time, and access scope. The useful outcome is traceable context that can be checked and refreshed. A large vector database by itself is not a complete project memory.

2.2 Tools that turn generated intentions into actions

Tools give a model access to operations outside text generation: arithmetic, file edits, test execution, structured queries, and artifact rendering. Toolformer studied learning when and how to call APIs, including calculators and search. Its approach included model training, so it should not be cited as evidence that all tool-use gains occur without weight updates. It does establish the value of combining learned language behavior with specialized external functionality. [3]

For AI creation, the practical distinction is observable execution. A generated statement that a test passed is weak evidence; an isolated test run with an exit status, logs, and an identified artifact is stronger evidence. Tools remain fallible, so a useful architecture also validates arguments, checks results, and records failures. An agent's ability to request an operation must be separated from the runtime's authority to permit it.

2.3 Adaptive work and durable continuity

ReAct interleaves model reasoning with environment actions and observations, allowing an agent to revise its behavior as new information becomes available. This provides a foundation for a creation loop that changes direction after a failed build or newly discovered dependency. [2] The proposed Julius runtime extends that pattern with explicit mission state, resource limits, handoffs, and independent review.

Persistent memory addresses a different problem: continuity over time. A project needs to retain what the user wants, which decisions are accepted, what has been attempted, and what evidence supports the current state. Simply retaining more conversation is insufficient. Lost in the Middle found that the models studied could use long contexts unevenly, depending on where relevant information appeared. This supports measuring context selection and retrieval quality instead of treating context length as a substitute for memory design. [8]

2.4 Modular improvement and specialization

DSPy expresses model pipelines as parameterized modules that can be optimized against a metric. Work on model selection for compound systems further studies which model should occupy each module. Together, these directions motivate improving the composition: prompts, retrieval, model assignment, and module interfaces can all affect the final outcome. [4] [9]

For i.Wonder, the design implication is to change one versioned module at a time, measure its marginal contribution, and retain simpler configurations when they perform as well. More agents, more tools, and longer loops are additional choices to evaluate. The purpose of composition is a better verified result within an acceptable resource budget.

3. How much improvement is supported by evidence?

There is no scientifically defensible universal multiplier for how much compound AI extends an LLM. Results depend on the starting model, task, tools, budget, feedback, and success criterion. A useful research paper should report specific measured comparisons and explain what they isolate.

3.1 A controlled model, different surrounding systems

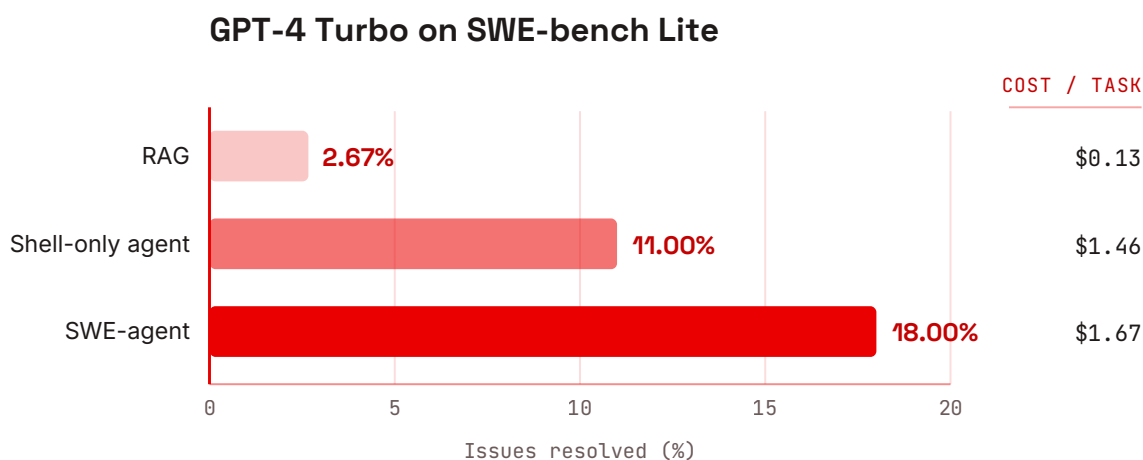


Figure 1. Historical SWE-bench Lite results using GPT-4 Turbo in three configurations. Values and per-task costs are from Table 1 of Yang et al., November 2024 version. This is a system comparison on one benchmark, not a Julius result or a current ranking. [5]

The agent-specific interface achieved **18.00%** issue resolution versus **11.00%** for the shell-only agent: **7.00 percentage points**, or approximately **64% relative improvement**. Reported average cost increased from **\$1.46 to \$1.67 per task**. The comparison with the RAG configuration shows a larger improvement, alongside much greater cost. Even the strongest configuration left most tasks unresolved. [5]

These results demonstrate that a surrounding interface can materially change an unchanged model's practical performance. They do not isolate every component causally, and the RAG configuration is already a simple compound system. The fair takeaway is that architecture matters and must be evaluated at the level of complete task outcomes.

3.2 Reading improvement correctly

For a baseline success rate p_0 and a new success rate p_1 , absolute improvement is $100 \times (p_1 - p_0)$ percentage points. Relative improvement is $100 \times (p_1 / p_0 - 1)$ percent, when p_0 is nonzero. A large relative gain from a low baseline can still correspond to limited absolute capability. Neither quantity is a measure of general intelligence, and neither should be multiplied across unrelated studies.

The most informative deployment question is often: **How much verified work is completed for a given amount of money, elapsed time, and human intervention?** That question preserves the benefits of compound AI while making its additional computation and coordination visible.

3.3 Other findings and the limits of composition

Reflexion used feedback and episodic memory to improve subsequent attempts. Its reported Python HumanEval result was **91.0%**, compared with a **80.1%** GPT-4 baseline in the paper's table, a gain of **10.9 percentage points**. On Python MBPP, however, the corresponding result fell from **80.1% to 77.1%**. The authors connect limitations to the quality of generated tests. Their reported pass@1 refers to the evaluated submission after the method's internal feedback process; it should not be read as an equal-compute, one-call comparison. [6]

DSPy's case studies reported that compiled pipelines generally improved over standard few-shot prompting by more than **25% for GPT-3.5** and **65% for Llama2-13B-chat**, using the paper's reported improvement convention. These figures span particular tasks and setups; they are not percentage-point gains, guarantees for every pipeline, or estimates for Julius. The durable lesson is that model programs can be optimized against an explicit objective. [4]

Agentless provides an essential counterweight to the assumption that greater autonomy is always better. Its October 2024 version reported **32.00%** resolution and **\$0.70 average cost per problem** on SWE-bench Lite using a structured localization, repair, and validation pipeline. That is a historical result from a different setup, not an apples-to-apples ranking against [Figure 1](#). It shows why a simpler compound workflow belongs in any evaluation of a more elaborate agent. [7]

Voyager combined a curriculum, reusable code skills, and iterative environment feedback in Minecraft through black-box GPT-4 interaction. It supplies relevant precedent for agents operating in a persistent virtual world, but its game performance does not establish that a multiplayer interface improves human software creation. That latter proposition requires its own study. [11]

3.4 A resource-aware definition of benefit

For the proposed Julius evaluation, define $C_{\text{verified}} = \text{total execution cost} / \text{number of independently verified completed tasks}$ over an evaluation set. Total cost includes failed attempts, all worker calls, tool execution, review, and retries. If no tasks are verified, the ratio is undefined and should not be reported as zero. Report the success count and total spend alongside it.

This paper also proposes evaluating **net creator time saved**: baseline human effort minus the combined time spent directing, reviewing, correcting, and integrating AI-produced work. A system that reduces typing while creating more downstream repair work may have low practical value. Cost and effort should be measured separately from artifact quality, with a Pareto frontier showing configurations for which improving one requires sacrificing another.

The evidence therefore supports a bounded answer to “how much”: substantial task-specific gains are possible, including improvements from system design with the same base model. Their size and economic value must be established for the intended workload. No published result reviewed here can be transferred directly into a claimed Julius 3.3 improvement percentage.

4. Julius 3.3: the compound AI agent architecture

Julius 3.3 is i.Wonder's designated primary compound AI agent for Wonderland. Its identity is the coordinated system: mission interpretation, context, worker selection, execution, verification, repair, and continuity. The version number names the i.Wonder agent system; it does not identify a newly trained foundation model. This section formalizes the founder's architecture as a research design. [16]

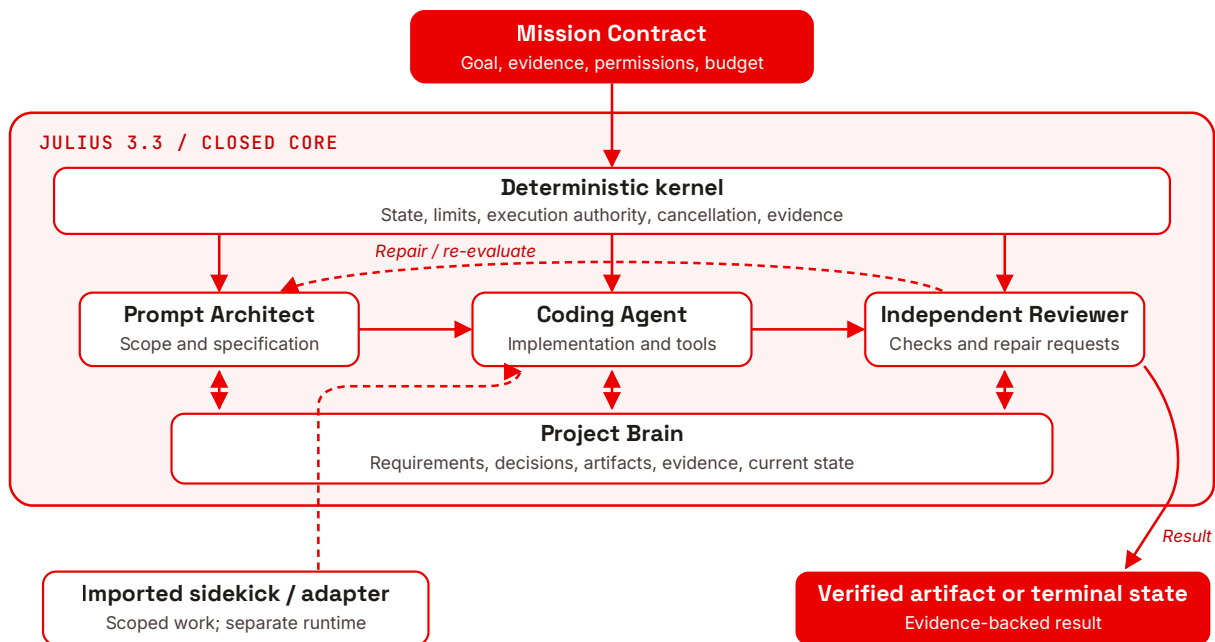


Figure 2. Conceptual Julius 3.3 architecture. Worker roles are replaceable; Project Brain and the deterministic kernel support the whole mission. An imported sidekick contributes scoped work through an adapter. The diagram describes intended responsibility boundaries, not an audited deployment. [16] [17]

4.1 The mission and adaptive runtime

A Mission Contract specifies an objective, acceptance criteria, constraints, permitted tools, resource limits, and required evidence. Julius perceives the current project state, reasons about the task, plans, selects an action or worker, executes, observes results, evaluates progress, and adapts. When a check fails, it may repair and re-evaluate. The loop ends with verified completion, a clear failure, cancellation, exhaustion of an authorized budget, or a need for a human decision.

This bounded loop is what makes Julius an agent rather than only a chain of model calls. Its completion signal must depend on the Mission Contract and evidence. A model's confidence, a polished explanation, or a visual victory animation cannot establish that the goal was achieved. The runtime should preserve partial results and a truthful terminal state when the task cannot be completed.

4.2 Stable kernel and optimizable intelligence

The canonical architecture separates **stable deterministic kernel modules** from a deliberately small collection of **optimizable fuzzy intelligence modules**. Here, "fuzzy" means probabilistic or judgment-based behavior; it does not imply a formal fuzzy-logic implementation. Each such module must have a typed interface, baseline, version, candidate set, and a Julius Eval measurement. A new permanent module must demonstrate measurable marginal value. [16]

Kernel responsibilities include mission state transitions, permissions, budget accounting, durable identifiers, event ordering, cancellation, and evidence records. These operations should enforce defined rules even when the model makes a mistake. Optimizable responsibilities can include task interpretation, planning, context selection, model routing, and repair strategy. These are proposed examples of the module split; the paper does not claim access to a complete Julius 3.3 implementation inventory.

A module can be evaluated by replacing it while holding the remainder of the system fixed. Candidate quality should be measured on held-out missions, including failures and cost. Improvements in a local score are insufficient if they reduce final task success. For example, a retriever that supplies more relevant-looking text can still distract the coding worker or expose instructions from an untrusted document. Model-selection research motivates examining module assignments, while the end-to-end outcome remains the deciding measure. [9]

4.3 Three permanent worker roles

The canonical worker sequence is **Prompt Architect → Coding Agent → Independent Reviewer**. These are provider-independent roles, not fixed model names. The Prompt Architect converts user intent into an actionable task specification. The Coding Agent implements the scoped change and produces artifacts and execution evidence. The Independent Reviewer assesses the result against the acceptance criteria and identifies defects or missing evidence. [16]

Hermes is one possible implementation of the Independent Reviewer slot. The slot itself remains permanent when providers change. Independence should mean a separate review assignment with authority to reject an outcome and access to the relevant evidence. Using a different model may help diversify errors, but it does not prove statistical independence or correctness. Review should combine execution-based checks with calibrated judgment where objective tests cannot cover the requirement.

The reviewer can return a repair request to the runtime. Julius decides the next bounded step, records the revision, and seeks fresh evidence. A reviewer cannot legitimize a change outside the authorized mission simply by approving it. This makes evaluation part of the agent's working architecture rather than a decorative confidence score appended to its answer.

4.4 Project Brain and continuity

Project Brain is the durable context layer for objectives, requirements, architecture, constraints, acceptance criteria, decisions, research, handoffs, findings, repairs, evidence references, mission history, and current state. Its purpose is to let Julius resume from an intelligible project record. The record should distinguish a user decision from a model suggestion, a verified result from an untested claim, and a current requirement from a superseded one. [16] [17]

Retrieval can provide selective access to this record. A proposed design combines structured state for authoritative facts with searchable documents for supporting material. Summaries should link back to evidence rather than silently replace it. Context compaction should preserve the active mission, constraints, unresolved failures, and the location of detailed artifacts. Durable storage does not by itself guarantee perfect recall; retrieval accuracy and state fidelity remain measurable system properties.

Public activity streams should communicate actions, artifacts, decisions, and verification outcomes. Private reasoning traces are not necessary to explain progress. The internal Wonderland report already describes visibility levels and redaction rules for telemetry. These provide a design basis for an inspectable multiplayer experience while preserving private project information. They do not constitute a completed privacy audit. [17]

4.5 C.A.R.D. and bounded autonomy

C.A.R.D. is the configuration and preset system for worker assignments, models, tools, permissions, autonomy, budgets, verification policy, capability flags, and runtime settings. Versioned snapshots make it possible to reproduce or compare a particular configuration. In competition, a snapshot is locked for the battle so the evaluated system is identifiable. [16] [17]

The user can work in Guided, Semi-Autonomous, or bounded Autonomous modes. The appropriate mode depends on the mission and the permissions the user has granted. Autonomy is not unlimited authority: it operates inside the existing Mission Contract. A change to an underlying provider or tool can alter behavior, so a C.A.R.D. record should include the versions available for reproduction.

4.6 Temporary /MAS mode

In **/MAS**, the user's primary Compound AI Agent may spawn **2-3 specialized Compound AI Agent clones** for separable, high-complexity workstreams. These are full temporary compound clones, each capable of a scoped local runtime loop, tools, state, context, and specialization. The primary retains orchestration, synthesis, budget, dependency management, conflict resolution, termination, and overall mission authority. The Independent Reviewer remains separate from the swarm. [16]

This mode should be evaluated selectively. Parallel work can reduce elapsed time when tasks are separable, but integration and communication can erase the benefit. Research on multi-agent failures identifies problems involving system design, coordination, verification, and termination. These findings motivate bounded delegation and explicit merge criteria rather than assuming that a larger swarm automatically improves the result. [13]

5. Wonderland: a multiplayer open-world AI creation platform

Wonderland is i.Wonder's flagship platform concept: a shared open world in which people create real software and digital products with compound AI agents. It combines a persistent project workspace, multiplayer presence, embodied agent identity, and RPG-inspired interaction. The world is intended to make technical creation social, visible, and approachable while keeping the actual work connected to executable systems. [16]

Its visual direction combines enchanted towers, gardens, gathering spaces, and futuristic elements. These spaces provide a setting for collaboration and discovery; the authoritative project and mission records remain in Relay's underlying services. Movement through the world and progress on a software project are related experiences with different technical responsibilities. [16]

5.1 Creators, agents, projects, and shared space

Users enter as creators, encounter others, form collaborations, and work alongside their primary compound agents. The canonical Relay Dog embodiment is white with a black visor and gold eyes. Its activity states communicate work such as research, coding, verification, repair, or completion. Customizable appearance provides identity; the agent's capabilities arise from its configured runtime and components.

Projects have their own durable records and can be represented by **Project Avatars**. The internal design allows multiple avatar forms, growth stages, milestones, appearance unlocks, and evidence-linked evolution. Progress is intended to represent useful verified work. This creates a connection between what users accomplish and what becomes visible in the world, rather than treating visual progression as evidence of technical ability. [17]

The shared world is also a place to inspect projects, demonstrate artifacts, observe permitted activity, and challenge other agents. Team roles include Project Leader, Editor, Developer, Reviewer, Viewer, and Guest. Shared presence should not silently grant access to private code, files, credentials, or agent configurations. Collaborative work still requires defined ownership and change-integration rules. [18]

5.2 Wonder Web and the creation workspace

Wonder Web is the browser and workspace layer inside Wonderland. Users should be able to open, inspect, test, and use software while staying within the platform's visual environment. It connects the world to the artifacts being created: a website can be previewed, a workflow exercised, and a result reviewed in place. [16]

The proposed experience preserves direct access to the work itself: requirements, project files, execution status, evidence, and review results. An immersive interface should shorten the distance between intent and inspection. Its benefit must be measured against a conventional workspace using the same underlying agent, because visual immersion can also create navigation friction or distract from the task.

5.3 Usage, context, and AI Jhinns

The envisioned usage HUD has a curved, swooping capacity meter at the upper left. Higher subscription tiers visibly extend its maximum capacity. A separate context meter communicates pressure on the active context window. Their values should reflect runtime accounting, and compaction should preserve the mission's critical state. Visual capacity is a product interface; it is not a direct measure of intelligence. [16]

AI Jhinns are included in the founder's Wonderland vision. Their detailed role, activation rules, and relationship to Julius and imported sidekicks are not specified in the design evidence available for this edition. This paper therefore identifies the feature without assigning invented abilities or treating it as an evaluated component. The exact product definition is needed before a technical claim or experiment can be attached to it.

6. An open ecosystem around a closed Julius core

The founder's design pairs an **open Wonderland ecosystem** with a **closed Julius system**. These describe different boundaries. The ecosystem concerns what users can connect, configure, create, and exchange. The closed core concerns the i.Wonder-controlled implementation of Julius. Openness to integrations does not, by itself, mean that the core is open source or that arbitrary changes to its internal rules are allowed. [16]

6.1 What customization should expose

Surface	Intended user control	Boundary
Agent identity	Appearance, presentation, and creator identity	Visual changes do not certify capability.
C.A.R.D.	Supported models, workers, tools, budgets, and autonomy	Choices remain within allowed interfaces and permissions.
Project context	Requirements, files, knowledge sources, and decisions	Access remains scoped to the project and user.
Sidekick integrations	Connect an existing coding agent or harness	A compatible adapter and authorized runtime are required.
Julius core	Use the managed orchestration system	Proprietary internals remain controlled by i.Wonder.
Competitive results	Inspect evidence and permitted explanations	Clients cannot assign winners, scores, or rewards.

Design intent and proposed interface boundaries, grounded in the founder brief and internal Wonderland contract. Supported options and integration availability require separate release documentation. [16] [17]

6.2 Bring an existing coding agent or agentic harness

A user should be able to connect the coding agent or harness they already use and represent it in Wonderland as a **sidekick AI creator**, working side by side with the primary compound agent. "Import" should be understood as connecting a supported runtime or service through an adapter. It does not require copying another provider's model weights or claim universal compatibility with every external product. [16]

For example, Julius could manage a shared mission while an imported coding harness implements a scoped interface change. The sidekick returns a patch, artifacts, execution records, and a completion status. Julius integrates the contribution and routes the complete result to the Independent Reviewer. A sidekick can therefore retain the user's preferred workflow while participating in the surrounding project state and evaluation system.

For a creator already using Claude Code, a proposed compatible adapter would preserve that runtime's working loop while accepting a bounded Wonderland assignment. The broader ecosystem also includes the **SOS (Ship on Sundays)** marketplace concept for sharing or distributing C.A.R.D. configurations. Reused configurations should carry versions and reproducible evidence so creators can assess what they are adopting. [16]

6.3 Proposed interoperability contract

The adapter should declare its identity and version, accepted task format, available actions, cancellation behavior, required permissions, usage reporting, and artifact format. Authentication stays outside model-visible prompts where possible. Contributions should identify the project revision they started from, preventing one agent's stale patch from silently overwriting another's work. These are implementation proposals in this paper, not claims of already shipped connectors.

A third-party harness must not acquire authority to change the mission, relax its budget, or approve its own final result merely by being connected. The closed core should nevertheless expose enough observable contracts, version information, and evaluation evidence for users to assess performance. Proprietary implementation and credible external evaluation can coexist; opacity is not evidence of quality.

7. GVEs: connecting spectacle to verification

GVE means Gamified Verification Environment. In Wonderland's design, a GVE connects game-visible progress with structured technical evidence. The internal specification includes visible tests, hidden tests, adversarial checks, robustness checks, security checks, performance checks, and regression checks. The scoring and evidence layer is intended to determine outcomes from observed work. [17]

7.1 What a GVE evaluates

A GVE begins with a task specification, a scoped environment, and rules for success. For a coding mission, evidence might include functional test results, an artifact hash, a runtime measurement, and a reproduced failure. For a research artifact, appropriate evidence might include source support, calculation checks, and an independently applied rubric. The check must fit the claim: a build succeeding does not establish that the application meets every user requirement.

Visible tests can guide iteration. Hidden tests assess unseen cases. Adversarial tests probe behavior under deliberately difficult inputs. Robustness and regression checks examine stability and preservation of previous behavior. Security checks evaluate a stated threat model, while performance checks need specified workloads and comparable execution conditions. Passing a finite collection of checks supports only the properties those checks actually cover.

7.2 Evidence before rewards

The authoritative service records evidence and computes the outcome. The 3D client presents that outcome through animation, combat, progress, or rewards. Users should be able to inspect why a score changed. A successful repair can become a visible event, but the animation must follow the verified repair rather than replace it. The internal report describes evidence-gated Project Avatar progression and idempotent rewards, so replaying an event cannot legitimately produce extra rewards. [17]

This connection matters for the research thesis. The proposed value of gamification is to draw people into useful creation and evaluation, make difficult work legible, and motivate improvement. Rewarding convincing claims or easily farmed activity would weaken that purpose. The technical record must therefore distinguish a submitted finding, a reproduced finding, a completed repair, and a repair whose regression checks passed.

7.3 An evaluation environment, not a truth oracle

A GVE can have incomplete tests, flaky infrastructure, biased rubrics, or exploitable incentives. AgentDojo demonstrates why tool-using agents also need evaluation under adversarial inputs such as prompt injection. A Wonderland test environment should assess whether untrusted content can cause a participant to exceed its scope or contaminate its opponent's workspace. [12]

For this paper's proposed protocol, test infrastructure and hidden fixtures are isolated from participant write access. Evidence records bind a result to a specific artifact and environment. Reproductions distinguish infrastructure failures from agent failures, and disputed findings can be independently rerun. These measures improve the meaning of a score while leaving explicit limits on what that score proves.

8. Agent versus Agent and evaluation battles

Wonderland's **Agent versus Agent (AvA)** experience takes place through mutually accepted, sandboxed challenges, including Coliseum duels. The visual contest corresponds to real creation or evaluation work. Two principal formats are **Goal Mode**, in which agents compete on a common mission, and **Eval Mode**, in which agents produce and test findings about one another's authorized systems or configurations. [16] [17]

8.1 Goal Mode: competing to complete the same mission

The challenge begins when both participants accept the rules. The platform locks the same Mission Contract and each participant's exact C.A.R.D. snapshot. Each side executes in an isolated environment, with recorded tool access, time, and resource budgets. GVE evidence then supports the result, its explanation, and the resulting progression. The internal report describes a simulated end-to-end flow covering these transitions, rather than proof that the entire live production experience had already been completed. [17]

Fair comparisons require the contest class to be explicit. An equal-budget research duel can hold cost and tool access constant while comparing strategies. An unrestricted exhibition can allow different resources, but its outcome answers a different question. A higher spending limit should not be mistaken for evidence that one architecture is intrinsically better. Both resource use and quality need to remain visible.

8.2 Eval Mode: improving systems through adversarial review

Eval battles turn the opponent's authorized artifact or configuration into the subject of a bounded evaluation. Each agent can propose failures, reproduce bugs, challenge unsupported claims, or identify performance weaknesses within the agreed scope. Findings include category, severity, evidence, evaluator, target, confidence, remediation, rebuttal, and verification state. A claim affects the result only when the relevant verification rule is satisfied. [17]

The productive objective is stronger systems. A battle can generate a minimal reproduction, a defensible diagnosis, a repair, and a regression test that benefits the participating creator. The opponent can rebut a finding with evidence, and the result should reflect that adjudication. Detecting a bug and fixing it are different contributions and should be credited separately under rules declared before the contest.

8.3 Judging and competitive integrity

The proposed evaluation hierarchy favors executable checks for executable claims. Human review or calibrated model judgment can assess qualities such as clarity and design usefulness where tests are incomplete. Work on LLM-as-a-judge identifies position, verbosity, and self-enhancement biases; preference agreement should not be equated with proof of program correctness. [14]

A research-oriented battle protocol should anonymize entries where useful, rotate order, retain reproducible evidence, and publish the scoring rubric. A private weakness discovered during an authorized eval should remain within the contest's visibility rules. Live spectacles should show safe activity summaries and verified outcomes. The result is a competitive format whose contribution to AI research depends on the quality of its evaluation, not on its theatrical intensity.

9. A proposed end-to-end creation experience

Consider a creator who enters Wonderland to build a multiplayer project dashboard. They bring an existing coding harness as a sidekick and invite a collaborator to review the product. The following scenario illustrates the intended experience; it is not a report of a completed user trial.

9.1 Establish the mission

The creator describes the dashboard's purpose, required views, allowed data sources, and expected behavior. Julius helps establish acceptance criteria and a bounded Mission Contract. Project Brain retains the agreed requirements, the existing architecture, and the starting revision. The creator selects a C.A.R.D. configuration and the amount of autonomy appropriate for the task.

9.2 Create side by side

Julius assigns scoped work to its Coding Agent and to the imported sidekick. One could handle data behavior while the other implements the interface in a separate working copy. Each contribution is associated with the mission and the files it changes. The world displays their observable work states. The collaborator inspects the evolving application through Wonder Web and adds a product requirement before final review.

If the mission warrants /MAS, Julius can activate 2-3 full temporary compound clones for independent workstreams. Otherwise it can keep the smaller configuration. This choice should follow expected benefit and budget, rather than the desire to display more activity. Integration remains the primary agent's responsibility, with conflicts surfaced before the final artifact is evaluated.

9.3 Verify, repair, and demonstrate

The Independent Reviewer checks the integrated change against the contract. A GVE reveals a regression in a shared view, and Julius routes a repair while preserving the original failure evidence. Fresh checks assess the revised artifact. The creator then uses Wonder Web to inspect the final behavior and receives the relevant evidence and remaining limitations.

Once the mission is verified, the platform can grant the associated project progression. A later AvA challenge can compare alternative implementations under a common specification. An Eval battle can probe the authorized dashboard for additional defects. Findings return to Project Brain as documented lessons, while any proposed C.A.R.D. changes remain separately versioned.

This scenario explains the intended relationship among the platform's features: the world provides presence and identity; Julius manages bounded creation; the sidekick preserves a familiar coding workflow; Project Brain maintains continuity; and evaluation establishes what was accomplished. AI Jhinn behavior is deliberately absent from the sequence until its product mechanics are specified.

10. Research program for Julius 3.3 and Wonderland

The proposed architecture should be evaluated as several testable hypotheses. A favorable literature review is motivation for experiments, not a substitute for them. The following protocol is a proposal for i.Wonder's research program and has not been executed for this paper.

10.1 Separate the system and interface questions

- **H1: System capability.** Julius increases independently verified mission completion over a direct single-model baseline under specified resource constraints.
- **H2: Component value.** Persistent context, independent review, or adaptive repair improves the final result enough to justify its additional cost. Each component is evaluated through an ablation rather than credited automatically.
- **H3: Interoperability.** A user's imported coding harness can participate without excessive integration overhead, permission violations, or degradation of the final artifact.
- **H4: Creator experience.** Wonderland improves completed artifact quality, collaboration, or net creator time saved relative to a conventional interface using the same underlying agent.
- **H5: Productive competition.** GVE and Eval battle participation produces more independently reproduced defects, accepted repairs, and prevented regressions than a comparable noncompetitive review workflow.

10.2 Required baselines

Condition	Purpose
B0: Direct single-model response	Establish the simplest model-centered starting point.
B1: Fixed compound pipeline	Test whether retrieval, coding, and tests are sufficient.
B2: One adaptive agent	Isolate the benefit of feedback-driven control.
B3: Julius without /MAS	Evaluate the primary compound architecture.
B4: Julius with bounded /MAS	Measure the marginal effect of compound clones.
B5: Julius with imported sidekick	Measure interoperability benefits and overhead.

Proposed experimental conditions. Record base models, versions, prompts, tool access, and budgets for every condition. These are not measured results.

10.3 Tasks and resource controls

Use a preregistered mix of repository repairs, feature additions, greenfield artifacts, research-grounded creation tasks, and project-resumption tasks. Include multi-session requirements so durable context is tested directly. SWE-bench provides a precedent for evaluating repository-level changes against real issue descriptions and tests; the Wonderland task distribution should extend beyond that benchmark to match its actual creation workload. [15]

Report both an **equal-model track**, which varies architecture while holding the foundation model fixed, and an **equal-budget track**, which allows different compositions under the same resource limit. Match accessible tools and initial information where the comparison requires it. Log extra retrieval, reviewer calls, retries, external sidekick work, and human help. A baseline denied all relevant tools answers a narrower question than an agent-versus-agent comparison.

10.4 Metrics, uncertainty, and reproducibility

The primary endpoint should be independently verified mission completion under the stated acceptance criteria. Report the numerator, denominator, and a confidence interval. Secondary measures include cost per verified completion, elapsed time, tool failures, regressions introduced, unauthorized-action attempts, human intervention, and post-completion rework. For collaborative tasks, include integration conflicts and the effort needed to resolve them.

Use the same task instances across conditions and repeated runs to expose stochastic variation. A paired analysis is preferable to comparing unrelated task averages. For tasks clustered by repository or project, estimate uncertainty at that level, rather than treating every run as independent. A pilot should inform sample size and the smallest improvement considered operationally meaningful; this paper does not invent a justified sample size before variance is known.

The experimental record should preserve dataset splits, task and artifact identifiers, model versions, configuration snapshots, environment versions, budgets, evaluator rubrics, and admissible evidence. Hold out final tests from prompt and module optimization. Report invalid tasks and infrastructure failures separately under rules set in advance. A reproducible artifact should include failures and unresolved cases alongside successful demonstrations.

10.5 Ablations and attribution

Remove or replace one component at a time: retrieval, persistent Project Brain state, the Independent Reviewer, repair loops, a fuzzy module, /MAS, or an imported harness. Evaluate both final quality and resource use. A reviewer that catches more defects but rarely changes the final acceptance decision may have a different value from one that prevents dangerous regressions. A faster agent can still lose on total human effort if its outputs are difficult to integrate.

For the multiplayer interface, compare participants using Wonderland with participants using a conventional workspace, holding the agent configuration constant. Randomize task and interface order where practical, account for learning and prior expertise, and separate initial novelty from sustained benefit. Engagement and time spent in the world are descriptive metrics; useful completed work is the central outcome.

10.6 A proposed scoring rule for research battles

For a research contest, declare mandatory acceptance and permission gates first. A candidate that fails a mandatory gate cannot compensate by being cheaper or producing a more persuasive explanation. Among eligible candidates, compare verified task quality; use resource efficiency or elapsed time only according to predeclared tie-breaking rules. This is a proposed research rubric, not a claim about the current production GVE formula.

Eval findings should be deduplicated by underlying defect and linked to a reproduction. Rewards can separately recognize discovery, successful repair, and a regression test that remains valid on a held-out revision. Keep evaluator calibration and appeals observable. This reduces the incentive to overwhelm the opponent with many weak claims and makes the contest's output useful to developers after the battle ends.

11. i.Wonder's three missions and their industry implications

The founder frames i.Wonder Laboratories, Inc. around three connected missions. Together they explain why the company is pursuing both a technical agent architecture and a distinctive creation environment. [16]

11.1 Research, develop, and deploy compound AI

The first mission is to **research, develop, and deploy compound AI systems and compound AI agents**. Its research focus is the interaction among components: how models use tools, how state survives long work, how review improves outcomes, and how modules should be selected and optimized. Its deployment focus is turning these insights into useful, observable, bounded systems.

Julius 3.3 is the first i.Wonder agent introduced under this mission. The relevant achievement would be a demonstrated improvement in verified creation outcomes, supported by reproducible evaluations and transparent limitations. A proprietary core can support this mission if enough interfaces, methods, and results are disclosed for external scrutiny.

11.2 Innovate multiplayer open-world AI creation platforms

The second mission is to **develop, deploy, and innovate multiplayer open-world AI creation platforms**. Wonderland is the flagship expression: human creators and compound agents work in a shared world, with projects, collaboration, imported sidekicks, verification environments, and evaluation battles.

The industry hypothesis is that creation can become a social environment in which learning, making, demonstrating, and improving happen together. A strong ecosystem could let users bring familiar tools, reuse configurations, and collaborate without rebuilding their entire workflow. Those benefits remain hypotheses until compatibility, accessibility, task performance, and sustained use are measured.

11.3 Produce technology perceived as curious, beautiful, and dystopian

The third mission is to **produce technology that humanity sees as curious, beautiful, and dystopian**.

This wording preserves the founder's aesthetic ambition. It points toward unfamiliar and striking experiences that provoke questions about possible futures. Wonder is expressed through the product's form as well as its function.

As a research interpretation, a dystopian aesthetic need not specify a dystopian social outcome. The platform's measurable commitments should include creator agency, intelligible controls, and a clear understanding of what the system can do. The emotional response itself can be studied through user research, while the software's usefulness is evaluated independently. A beautiful world does not prove reliability; reliable work does not determine whether a world evokes wonder.



12. Limitations, discussion, and conclusion

12.1 What this paper does not establish

This paper does not report an executed Julius 3.3 benchmark, production readiness audit, completed live AvA evaluation, or a controlled Wonderland user study. The internal milestone evidence is a report of development work, including simulated flows; it cannot establish present deployment status or broad performance. The AI Jhinn concept needs a precise specification before its behavior can be analyzed.

[16] [17]

The literature is heterogeneous. Studies use different models, tasks, feedback, budgets, and scoring rules. Their results demonstrate possibilities and failure modes, but do not produce a pooled improvement estimate. Historical percentages should not be interpreted as expected performance on a contemporary workload. This paper's company affiliation also creates a motivation to favor the proposed product; explicit hypotheses and comparative experiments are intended to make that proposition contestable.

12.2 Costs and failure modes that matter

Compound systems introduce additional dependencies and interfaces. Retrieval can fail; workers can misunderstand one another; tools can time out; state can become inconsistent; and reviewers can miss defects. A closed core can make external reproduction harder. Open integrations create compatibility and permission-management work. These are central engineering variables that the research program must measure, rather than incidental details to postpone.

Competition creates another tension: optimizing the score may diverge from improving the product. Hidden tests, independent reproduction, declared budgets, and rewards for accepted repairs help align the two, but they do not eliminate gaming. The strongest scientific value of Wonderland's evaluation layer would be a growing record of reproducible tasks, failures, repairs, and configuration comparisons produced under clear rules.

12.3 Conclusion

Compound AI is a promising way to extend foundation models into practical creation systems. Its advantages arise from coordinated access to information, tools, state, adaptive execution, and verification. Published research shows that these choices can materially affect outcomes; it also shows why complexity must earn its place through measurement.

Julius 3.3 and Wonderland express i.Wonder's approach to this opportunity: a primary compound agent with a stable kernel, optimizable modules, durable project context, independent review, and bounded compound clones, operating within a multiplayer creation world. Users are intended to customize their agent experience and connect familiar coding harnesses as sidekicks, while Julius remains a closed core. GVEs and evaluation battles aim to connect the visible world to useful technical evidence.

The decisive research test is whether this complete system helps people create better artifacts with less total effort and acceptable cost. Demonstrating that result would make the case for Julius and Wonderland stronger than any claim based solely on agent count, spectacle, or model scale.

References

- [1] Lewis, P., et al. (2020). [Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks](#). NeurIPS 2020; arXiv:2005.11401.
- [2] Yao, S., et al. (2023). [ReAct: Synergizing Reasoning and Acting in Language Models](#). ICLR 2023; arXiv:2210.03629v3.
- [3] Schick, T., et al. (2023). [Toolformer: Language Models Can Teach Themselves to Use Tools](#). NeurIPS 2023; arXiv:2302.04761.
- [4] Khattab, O., et al. (2023). [DSPy: Compiling Declarative Language Model Calls into Self-Improving Pipelines](#). arXiv:2310.03714.
- [5] Yang, J., et al. (2024). [SWE-agent: Agent-Computer Interfaces Enable Automated Software Engineering](#). NeurIPS 2024; arXiv:2405.15793v3, Table 1.
- [6] Shinn, N., et al. (2023). [Reflexion: Language Agents with Verbal Reinforcement Learning](#). NeurIPS 2023; arXiv:2303.11366v4, Section 4.3 and Table 1.
- [7] Xia, C. S., Deng, Y., Dunn, S., & Zhang, L. (2024). [Agentless: Demystifying LLM-based Software Engineering Agents](#). arXiv:2407.01489v2, 29 October 2024.
- [8] Liu, N. F., et al. (2024). [Lost in the Middle: How Language Models Use Long Contexts](#). Transactions of the Association for Computational Linguistics, 12, 157-173.
- [9] Chen, L., et al. (2025). [Optimizing Model Selection for Compound AI Systems](#). arXiv:2502.14815.
- [10] Wu, Q., et al. (2023). [AutoGen: Enabling Next-Gen LLM Applications via Multi-Agent Conversation](#). arXiv:2308.08155v2.
- [11] Wang, G., et al. (2023). [Voyager: An Open-Ended Embodied Agent with Large Language Models](#). arXiv:2305.16291v2.
- [12] DeBenedetti, E., et al. (2024). [AgentDojo: A Dynamic Environment to Evaluate Prompt Injection Attacks and Defenses for LLM Agents](#). NeurIPS 2024; arXiv:2406.13352.
- [13] Cemri, M., et al. (2025). [Why Do Multi-Agent LLM Systems Fail?](#) arXiv:2503.13657.
- [14] Zheng, L., et al. (2023). [Judging LLM-as-a-Judge with MT-Bench and Chatbot Arena](#). NeurIPS 2023 Datasets and Benchmarks; arXiv:2306.05685v4.
- [15] Jimenez, C. E., et al. (2024). [SWE-bench: Can Language Models Resolve Real-World GitHub Issues?](#) ICLR 2024; arXiv:2310.06770.
- [16] Rogodfree, K., and i.Wonder (2026). *Founder product brief and canonical architecture directions for Julius 3.3 and Wonderland*. Unpublished product-design material supplied or recalled for this paper, August-September 2026.
- [17] i.Wonder (2026). *Wonderland Backend Milestone Report*. Unpublished internal development report, 24 August 2026; supplied as Pasted markdown.md. Reports simulated flows and remaining hosted integration work.