

TECHNICAL WHITEPAPER / PUBLIC RELEASE v1.0 PARS: A Constraint-Driven Reasoning and Verification Architecture for AI Freehand Exact-Binary Generation

**Architecture, Prompting Protocols, Exact-Byte Provenance, Reproducible
Verification, and Historical Evidence**

Author: Rooke Alan Poole

Status: Public Release v1.0

Public scope: Architecture, methods, documented artifacts, evaluation procedures, and evidence already disclosed.
Unpublished extensions, proprietary research directions, and non-public implementation plans are intentionally excluded.

Abstract

Generative AI systems commonly produce software through conventional programming representations such as source code, scripts, or assembly language, after which compilers, assemblers, linkers, interpreters, and related tooling transform those representations into executable artifacts. Direct generation of an executable at the level of its complete byte representation creates a substantially different problem. In addition to specifying intended behavior, the generating system must reason about executable structure, machine-instruction encoding, offsets, data placement, architecture-specific constraints, and the exact serialized artifact that will ultimately execute.

This paper presents **PARS**, a constraint-driven reasoning and verification architecture, and applies it to a class of experiments termed **freehand exact-binary generation**. In these experiments, a generative model is tasked with authoring the complete byte-level representation of a machine-consumable binary while specified conventional generation paths - including compilation, assembly, linking, prebuilt executables, or other undisclosed binary-producing transformations - are excluded from the claimed generation path.

PARS organizes substantive reasoning through the recurring cycle **Parse -> Branch -> Transform -> Perturb -> Test -> Reconstruct -> Build**. New evidence may return execution to earlier stages rather than forcing commitment to a defective candidate. For hard-constrained artifacts, PARS additionally freezes acceptance-critical requirements as explicit invariants and requires them to be replayed against the exact final candidate before Build; materialization-sensitive invariants are tested again after the artifact is built [S1].

The methodology distinguishes executable correctness from provenance. Exact file identity, reconstruction from recorded bytes, and evidence that a conventional compiler, assembler, or linker did not participate in the claimed generation path represent different evidentiary claims. PARS separates these through binary-provenance classes and explicitly rejects the inference that reconstruction identity alone establishes toolchain exclusion [S1]. The same discipline is extended to visual binary artifacts through classes distinguishing finished-output emitters, runtime raster builders, procedural renderers, and reference-conditioned reconstructions.

Accordingly, this methodology does not treat successful execution as sufficient proof of freehand exact-binary generation. A defensible experiment instead combines a frozen generation contract, complete byte representation, independent reconstruction, cryptographic identity, format inspection, contained execution, runtime-output verification, generation-path evidence, adversarial testing, and explicit claim classification. Historical evidence examined here includes compact Linux executables, a directly serialized WebAssembly module, procedural visual and audiovisual generators, a multiprocessing recursive system, a custom virtual computer, byte-level repair lineages, and a separately audited case where a technically successful binary was classified **INCONCLUSIVE** because the stronger upstream provenance claim was not established.

This paper defines the PARS architecture, formalizes the **PARS Freehand Exact-Binary Protocol (PARS-FEBP)**, describes reproducible proof and materialization procedures, presents a prospective evaluation design, and documents historical evidence under explicit claim boundaries. It does not claim that PARS has already been prospectively proven superior to simpler prompting, that exact-binary generation generalizes to arbitrary programs, or that prompt/context adaptation constitutes model-parameter learning.

Keywords: PARS, generative AI, exact binary, direct byte generation, executable provenance, machine code, ELF, WebAssembly, procedural rendering, verification, reproducibility, constrained generation

Reader's Evidence Guide

The paper uses several compact evidence labels. They are defined formally in Chapter 3, but the following guide is provided for quick reference. [See Table 1.](#)

Table 1. Reader's evidence labels and status meanings used throughout the paper.

Label	Meaning
BP0	Exact artifact identity is established.
BP1	The recorded/frozen byte representation reconstructs the exact artifact.
BP2	The audited creation/materialization path supports the stated conventional-toolchain exclusions.
BP3	Creation provenance is unknown or insufficiently evidenced.
BV0	Exact-byte emitter containing finished output.
BV1	Binary-first raster builder constructing the accepted raster at runtime from structural data.
BV2	Procedural runtime renderer computing visible pixels/frames algorithmically.
BV3	Reference-conditioned BV1/BV2 reconstruction.
RL0	In-task adaptation.
RL1	Verified scoped strategy memory.
RL2	Verified controller adaptation.
RL3	Actual model-parameter learning through a training mechanism.
PASS	All applicable acceptance requirements are established.
FAIL	At least one acceptance requirement is demonstrably violated.
INCONCLUSIVE	A required property cannot be established from the available evidence.

Contents

Clickable entries link directly to chapters and sections. Page numbers are frozen from the public-release render.

1. Introduction	9
1.1 Conventional AI-Assisted Software Generation	9
1.2 The Freehand Exact-Binary Problem	9
1.3 Why Direct Byte Generation Is Not Merely "Assembly in Hex"	10
1.4 Execution Is Not Provenance	10
1.5 Why Prompting Becomes an Experimental Contract	10
1.6 The Hard-Invariant Problem	11
1.7 Research Objective and Claim Boundary	11
2. The PARS Architecture	13
2.1 Architectural Thesis	13
2.2 Parse and the Task Contract	15
2.3 Experimental Control Layer	15
2.4 Branch	15
2.5 Transform	16
2.6 Perturb	16
2.7 Test	16
2.8 Reconstruct	16
2.9 Recursive Search	17
2.10 Specialist Modules	17
2.11 Final Invariant Gate	18
2.12 Build and Post-Build Acceptance Replay	18
2.13 Evidence and Claim Boundary	18
2.14 Verified Adaptation	18
2.15 Anti-Self-Confirmation and Controller Lineage	19
3. Operational Model of Freehand Exact-Binary Generation	20
3.1 Operational Definition	20
3.2 Three Experimental Planes	20
3.3 Semantic Generation Versus Literal Materialization	21
3.4 Freeze-Before-Materialization Rule	22
3.5 Binary Provenance Classes	22
3.6 Visual Binary Classes	23
3.7 Runtime Dependency Versus Generation Dependency	24
3.8 Containment for Newly Generated Binaries	24
4. PARS-FEBP: The Prompting Architecture	25
4.1 Why Ordinary Prompting Leaves Escape Routes	25
4.2 The Eleven Prompt Locks	26
4.3 Why the Locks Are Ordered	28
4.4 Classification-Over-Success Principle	28
4.5 Repair After Contract Escape	28
4.6 Independent Audit Role	28
4.7 Prompting Is Not Provenance	28
5. Exact-Byte Materialization and Proof Protocol	29
5.1 Freeze Before Materialization	29
5.2 Literal Materializer Definition	29
5.3 Independent Reconstruction	29
5.4 Byte-for-Byte Identity	29
5.5 Structural Verification	29
5.6 Generation-Path Audit	29
5.7 Contained Execution	29
5.8 Runtime-Result Identity	30
5.9 Finished-Payload Contamination Tests	30

5.10 Post-Build Acceptance Replay	30
5.11 Canonical Proof Chain	30
5.12 Public Showcase Evidence Order	31
6. Prospective Evaluation Protocol	32
6.1 Evaluation Question	32
6.2 Preregistered Hypotheses	32
6.3 Prompting Conditions	32
6.4 Difficulty Ladder	32
6.5 Development and Held-Out Cohorts	33
6.6 Primary Metrics	33
6.7 Constraint-Compliant Exact-Binary Success (CCEBS)	33
6.8 Failure Taxonomy	33
6.9 Retry and Repair Policy	34
6.10 Evaluator Independence	34
6.11 Preregistered Ablations	34
6.12 Severe-Regression Vetoes	35
6.13 Interpretation Rules	35
7. Historical Evidence and Case Studies	36
7.1 Evidence-Selection Method	36
7.2 Worked Example 0 - 907-Byte Starship ELF	36
7.3 Case Study 1 - LHC Collider v2.1	38
7.4 Case Study 2 - WASM Hyperlattice	40
7.5 Case Study 3 - Binary Brains Gate 9	41
7.6 Case Study 4 - PARS-VM/4	42
7.7 Case Study 5 - House of the Rising Sun	43
7.8 Case Study 6 - When a Working Binary Still Cannot Pass the Strict Contract	45
7.9 Cross-Case Synthesis	47
8. Discussion	48
8.1 What PARS Appears to Contribute	48
8.2 Constraint Preservation Versus Candidate Generation	48
8.3 Failure Preservation as Evidence	48
8.4 Correctness and Provenance Are Orthogonal	48
8.5 Why "Freehand" Requires an Operational Definition	48
8.6 Materialization as an Information Boundary	48
8.7 Exact-Byte Generation Is Not a Replacement for Conventional Toolchains	49
8.8 Exact Bytes Do Not Reveal a Unique Internal Cognitive Mechanism	49
8.9 Exact-Binary Repair Does Not Establish Model-Weight Learning	49
8.10 Scaling the Artifact Versus Scaling the Proof	49
9. Limitations, Alternative Explanations, and Threats to Validity	50
9.1 Historical Selection Bias	50
9.2 Incomplete Historical Prompt Archives	50
9.3 Generator/Verifier Correlation	50
9.4 Limits of Provenance Evidence	50
9.5 Runtime Environment Dependence	50
9.6 Capture and Presentation Distortion	50
9.7 Subjective Visual Acceptance	51
9.8 Toolchain-Marker Absence Is Not BP2	51
9.9 Model and Configuration Dependence	51
9.10 Human-Readable Proof Does Not Scale Indefinitely	51
9.11 Learned Binary Knowledge as an Alternative Explanation	51
9.12 Prompt Specificity as an Alternative Explanation	51
9.13 Iterative Debugging as an Alternative Explanation	51
9.14 Task-Family Selection Effects	51
10. Conclusion	52
10.1 Public Disclosure Boundary	52
Appendix A - Terminology and Glossary	53
Appendix B - PARS-FEBP Prompt Protocol v1.0	54

B.1 Minimal Freehand Prompt..... 54

B.2 Strict Research Prompt..... 54

B.3 Maximum-Assurance Prompt..... 55

B.4 Procedural Visual / BV2 Prompt..... 56

B.5 Contract-Repair Prompt..... 56

B.6 Independent Audit Prompt..... 57

Appendix C - PARS-FEBP Evidence Receipt..... 58

Appendix D - PARS-ECS Case Study Template..... 60

Appendix E - Benchmark Preregistration Template..... 61

Appendix F - Historical Artifact Identity Index..... 62

Appendix G - Mechanical Reproduction Examples..... 63

 Literal Hex to Bytes..... 63

 Identity..... 63

Appendix H - Evidence Package Layout..... 64

Appendix I - Primary Source Artifact Index..... 65

List of Figures

Figure 1. Conventional AI-assisted software generation compared with the PARS freehand exact-binary experiment	9
Figure 2. Canonical PARS cycle with backward evidence flow	11
Figure 3. Master PARS layered architecture	14
Figure 4. Recursive search frontier and controller lineage	17
Figure 5. Operational separation used by this paper between semantic generation, literal materialization, and evidence collection	21
Figure 6. BP0-BP3 provenance classes	23
Figure 7. BV0-BV3 visual exact-binary claim classes	24
Figure 8. The eleven PARS-FEBP prompt locks and the failure mode each is intended to prevent	26
Figure 9. Semantic generation versus literal materialization	22
Figure 10. End-to-end exact-binary proof chain	30
Figure 11. Prospective PARS-FEBP evaluation architecture	34
Figure 12. Historical 907-byte Starship evidence flow from retained byte representation through exact reconstruction and deterministic 256×256 PPM output	37
Figure 13. LHC v2.1 runtime architecture	38
Figure 14. Preserved LHC candidate lineage	39
Figure 15. Executable and runtime round-trip identity for LHC v2.1: the frozen representation reconstructs the same ELF and the reconstructed executable reproduces the same deterministic runtime hash	40
Figure 16. Binary Brains Gate 9 causal process architecture	41
Figure 17. PARS-VM/4 virtual-computer architecture: a 5,392-byte x86-64 host implements a custom virtual CPU/ISA, guest memory and monitor, guest-side 3D computation, software VGPU, framebuffer, and raw X11 display transport	43
Figure 18. Historical complexity/evidence ladder	47
Figure 19. BVIS-001 evidence path	46

List of Tables

Table 1. Reader’s evidence labels and status meanings used throughout the paper 8

Table 2. Historical case portfolio, artifact sizes, and primary methodological contribution 47

Table 3. Historical artifact identity index and source-key mapping 62

Appendix Quick Links

Direct navigation to reusable protocols, receipts, templates, evidence-package structure, and source records.

- [Appendix A - Terminology and Glossary](#)
- [Appendix B - PARS-FEBP Prompt Protocol v1.0](#)
- [Appendix C - PARS-FEBP Evidence Receipt](#)
- [Appendix D - PARS-ECS Case Study Template](#)
- [Appendix E - Benchmark Preregistration Template](#)
- [Appendix F - Historical Artifact Identity Index](#)
- [Appendix G - Mechanical Reproduction Examples](#)
- [Appendix H - Evidence Package Layout](#)
- [Appendix I - Primary Source Artifact Index](#)

1. Introduction [Contents](#)

1.1 Conventional AI-Assisted Software Generation

Modern software engineering is built around layered representations. A programmer or generative model typically expresses behavior in a high-level language, an intermediate representation, or assembly. Conventional tooling then performs the mechanical work required to transform that representation into executable bytes.

A simplified pipeline is:

Human objective → generative AI → source/assembly → compiler → assembler → linker → executable → execution

The toolchain is responsible for a large amount of hidden but essential work: instruction encoding, relocation, layout, symbol resolution, section placement, alignment, executable headers, ABI conventions, and target-specific serialization. Generative AI fits naturally into this workflow because it can produce a symbolic program while established software tools handle the lowering process.

There is nothing defective about this pipeline. It is the normal and usually preferable way to build maintainable software. The research question addressed here is deliberately different:

What happens when the final executable byte representation itself becomes the artifact the generative model is required to author? See Figure 1.

FIGURE 1

Conventional software generation vs. freehand exact-binary generation

The experimental change is not the desired behavior; it is where executable semantics become bytes.

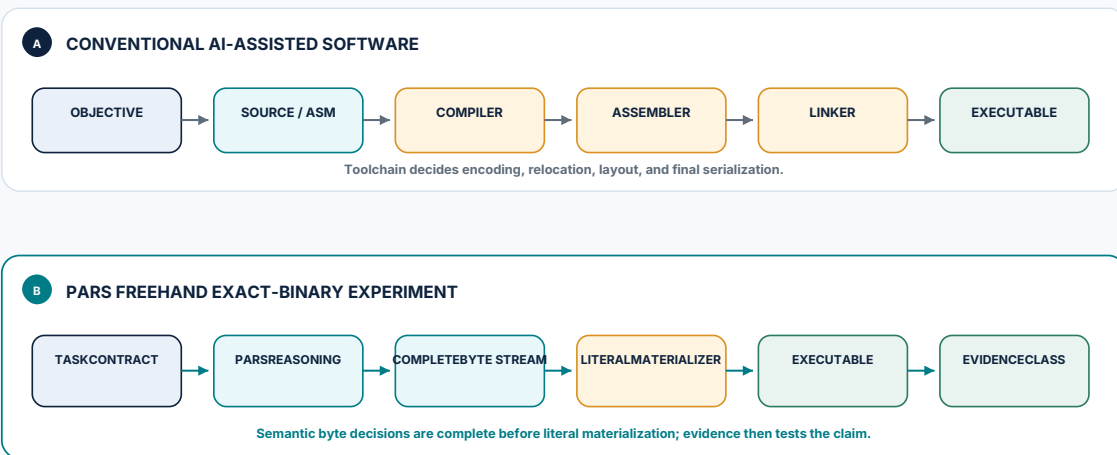


Figure 1. Conventional AI-assisted software generation compared with the PARS freehand exact-binary experiment. The latter moves semantic byte decisions into the authored representation and treats literal materialization and verification as separate planes.

1.2 The Freehand Exact-Binary Problem

This paper uses **freehand exact-binary generation** as an operational term. The term describes an experiment in which the complete byte-level representation of a machine-consumable binary is authored directly, while explicitly prohibited binary-generation mechanisms are excluded from the claimed path.

The experimental transformation is therefore not merely:

source code → binary

but:

task contract → model reasoning → complete exact-byte representation → literal materialization → executable artifact → verification

The distinction matters. If a model writes C and a compiler produces the executable, the model has solved a source-generation problem. If it writes assembly and an assembler encodes the instructions, it has solved an assembly-generation problem. In the experiments examined here, the object under investigation is the model’s ability to reason about the final serialized machine representation sufficiently to specify the executable bytes themselves.

PARS is not treated as a compiler, assembler, or hidden software build engine. The supplied specification defines PARS as the reasoning procedure applied to the user’s actual task [S1]. This boundary is fundamental to every claim in the paper.

1.3 Why Direct Byte Generation Is Not Merely “Assembly in Hex”

At first glance, direct binary generation can look like replacing symbolic instructions with hexadecimal values. That interpretation understates the problem.

A functioning executable can require simultaneous satisfaction of constraints across multiple levels:

behavioral intent → program structure → machine instructions → instruction encodings → branch/address dependencies → executable structure → segment/data layout → serialized bytes → loader behavior → runtime behavior

An error at one level can invalidate assumptions at another. A one-byte change may alter an operand. An instruction-length change may invalidate later relative branches. A corrected branch may change code or data positions. A header field may refer to an offset calculated against an earlier layout. A logically correct instruction sequence can therefore live inside a structurally invalid executable, while a structurally valid executable can still behave incorrectly.

This nonlocal dependency structure is one reason PARS makes **Reconstruct** a mandatory stage after decision-relevant evidence changes. Corrected evidence is not merely appended as a note; affected dependencies and downstream tests are rebuilt [S1].

1.4 Execution Is Not Provenance

A binary executing successfully does not, by itself, establish how it was produced.

Consider four superficially similar demonstrations:

1. AI writes C; a compiler produces a working executable.
2. AI writes assembly; an assembler and linker produce a working executable.
3. AI displays a byte sequence but executes an unrelated prebuilt artifact.
4. AI authors a complete byte representation; the binary is reconstructed from that representation, shown to be byte-identical, executed, and accompanied by audited generation-path evidence.

All four may end with a program that works. Only the fourth supports the stronger experimental claim sought here.

This gives two independent questions:

- **Does the artifact behave correctly?**
- **Does the evidence establish the claimed generation path?**

A positive answer to the first cannot substitute for evidence about the second.

1.5 Why Prompting Becomes an Experimental Contract

Ordinary prompting often leaves implementation choices open. A request such as “make a program that displays a message” permits many valid solutions. Freehand exact-binary experiments require the opposite discipline: the prompt

must define not only the desired behavior but also what representation counts as the artifact, which generation mechanisms are prohibited, what evidence must be preserved, and what conditions invalidate an otherwise functioning candidate.

PARS formalizes this during Parse by identifying the desired artifact, observable success, failure conditions, unknowns, hard negative constraints, claim boundary, and acceptance-critical invariants [S1]. The prompt therefore becomes part of an experimental contract rather than a casual request.

1.6 The Hard-Invariant Problem

Generative systems can produce candidates that are mostly correct while violating one acceptance-critical condition. For exact-binary work, a candidate might:

- execute correctly but have been assembled;
- display the requested image but embed the complete finished raster;
- reconstruct correctly but execute a different file;
- satisfy behavior while targeting the wrong architecture;
- preserve most bytes while omitting a region from the published representation;
- satisfy the artifact contract but overclaim provenance.

PARS candidate.6 is designed to address this class of failure by specifying an unconditional **Final Invariant Gate**. Active hard invariants are replayed against the exact final candidate immediately before Build. A single failure blocks Build; an unverifiable hard invariant remains unverifiable rather than being silently promoted to PASS [S1]. See Figure 2.

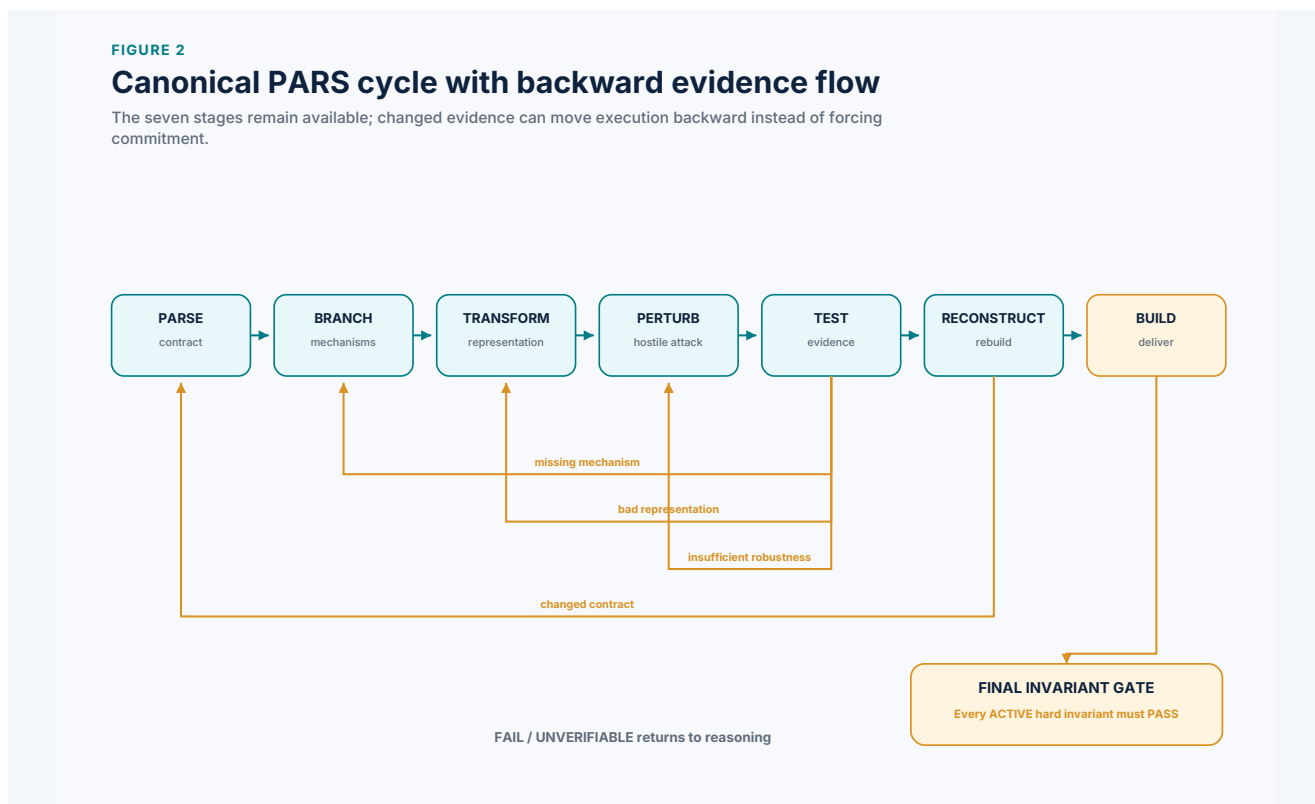


Figure 2. Canonical PARS cycle with backward evidence flow. Changed mechanism, representation, robustness, evidence, or task-contract state can return execution to earlier stages; Build remains gated by final-invariant replay [S1].

1.7 Research Objective and Claim Boundary

The central architectural proposition of this paper is:

Freehand exact-binary generation becomes a more defensible experimental procedure when generation is governed by an explicit constraint contract and when artifact correctness, byte identity, generation provenance, runtime behavior, and claim strength are verified separately.

A stronger proposition must be tested rather than assumed:

Does PARS materially improve a generative model's rate of successful, constraint-compliant freehand exact-binary generation relative to simpler prompting?

The historical cases in this paper motivate and illustrate the architecture. They are not used as prospective comparative proof of superiority. The supplied candidate.6 specification itself states that the candidate does not by itself establish superior reasoning, model-weight self-learning, or verified recursive self-improvement [S1].

2. The PARS Architecture [Contents](#)

2.1 Architectural Thesis

PARS separates five questions that are often conflated in generative-AI work:

1. What is the actual task contract?
2. How should candidate solutions be explored?
3. How should evidence alter those candidates?
4. What must be true before a result can be accepted?
5. What does the resulting evidence permit us to claim?

The resulting architecture can be summarized as:

Contract → Search → Evidence → Reconstruction → Invariant Verification → Build → Post-Build Verification → Claim

The canonical reasoning engine remains:

Parse → Branch → Transform → Perturb → Test → Reconstruct → Build [\[S1\]](#)

Additional layers determine how the engine is controlled, recursively expanded, checked, and prevented from silently weakening the task.

2.1.1 Specification Status and Authority Boundary

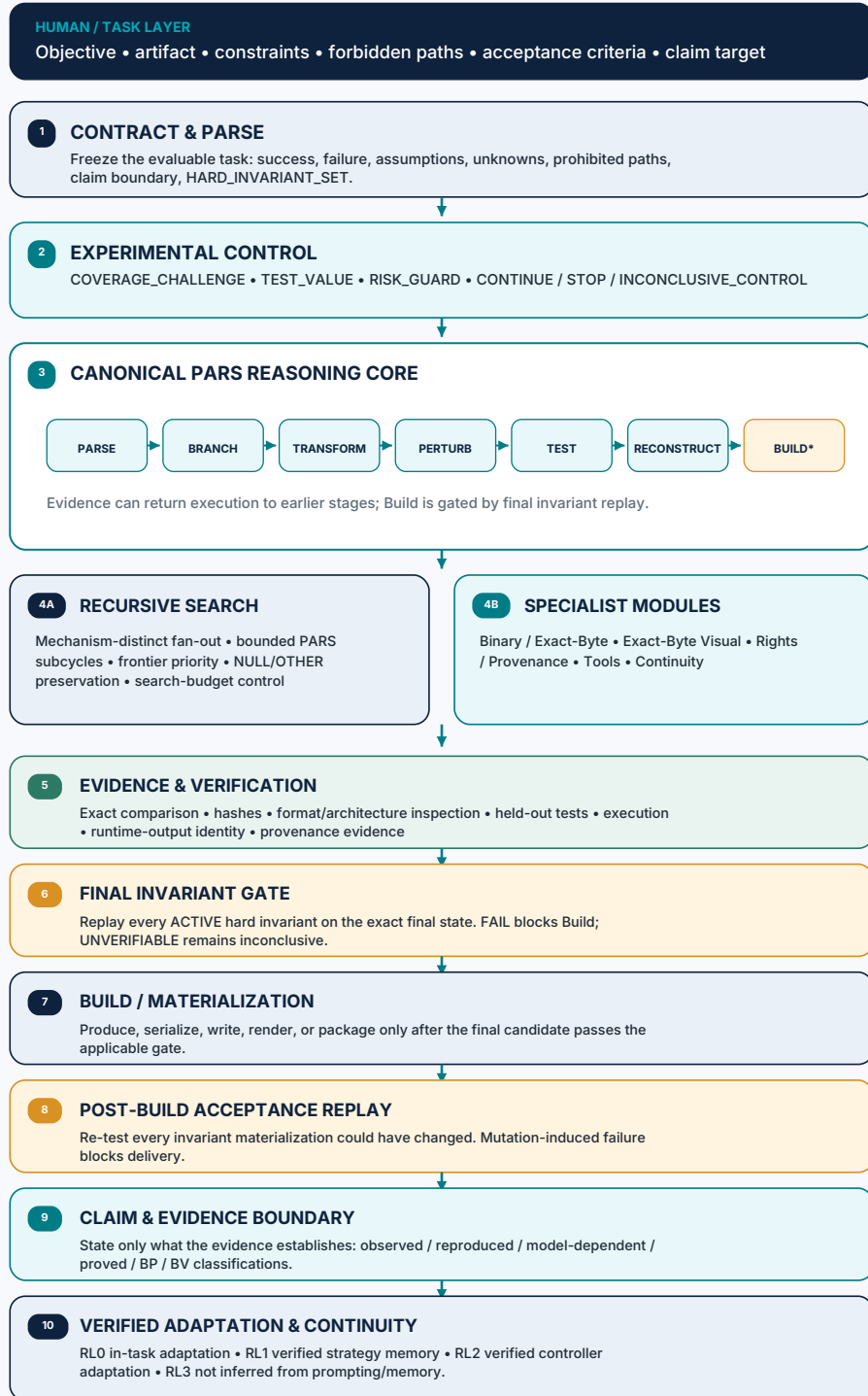
STATUS BOUNDARY The architecture documented in this paper is sourced primarily from the supplied PARS v1.25.0-candidate.6 specification. That source explicitly labels candidate.6 a non-authoritative experimental repair candidate and identifies candidate.4 as the current passing candidate pending prospective gates [\[S1\]](#). The source also preserves inherited parent/authority statements inside earlier sections. This paper therefore uses candidate.6 as the architectural specification being documented; it does not treat citation of candidate.6 as evidence that candidate.6 has been promoted to authoritative PARS or independently validated as superior.

Experimental layers described below - including the Control layer, recursive search/verified adaptation, and the Final Invariant Gate repair - are reported as **specified architecture** unless a separate evidence source establishes more. [See Figure 3.](#)

FIGURE 3

Master PARS layered architecture

Contract-driven reasoning, recursive search, verification, materialization, and evidence-bounded claims.



PARS is the reasoning procedure; verification and evidence gates bound what may be claimed.

Figure 3. Master PARS layered architecture. Whitepaper synthesis of the supplied candidate.6 architecture; candidate.6 remains experimental/non-authoritative pending its specified prospective gates [S1].

2.2 Parse and the Task Contract

Parse converts an informal request into an evaluable state. Depending on the task, the state includes:

- the actual question;
- the desired result or artifact;
- observable success;
- failure conditions;
- facts, assumptions, preferences, and unknowns;
- explicit prohibited tools or generation paths;
- rights/provenance state when relevant;
- exact-byte or visual claim class;
- the narrowest defensible claim;
- the hard-invariant set;
- whether Build can mutate any invariant;
- the applicable adaptation class and controller version.

Parse is not complete while the request remains too ambiguous to evaluate. In exact-binary work, this is particularly important because “make a binary” and “directly author every executable byte without a compiler, assembler, linker, or prebuilt binary” describe materially different experiments.

2.3 Experimental Control Layer

The Control layer determines where reasoning effort should go without deleting required capabilities. It introduces three central ideas [S1].

COVERAGE_CHALLENGE

Before robust commitment, ask whether a plausible observation or mechanism class could materially change the claim yet remain unexplained by the current branches. If yes, return to Branch. The goal is mechanism diversity, not branch-count inflation.

TEST_VALUE

When multiple valid tests exist, prefer those with greater decision-relevant value relative to cost. A test matters when it can alter the selected action, surviving branch set, claim boundary, reconstruction state, or important tail risk.

RISK_GUARD

Average-case efficiency must not eliminate a low-probability NULL/OTHER branch when missing it could be severe and a feasible discriminating check exists.

The Control layer can produce **CONTINUE**, **STOP**, or **INCONCLUSIVE_CONTROL**. **STOP** terminates further evidence acquisition, not Build. Control cannot override evidence integrity, required branching, hostile perturbation, provenance, Reconstruction, claim boundaries, or hard-invariant verification.

2.4 Branch

Branch creates mechanism-distinct candidate explanations or solution paths. PARS-Deep calls for five or more branches when justified, preserves a NULL/OTHER possibility, and expects at least two candidates capable of making different predictions before robust commitment [S1].

In exact-binary generation, branches might represent:

- different executable layouts;
- direct syscall versus imported API strategies;
- compact interpreters versus direct logic;
- procedural drawing versus payload emission;
- alternative data-layout and control-flow schemes;

- the NULL possibility that the target is not feasible under the current artifact or resource constraints.

The purpose is to prevent the first plausible construction from becoming an unquestioned commitment.

2.5 Transform

Transform changes the representation of the problem when the current representation hides structure. Useful transforms include:

- desired behavior -> machine-level operations;
- machine operations -> instructions;
- instructions -> opcodes and immediate fields;
- symbolic addresses -> concrete offsets;
- visual target -> geometric primitives;
- bug symptom -> state/event flow;
- failure -> invariant violation;
- large artifact -> local dependency graph.

A useful transform should reveal a testable relation or a hidden failure. In binary work, representation choice often determines whether the problem is tractable at all.

2.6 Perturb

PARS attempts to break leading candidates rather than merely confirm them. Applicable hostile perturbations include boundary changes, corrupted inputs, assumption deletion, implementation-failure injection, evidence reorder, provenance correction, prohibited-tool contamination, and finished-payload contamination [S1].

For a claimed procedural renderer, for example, a hostile perturbation may search the executable for the accepted finished image. Discovery of such a payload could collapse or downgrade a BV2 claim even if the visual result is excellent.

2.7 Test

PARS ranks evidence by discriminating strength. Proofs, exact oracles, independent measurements, controlled experiments, and held-out predictions outrank aesthetic judgment or model intuition. Important tests preserve what was tested, the method, result, pass/fail criterion, limitations, and version/hash where relevant [S1].

For exact binaries, stronger tests include:

- byte-for-byte comparison;
- SHA-256 identity;
- independent file-format inspection;
- deterministic output comparison;
- contained runtime execution;
- process-path evidence;
- hostile payload and provenance tests.

A check must not be described as executed unless evidence establishes that it actually ran and reached the final state.

2.8 Reconstruct

Reconstruction is mandatory when decision-relevant evidence changes. PARS removes invalid influence without erasing history, inserts corrected evidence, rebuilds dependencies, reruns affected tests, preserves unaffected work, and revises the claim boundary [S1].

This is central to byte-level work because local changes can have nonlocal consequences. If one instruction changes length, downstream branch displacements, data positions, and file offsets may all need to be recomputed.

The key question is:

If the strongest supporting assumption is deleted, does the conclusion survive?

2.9 Recursive Search

Candidate.6 extends ordinary branching with bounded recursive fan-out. A difficult branch may invoke a PARS subcycle when subsearch can materially reduce uncertainty, expose a new mechanism, or distinguish competing actions [S1].

Each recursive state may preserve the subproblem, branch mechanism, assumptions, hard constraints, inherited evidence, test target, failure modes, remaining budget, and parent/child lineage. Expansion is justified by decision relevance, uncertainty reduction, mechanism novelty, falsifiability, tail-risk coverage, repair value, and feasible cost.

Recursive breadth is not an objective in itself. Semantic duplicates are penalized, NULL/OTHER coverage is preserved, and stopped subtrees remain historically recoverable. See Figure 4.

FIGURE 4

Recursive search and controller lineage

Search fan-out and controller evolution are separate: branches explore task states; controller changes require explicit lineage and promotion evidence.

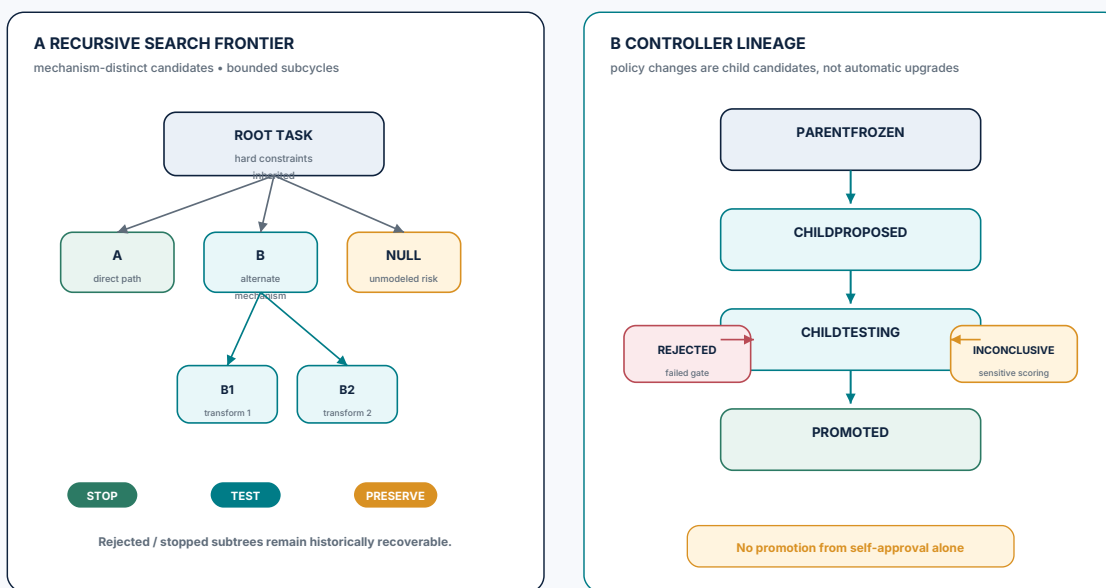


Figure 4. Recursive search frontier and controller lineage. Mechanism-distinct task branches are separate from controller mutations, which remain child candidates until promotion evidence supports them [S1].

2.10 Specialist Modules

The canonical seven-stage cycle is surrounded by specialist modules that activate when applicable.

Binary / Exact-Byte Module

Defines BP classes, exact-identity requirements, contained execution for unknown binaries, and exact repair discipline.

Exact-Byte Visual Module

Defines BV classes, finished-payload contamination checks, binary-first claim boundaries, and proof-video ordering.

Rights / Public-Domain Module

Separates rights evidence from tool-policy failures when external references matter.

Tools Module

Controls how external evidence is gathered and preserves prohibited tools as hard task constraints.

Continuity Module

Preserves authoritative artifacts, failures, provenance locks, controller lineage, and next moves across long-running projects.

2.11 Final Invariant Gate

During Parse, every acceptance-critical hard requirement is frozen with an ID, source, predicate, scope, verifier, and status. Immediately before Build, the exact selected final candidate is instantiated and every active invariant is replayed [S1].

Possible outcomes are:

- **PASS** - invariant established;
- **FAIL** - candidate becomes INVALID_FINAL_STATE and Build is forbidden;
- **UNVERIFIABLE** - evidence must be obtained or the result remains inconclusive.

A contradictory contract returns to Parse as an inconsistent contract rather than silently deleting a constraint. Search confidence, majority vote, prior partial checks, or aesthetic quality cannot override a failed invariant.

2.12 Build and Post-Build Acceptance Replay

Build produces or materializes the requested artifact, but it is gated. When Build itself can change an invariant - through serialization, packaging, compression, rendering, or writing - the affected invariants are replayed after materialization [S1].

This matters for exact binaries because a logically correct candidate can be corrupted while written to disk. A pre-Build PASS therefore does not authorize delivery of a mutated post-Build artifact.

2.13 Evidence and Claim Boundary

PARS requires the conclusion to be no stronger than the evidence. It prohibits upgrades such as:

- simulation -> theorem;
- synthetic success -> real-world validation;
- exact-byte emitter -> procedural renderer;
- prompt/context adaptation -> model-weight learning;
- self-scored controller success -> verified recursive improvement.

Every case study in this paper therefore distinguishes what was observed, what was reproduced, what provenance class is supported, what remains unverified, and what stronger claim is not established.

2.14 Verified Adaptation

PARS separates four learning classes [S1]:

- **RL0 - In-task adaptation:** evidence changes reasoning/search behavior during the current task.
- **RL1 - Verified strategy memory:** a scoped reasoning operator is retained with its trigger, evidence, scope, failures, provenance, and status.
- **RL2 - Verified controller adaptation:** search policy changes are promoted only after parent-versus-child evaluation.
- **RL3 - Model-parameter learning:** foundation-model weights or equivalent learned parameters are actually updated through a training mechanism.

PARS candidate.6 specifies RL0-RL2 procedures. It explicitly does not treat prompting, context, memory, or repeated inference as RL3.

2.15 Anti-Self-Confirmation and Controller Lineage

A controller or strategy cannot be promoted solely because it generated both the proposal and the favorable judgment. PARS prefers deterministic verifiers and exact oracles, then held-out evaluation, controlled benchmarks, independent measurement, and evaluator separation [S1].

Controller revisions follow an explicit lineage:

**PARENT_FROZEN → CHILD_PROPOSED → CHILD_TESTING → PROMOTED / REJECTED /
INCONCLUSIVE → optional repair → retest**

Promotion history remains visible, failed children remain evidence, and rollback returns to the last passing controller when later evidence exposes a regression.

3. Operational Model of Freehand Exact-Binary Generation [Contents](#)

3.1 Operational Definition

For this paper:

A freehand exact binary is a machine-consumable binary artifact whose complete byte-level representation is authored directly under a frozen generation contract, with the claimed prohibited generation mechanisms excluded from the audited path, and whose published representation can be losslessly reconstructed into the accepted artifact.

This is an operational definition for reproducible experiments, not a claim that the term is already standardized in academic literature.

3.2 Three Experimental Planes

The method separates three planes. The **Reasoning Plane / Materialization Plane / Evidence Plane** vocabulary is an analytical organization introduced by this paper to make the experimental boundary explicit; these plane names are not presented as verbatim module names from [\[S1\]](#).

Reasoning Plane

The human objective is transformed into a frozen task contract and then processed through PARS reasoning, search, perturbation, testing, and reconstruction until a complete candidate byte representation is selected.

Materialization Plane

The already-complete representation is converted into literal bytes without adding semantic executable information.

Evidence Plane

Hashes, byte counts, format inspection, reconstruction, execution, runtime-output verification, capture, and provenance audits evaluate the artifact without being misrepresented as generators. [See Figure 5.](#)

FIGURE 5

Generation, materialization, and evidence planes

The experiment separates semantic authorship from literal byte writing and from verification infrastructure.

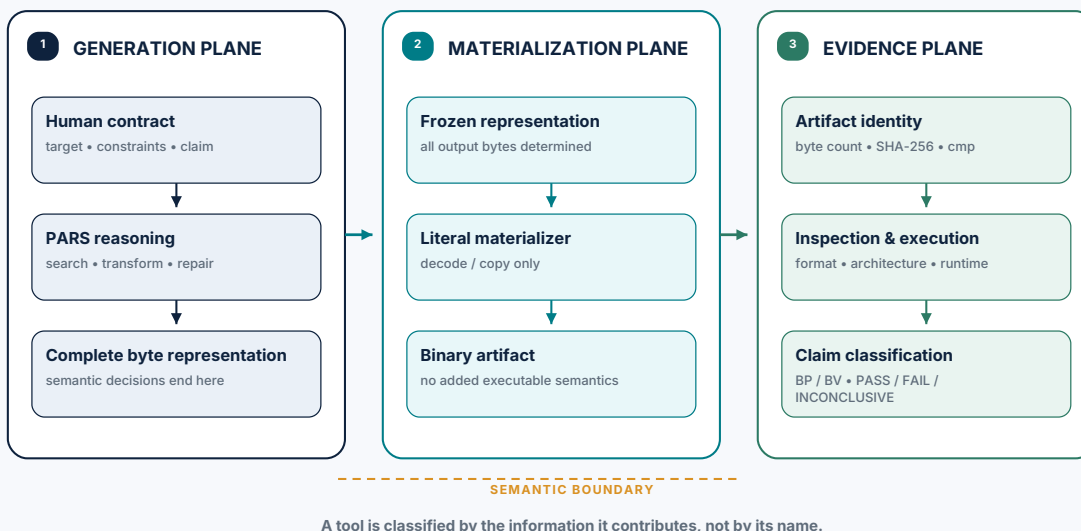


Figure 5. Operational separation used by this paper between semantic generation, literal materialization, and evidence collection.

3.3 Semantic Generation Versus Literal Materialization

The key boundary is not the name of the tool. It is whether the step contributes information about what the executable bytes should be.

For this paper, a **literal materializer** is defined as a mechanism that converts a complete frozen representation into the corresponding byte stream without deciding instruction encodings, resolving symbols, generating executable structures, performing relocations, synthesizing code, or linking components. The term is a paper-level operational definition rather than a named component of [S1].

Examples of permitted materialization operations include:

- hexadecimal pair -> literal byte;
- Base64 -> original frozen bytes;
- escaped byte literal -> corresponding byte;
- concatenation of already-frozen chunks in predetermined order.

Examples of semantic generation include:

- mnemonic instruction -> opcode;
- label -> branch displacement;
- symbol -> resolved address;
- high-level executable metadata -> generated header;
- library/object resolution -> linked output.

The practical test is:

If the materializer were replaced by another correct literal decoder, are all output bytes already completely determined?

If yes, the representation carries the artifact. If no, the step is contributing to generation. See Figure 9.

FIGURE 9

Semantic generation vs. literal materialization

A tool is classified by the information it contributes, not by the name of the language or executable used.

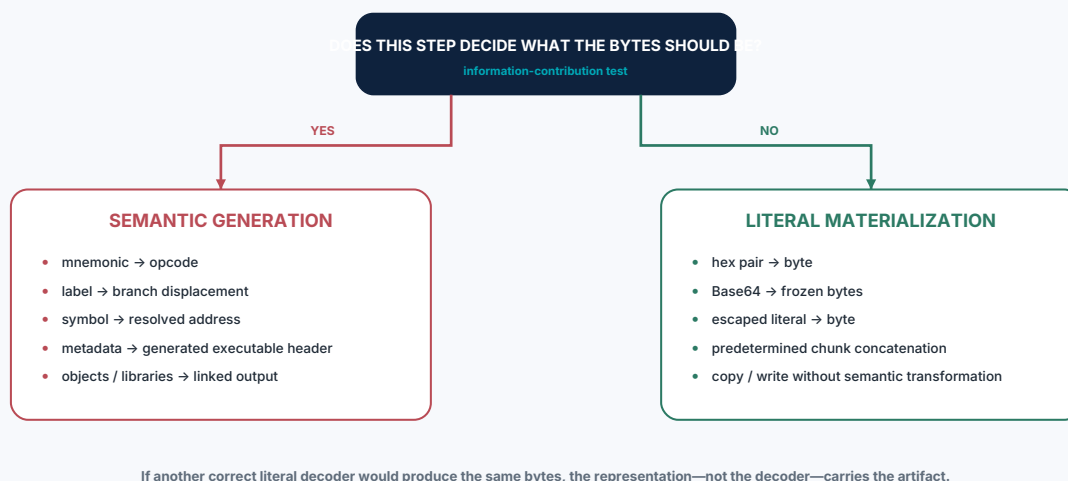


Figure 9. Semantic generation versus literal materialization. The operational boundary is whether the step contributes information about what the executable bytes should be.

3.4 Freeze-Before-Materialization Rule

The complete accepted representation should be frozen before the accepted artifact is materialized. Otherwise, a hidden feedback loop could allow the writer to discover a problem, silently change bytes, and present the modified stream retrospectively.

Any change after freeze creates a new candidate revision. The old candidate remains in failure history.

3.5 Binary Provenance Classes

PARS defines a provenance ladder [S1].

BP0 - Exact Artifact Identity

The specific file under discussion is frozen and identified.

BP1 - Recorded Bytes Reproduce the Artifact

The disclosed representation reconstructs the exact accepted artifact.

BP2 - Audited Creation Path

Evidence supports the stated absence of a conventional compiler, assembler, or linker in the claimed creation/materialization path.

BP3 - Creation Provenance Unknown

The artifact may exist and function, but the creation path is not adequately established.

BP1 does not imply BP2. Missing compiler metadata also does not prove BP2 [S1]. See Figure 6.

FIGURE 6

BP0–BP3 binary-provenance classes

Reconstruction identity and creation-path evidence are different claims. BP1 does not imply BP2.

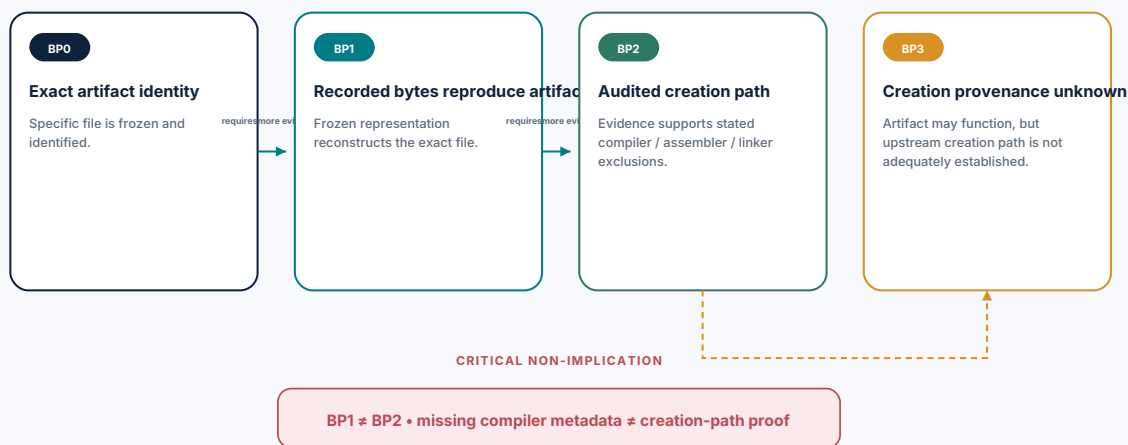


Figure 6. BP0–BP3 provenance classes. Exact artifact identity, byte-stream reconstruction, and audited creation-path evidence are distinct claims; BP1 does not imply BP2 [S1].

3.6 Visual Binary Classes

The visual module separates four classes [S1].

BV0 - Exact-Byte Emitter

The binary contains a finished raster/video payload or equivalent finished output and emits it. Exact-byte delivery may be proven, but binary-first rendering is not.

BV1 - Binary-First Raster Builder

The binary contains compact structural data or primitives and first materializes the accepted raster at runtime.

BV2 - Procedural Runtime Renderer

The binary computes visible pixels or frames algorithmically at runtime and contains no finished frame/video payload.

BV3 - Reference-Conditioned Reconstruction

BV1 or BV2 where supplied or verified references constrain the reconstruction.

A visually convincing output is not sufficient for BV2. Finished-payload contamination tests remain necessary. See Figure 7.

FIGURE 7

BV0–BV3 visual exact-binary claim classes

The classification follows where the accepted visual is materialized and whether external references constrain it.

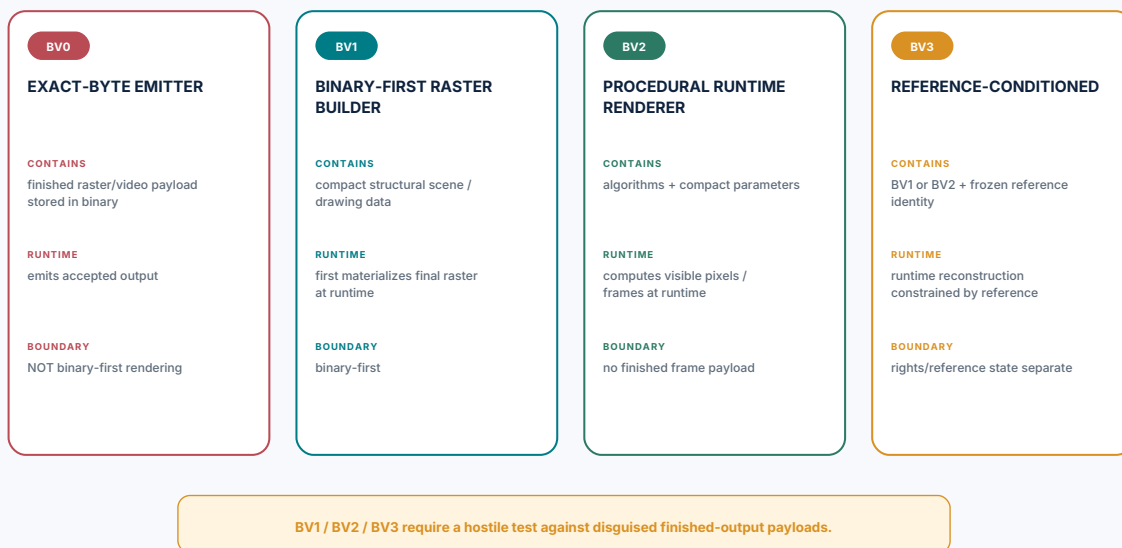


Figure 7. BV0–BV3 visual exact-binary claim classes. BV1/BV2/BV3 require discrimination against a disguised finished-output emitter [S1].

3.7 Runtime Dependency Versus Generation Dependency

A binary may depend on an operating system, display server, WebAssembly host, graphics driver, or another runtime service without those dependencies becoming part of the generation path. The paper therefore distinguishes:

- **generation dependencies** - components that determine executable semantics or bytes;
- **runtime dependencies** - infrastructure required to execute/display the artifact;
- **evidence dependencies** - tools used to inspect, record, hash, or compare it.

This separation is crucial when evaluating artifacts such as WebAssembly modules, raw X11 clients, or GLX/OpenGL programs.

3.8 Containment for Newly Generated Binaries

Unknown or newly generated binaries should be statically inspected before execution and run in a disposable, non-root, credential-free environment with outbound networking denied by default and bounded resources [S1]. This improves both safety and reproducibility.

4. PARS-FEBP: The Prompting Architecture [Contents](#)

4.1 Why Ordinary Prompting Leaves Escape Routes

A prompt such as “make a binary program” leaves many valid interpretations. A model may return C, assembly, shell commands, a compiler invocation, an encoded prebuilt file, or pseudocode. Those outputs may be useful in ordinary development but do not satisfy the exact-binary experiment.

PARS-FEBP therefore uses a stack of **constraint locks**. Each lock closes a different escape route and is tied to a specific failure mode.

Methodology provenance note. The names **PARS-FEBP**, **Eleven Prompt Locks**, and **Classification-Over-Success Principle** are formalizations introduced by this paper from the requirements in [S1] and the retained historical workflow. They should not be read as verbatim named modules from candidate.6. [See Figure 8.](#)

FIGURE 8

Eleven-lock PARS prompt architecture

Each lock closes a distinct escape route before the result can be classified.

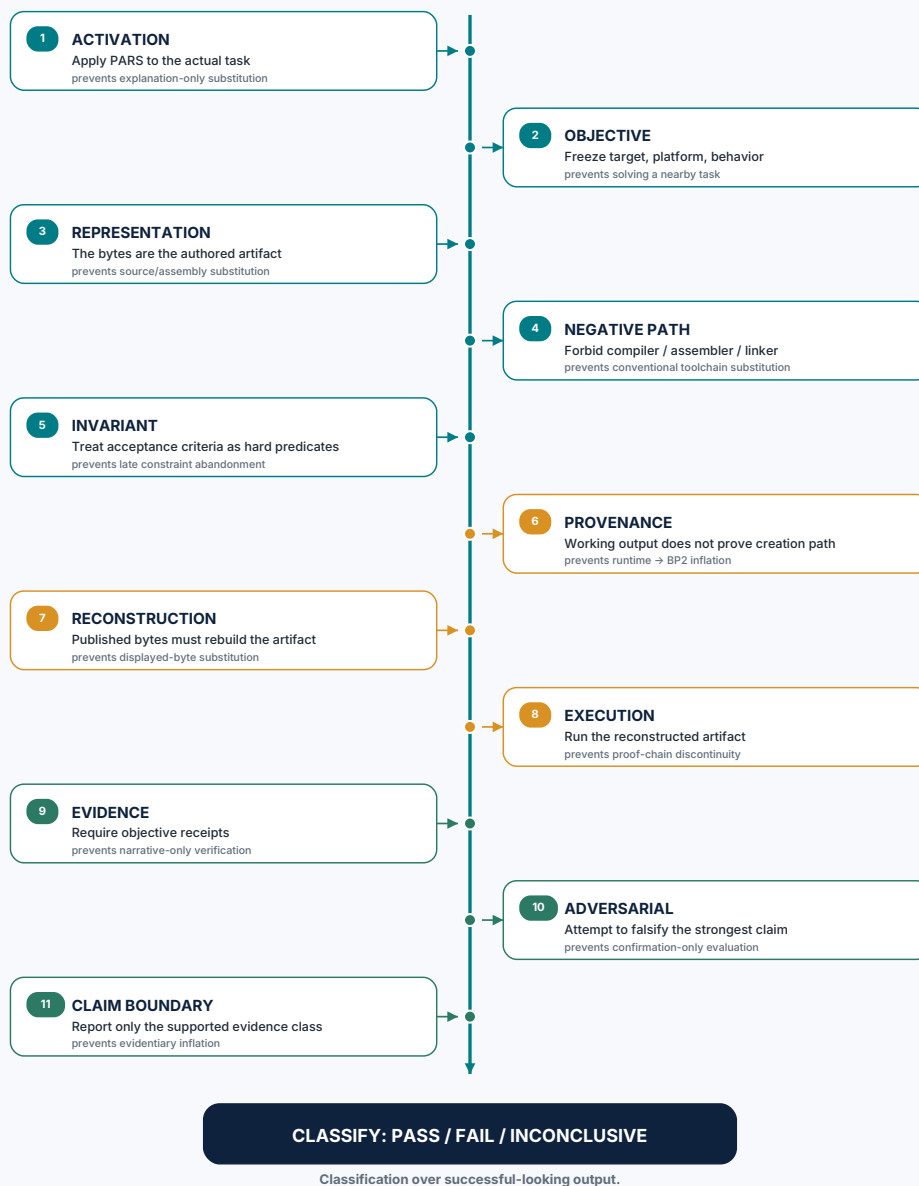


Figure 8. The eleven PARS-FEBP prompt locks and the failure mode each is intended to prevent. PARS-FEBP is a methodology formalized by this paper rather than a verbatim named candidate.6 module.

4.2 The Eleven Prompt Locks

1. Activation Lock

Select PARS-Deep as the reasoning procedure and require it to be applied to the actual task rather than merely described.

Failure prevented: explanatory substitution instead of artifact production.

2. Objective Lock

Freeze platform, format, architecture, required behavior, inputs, outputs, and termination conditions.

Failure prevented: solving a nearby but different task.

3. Representation Lock

State that the complete executable byte representation is the artifact being authored. Source code, assembly source, pseudocode, or build instructions do not satisfy the experiment.

Failure prevented: symbolic-source substitution.

4. Negative-Path Lock

Explicitly forbid compiler, assembler, linker, source-language binary generation, prebuilt executables, prebuilt objects, hidden binary substitution, and any other contract-specific prohibited path.

Failure prevented: conventional-toolchain substitution.

5. Invariant Lock

Treat every acceptance condition as a hard invariant. A candidate violating one active invariant is invalid regardless of whether it works.

Failure prevented: late constraint abandonment.

6. Provenance Lock

State that successful execution does not prove generation provenance and that missing metadata cannot be promoted to a no-toolchain claim.

Failure prevented: runtime success being mistaken for BP2.

7. Reconstruction Lock

Freeze a complete representation and require an independently reconstructed artifact to compare byte-for-byte with the accepted artifact.

Failure prevented: displayed-byte/executed-artifact substitution.

8. Execution Lock

Execute the reconstructed artifact, not an earlier or separately produced development copy.

Failure prevented: proof-chain discontinuity.

9. Evidence Lock

Require byte count, SHA-256, format, architecture, reconstruction identity, exit status, observable behavior, runtime-output identity where applicable, and provenance evidence.

Failure prevented: narrative-only verification.

10. Adversarial Lock

Require at least one hostile test capable of falsifying the strongest generation/provenance claim.

Failure prevented: confirmation-only evaluation.

11. Claim-Boundary Lock

Report only the strongest class supported by evidence. Do not promote execution to provenance, BP1 to BP2, BV0 to BV2, or prompt adaptation to model-weight learning.

Failure prevented: evidentiary inflation.

4.3 Why the Locks Are Ordered

The sequence progressively constrains the experiment:

Objective Lock prevents the wrong task. Representation Lock prevents source substitution. Negative-Path Lock prevents conventional-toolchain substitution. Invariant Lock prevents later escape. Provenance Lock prevents working output from swallowing the creation-path question. Reconstruction and Execution Locks connect published bytes to the artifact that actually runs. Evidence and Adversarial Locks turn claims into tests. Claim-Boundary Lock prevents the paper from becoming stronger than its evidence.

This causal structure later enables ablation: if a lock is removed, the benchmark can test whether the predicted failure reappears.

4.4 Classification-Over-Success Principle

A central PARS-FEBP principle is:

Do not optimize for obtaining PASS. Optimize for correctly distinguishing PASS, FAIL, and INCONCLUSIVE.

This changes the implied objective from producing a successful-looking artifact to correctly classifying the experimental state. Failure preservation, hostile testing, and the allowance for INCONCLUSIVE all follow from this principle.

4.5 Repair After Contract Escape

When a candidate violates a hard invariant, the repair instruction should not simply say “try again.” The violated predicate is identified, the candidate is marked invalid, unaffected evidence is preserved, and PARS returns to Reconstruct. The repair must not weaken or delete the invariant that caused the failure.

A repaired candidate is not a PASS until the complete final gate and all materialization-sensitive post-Build checks pass again.

4.6 Independent Audit Role

The generator and verifier need not have identical roles. An independent audit can be instructed to classify rather than rescue the artifact, actively search for evidence that would downgrade the claimed BP/BV class, and return the strongest class actually established.

This role separation is especially valuable when the original generator would otherwise be evaluating its own provenance narrative.

4.7 Prompting Is Not Provenance

A strict instruction such as “do not use a compiler” creates a contractual obligation. It does not prove compliance.

A prompt creates an obligation. Evidence establishes whether the obligation was satisfied.

This distinction prevents circular reasoning of the form: “the model was told not to use a compiler, therefore no compiler was used.”

The complete reusable prompt bodies are provided in Appendix B.

5. Exact-Byte Materialization and Proof Protocol [Contents](#)

5.1 Freeze Before Materialization

The accepted byte representation is frozen before materialization. Its length and identity are recorded. Any byte change after freeze becomes a new candidate revision.

5.2 Literal Materializer Definition

A literal materializer performs a lossless representation-to-byte conversion without adding executable semantics. It does not generate opcodes, calculate labels, perform relocation, create headers from symbolic descriptions, or link external code.

5.3 Independent Reconstruction

High-assurance experiments should reconstruct the accepted artifact again from the same frozen representation, ideally through a separate simple literal-decoding path.

One path demonstrates materialization; the second reduces the chance that a shared transformation mistake is mistaken for artifact identity.

5.4 Byte-for-Byte Identity

Let A be the accepted artifact and B the reconstructed artifact. Exact identity requires:

- equal byte length; and
- $A[i] = B[i]$ for every byte position.

SHA-256 provides a compact identity receipt, but direct byte comparison remains the primary equality predicate.

5.5 Structural Verification

After identity is established, the artifact is inspected independently for format, architecture, entry point, program/section layout, dynamic loader state, and any other frozen structural requirements.

These inspection tools operate in the evidence plane. They are not credited with generation.

5.6 Generation-Path Audit

BP₂ is harder than BP₁ because reconstructing published bytes does not establish how those bytes were originally derived. A BP₂ experiment therefore preserves evidence about the creation session and any tools that could have influenced the final stream.

The defensible formulation is not metaphysical omniscience. It is:

The observed and audited generation path supports BP₂ under the stated experimental environment and evidence procedure.

Where an upstream gap remains, the correct classification is weaker or inconclusive.

5.7 Contained Execution

The reconstructed artifact is executed in a controlled environment. The experiment records the exact artifact identity, launch method, exit status, observable behavior, and runtime outputs.

Unknown binaries are run non-root, without credentials, with network denied unless required, and under resource limits.

5.8 Runtime-Result Identity

When the binary produces another artifact - a raster, audio stream, file, frame sequence, or deterministic state - that output is independently identified by byte count and hash when appropriate.

Executable identity and runtime-output identity are recorded separately.

5.9 Finished-Payload Contamination Tests

For BV1/BV2/BV3 claims, the executable is tested to determine whether it merely contains the accepted finished raster/video in disguised or compressed form. Appropriate tests include payload search, size/layout inspection, output-expansion analysis, file/runtime-read tracing, and direct examination of structural data.

Output expansion is supporting evidence, not proof by itself.

5.10 Post-Build Acceptance Replay

After materialization, affected invariants are replayed. Byte count, SHA-256, direct identity, structural properties, and any generation-path-sensitive conditions are checked against the built artifact.

5.11 Canonical Proof Chain [See Figure 10.](#)



Figure 10. End-to-end exact-binary proof chain. A defensible classification is assembled through contract, identity, reconstruction, execution, provenance, hostile testing, and claim-boundary checkpoints rather than inferred from execution alone.

The canonical sequence is:

1. Freeze task contract.
2. Generate complete byte representation.
3. Freeze representation identity.
4. Replay final invariants.
5. Literal materialization.
6. Post-Build replay.
7. Byte count and SHA-256.
8. Format/architecture audit.
9. Independent reconstruction.
10. Byte-for-byte identity.
11. Generation-path audit.
12. Contained execution.
13. Runtime-result verification.
14. Hostile contamination/provenance test.
15. BP/BV classification.
16. PASS / FAIL / INCONCLUSIVE.

5.12 Public Showcase Evidence Order

For public demonstrations, the proof reel should place evidence before visual payoff:

Source/reference → provenance lock → target contract → complete exact byte stream → reconstruction → byte count/hash/format → execution → runtime result → post-runtime identity → RESULTS [S1]

The dedicated RESULTS segment is important because a terminal execution line alone does not clearly state the final claim class or remaining limitations.

The standardized evidence receipt is provided in Appendix C.

6. Prospective Evaluation Protocol [Contents](#)

6.1 Evaluation Question

The stronger empirical question is:

Does PARS-FEBP improve correctly verified, constraint-compliant exact-binary generation relative to simpler prompting conditions?

This chapter defines the prospective test required to answer that question. It does not report comparative results.

6.2 Preregistered Hypotheses

H1 - Constraint Compliance

H1 predicts that PARS-FEBP Strict will produce fewer hard-contract violations than naive or ordinary detailed prompting.

H2 - Reconstruction Reliability

H2 predicts that PARS-FEBP conditions will produce a higher proportion of candidates whose frozen representation reconstructs the exact accepted artifact.

H3 - Classification Accuracy

H3 predicts that PARS-FEBP Strict/Maximum Assurance will more accurately distinguish PASS, FAIL, and INCONCLUSIVE.

H4 - Provenance Discipline

H4 predicts that PARS-FEBP conditions will make fewer unsupported BP2 claims when only BP0/BP1 evidence exists.

H5 - Final-Invariant Protection

H5 predicts that the full Final Invariant Gate will reduce delivery of plausible-but-invalid final artifacts.

6.3 Prompting Conditions

Four primary conditions are frozen:

- **A - Naive request:** desired artifact, minimal methodological constraints.
- **B - Explicit exact-binary prompt:** asks for direct bytes and prohibits conventional toolchain, but without full PARS-FEBP architecture.
- **C - PARS-FEBP Minimal:** core locks, byte identity, reconstruction, execution, and claim classification.
- **D - PARS-FEBP Strict/Maximum Assurance:** explicit invariant ledger, provenance separation, hostile testing, Final Invariant Gate, post-Build replay, and failure preservation.

The underlying model, environment, and resource envelope remain constant within each benchmark cohort.

6.4 Difficulty Ladder

The benchmark increases structural difficulty:

- **Level 0:** controlled termination.
- **Level 1:** static terminal output.
- **Level 2:** deterministic computation.
- **Level 3:** input-dependent behavior.
- **Level 4:** deterministic file generation.
- **Level 5:** BVi binary-first raster construction.

- **Level 6:** BV2 procedural runtime rendering.
- **Level 7:** bounded interactive exact-binary artifact.

Each level contains multiple task families to reduce one-artifact memorization effects.

6.5 Development and Held-Out Cohorts

Prompt/verifier refinement is allowed only on a development cohort. Before evaluation, prompt wording, scoring, verifier implementation, retry policy, and promotion rules are frozen. Tasks used to develop the protocol cannot later be counted as held-out confirmation of the same change.

6.6 Primary Metrics

Every run preserves:

- artifact materialization success;
- structural validity;
- runtime behavioral success;
- exact reconstruction success;
- hard-invariant compliance;
- provenance classification accuracy;
- visual-class accuracy where applicable;
- final PASS/FAIL/INCONCLUSIVE classification accuracy;
- repair count;
- failure-preservation completeness;
- measurable work cost.

6.7 Constraint-Compliant Exact-Binary Success (CCEBS)

CCEBS is a prospective composite endpoint introduced by this paper; it is not a metric named in [S1].

A run counts as full success only when all applicable conditions are satisfied:

structural validity AND required runtime behavior AND complete frozen representation AND exact reconstruction identity AND all hard invariants PASS AND correct claim classification AND required provenance checks AND required post-Build checks.

For BV1/BV2 tasks, visual-class validity is also required.

CCEBS intentionally prevents a technically functioning but methodologically invalid artifact from being counted as a complete success.

6.8 Failure Taxonomy

The **Fi-Fro taxonomy** is a **paper-defined scoring taxonomy** for the prospective benchmark rather than a numbered taxonomy copied from [S1].

The benchmark uses ten failure classes:

- **Fi:** representation failure;
- **F2:** executable-structure failure;
- **F3:** instruction failure;
- **F4:** control-flow/offset failure;
- **F5:** ABI/environment failure;
- **F6:** behavioral failure;
- **F7:** provenance failure;
- **F8:** artifact-identity failure;
- **F9:** claim inflation;
- **Fro:** evidence-capture failure.

Multiple codes may apply to one run.

6.9 Retry and Repair Policy

A recommended primary design is one root attempt plus up to two evidence-triggered repair cycles per condition. Manual evaluator repairs are not permitted. A repair counts when an acceptance-relevant artifact or invariant changes; cosmetic reformulations do not.

6.10 Evaluator Independence

Verification prefers deterministic or externally observable checks:

direct byte comparison → cryptographic hash → format parser → known-output oracle → contained runtime behavior → process-path evidence → independent audit → model self-assessment.

The generating model's confidence cannot override stronger evidence.

6.11 Preregistered Ablations [See Figure 11.](#)

FIGURE 11

Prospective PARS-FEBP evaluation architecture

The causal comparison holds model, task, environment, and resource envelope constant while changing the prompting/protocol condition.

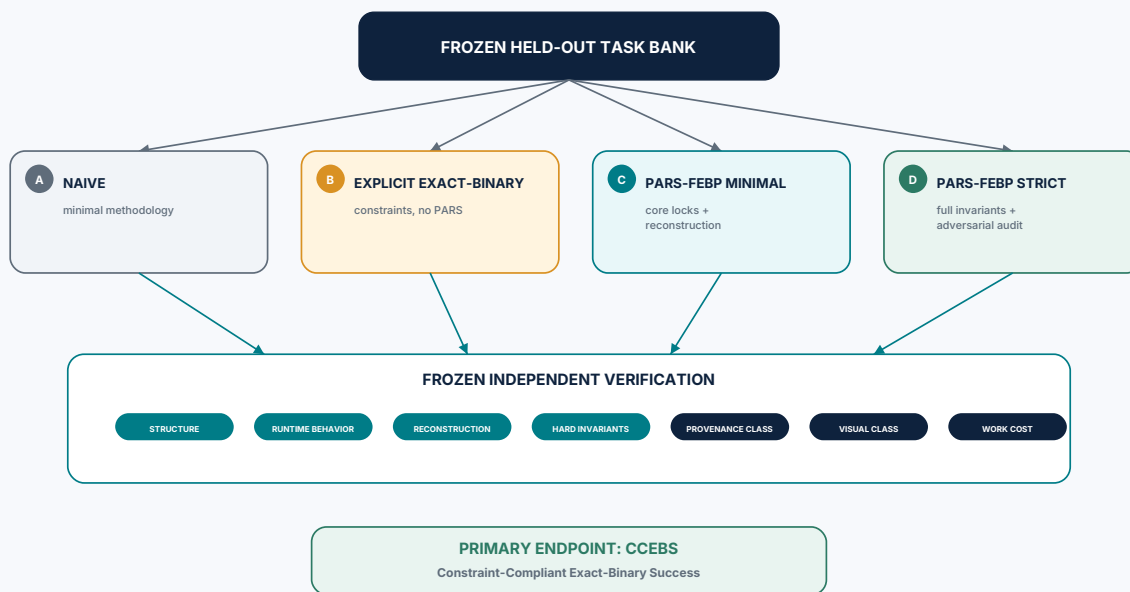


Figure 11. Prospective PARS-FEBP evaluation architecture. Model, task, environment, and resource envelope are held constant while prompting/protocol conditions change; CCEBS is the preregistered composite endpoint.

The protocol removes one component at a time and freezes its predicted failure before scoring:

- **ABL-1:** remove Representation Lock -> more source/assembly substitution.
- **ABL-2:** remove Negative-Path Lock -> more conventional-toolchain use.
- **ABL-3:** remove Reconstruction Lock -> more unverifiable or mismatched artifact identity.
- **ABL-4:** remove Provenance Lock -> more unsupported BP2 promotion.
- **ABL-5:** remove Adversarial Lock -> more hidden contamination survives.
- **ABL-6:** remove Final Invariant Gate -> more final candidates violate previously recognized constraints.
- **ABL-7:** remove Post-Build Replay -> more materialization defects survive delivery.
- **ABL-8:** remove Reconstruction stage -> more downstream failures after local repairs.

6.12 Severe-Regression Vetoes

A condition cannot be promoted merely because average output success increases if it introduces severe methodological failures such as artifact substitution, unsupported BP2 claims, hard-invariant bypass, false reconstruction claims, or BVo-to-BV2 inflation.

6.13 Interpretation Rules

Possible benchmark conclusions include:

- PARS improves CCEBS under the tested cohort;
- PARS improves verification/claim discipline without improving raw artifact generation;
- PARS improves reliability at significant work cost;
- no meaningful difference;
- PARS performs worse.

All outcomes are publishable evidence under the preregistered rules. **No comparative performance result is claimed by this chapter.**

7. Historical Evidence and Case Studies [Contents](#)

7.1 Evidence-Selection Method

Historical cases are selected by evidence diversity rather than spectacle. Eight editorial dimensions are considered:

- complete byte representation;
- artifact hash/byte count;
- independent reconstruction;
- execution;
- runtime-output/behavior verification;
- generation-path provenance;
- proof video;
- preserved failures/downgrades.

The cases are explicitly classified as:

- **Type A - Method Demonstration;**
- **Type B - Prospective Benchmark;**
- **Type C - Historical/Development Evidence.**

All cases in this chapter are Type C unless later regenerated under the prospective protocol. In two cases, exact historical prompt text was not recovered; the paper does not invent it. Instead it cites the retained byte streams, receipts, and audit records.

7.2 Worked Example 0 - 907-Byte Starship ELF

The compact Starship artifact provides a manageable end-to-end example of the proof chain.

The authoritative artifact is `PARS_FREEHAND_STARSHIP_X64.elf`, a 907-byte Linux x86-64 ELF with one `PT_LOAD`, no section table, and no dynamic loader. Its SHA-256 is:

1df35f8ffc64fac0f9e5b9de2c7ad4b763f16d4b40ab7e4e9987b5065030a354 [\[H1\]](#).

When executed with standard output redirected to a file, the binary emits a deterministic P6 PPM image at 256 x 256 RGB. The runtime artifact contains 196,623 bytes and has SHA-256:

97565975c7b49eed6278f391aeb96803d3cc03d79d59b1747692e26d895569c [\[H1\]](#).

The retained exact byte stream begins with the literal ELF magic and complete executable bytes. The historical provenance receipt states that executable fields, x86-64 instructions, relative branch displacements, constants, PPM header, and the final 907-byte stream were specified as literal bytes; it records no C/C++/Rust source, assembly source, compiler, assembler, linker, object file, code generator, or linked runtime library in the executable-generation semantics [\[H1\]](#).

After the representation was fixed, a shell read loop plus `printf '%b'` wrote the already-determined `\xHH` tokens to the ELF. The receipt explicitly distinguishes this from instruction generation or symbol resolution [\[H1\]](#).

The same frozen stream was materialized again into a reconstruction. Original and reconstructed files share the same SHA-256 and `cmp` reports exact identity [\[H1\]](#). See [Figure 12](#).

FIGURE 12

907-byte Starship evidence flow

A compact worked example connects the retained byte representation to exact reconstruction and deterministic raster output.

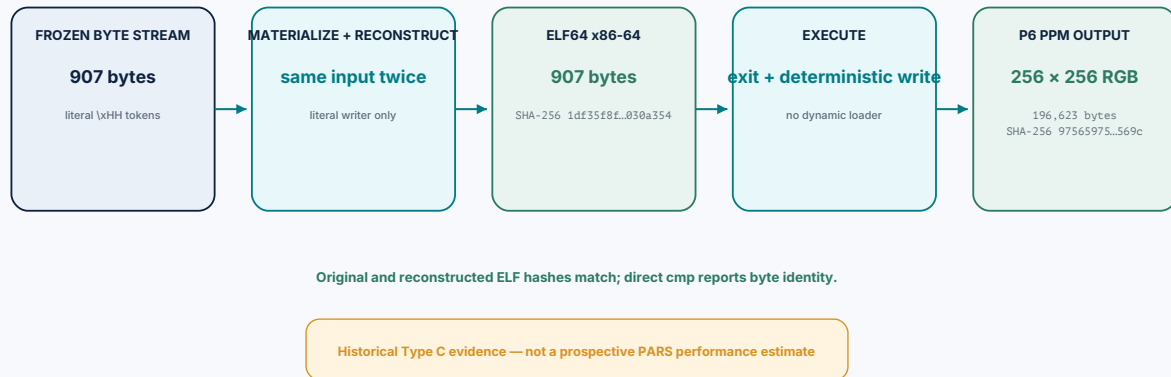


Figure 12. Historical 907-byte Starship evidence flow from retained byte representation through exact reconstruction and deterministic 256×256 PPM output [H1].

The compact case teaches the core sequence:

frozen representation → literal materialization → artifact hash → independent reconstruction → byte identity → execution → deterministic runtime output.

Its historical classification is PASS under its recorded contract. This paper reports the retained receipt’s provenance claim rather than treating the historical case as a newly conducted independent provenance audit. Because the original prompt was not recovered and the case predates the prospective benchmark, it is not used to estimate PARS performance.

7.3 Case Study 1 - LHC Collider v2.1

CASE FILE 01 / LHC COLLIDER v2.1			
ARTIFACT 2,048 BYTES	FORMAT ELF64 x86-64	CLAIM BP2 / BV2	STATUS PASS — HISTORICAL

The LHC-inspired collider experiment is the strongest methodological historical case because successful execution alone did not produce acceptance.

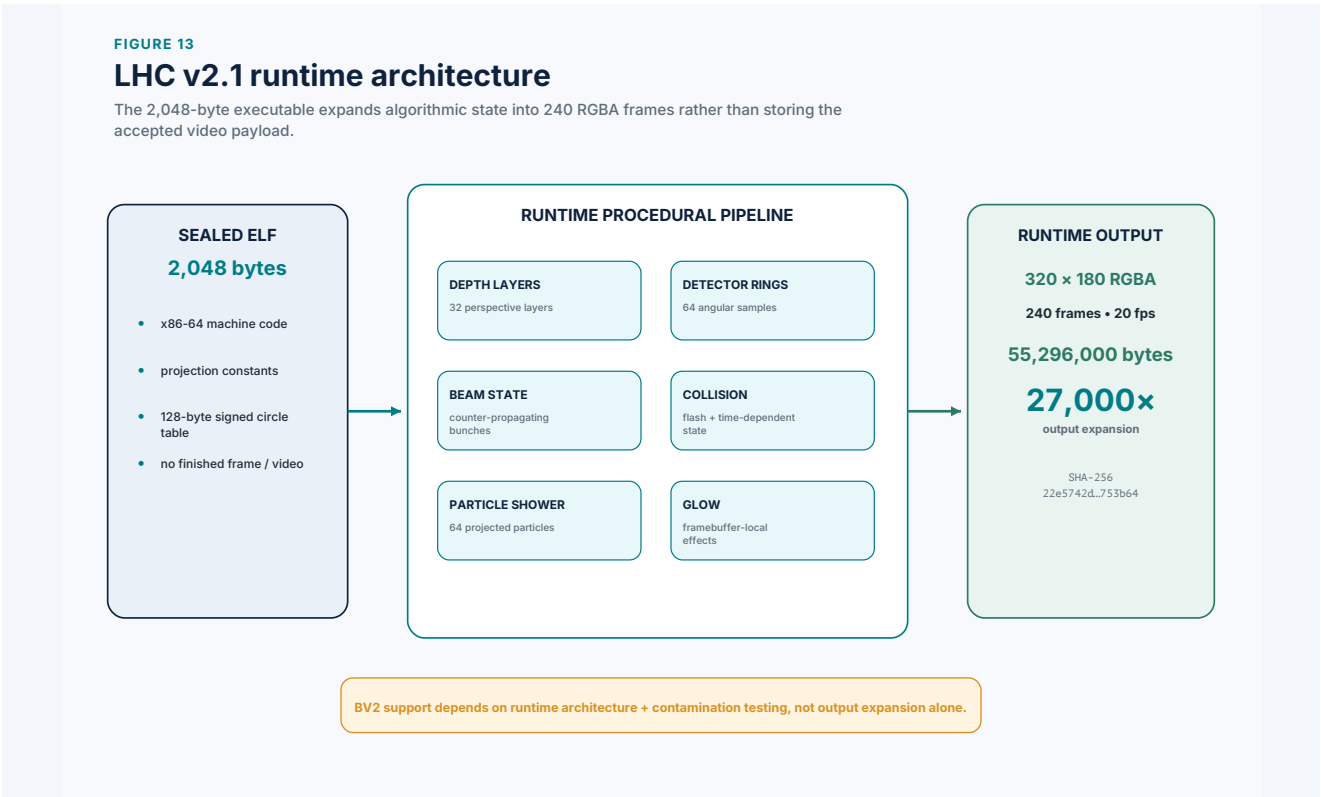
The accepted artifact, `PARS_LHC_COLLIDER_V2_1_FREEHAND_FINAL_X64.elf`, is a 2,048-byte static Linux x86-64 ELF with one `PT_LOAD`, no dynamic loader, and no section table. Its SHA-256 is:

04d3da29c36a584baaf9ddd0f8a7e664de1a4da96e2e4292d9cdb4ebcc507b1c [\[H2\]](#).

The complete exact representation contains 64 lines encoding all 2,048 executable bytes. At runtime the ELF computes a 320 x 180 RGBA framebuffer for 240 frames at 20 fps. The resulting raw output contains 55,296,000 bytes with SHA-256:

22e5742d9e9690deb42def9f292e2a2b5697a382eebda86b668c2a373f753b64 [\[H2\]](#).

The output is exactly 27,000 times larger than the sealed executable. The runtime computes perspective depth layers, detector rings, counter-propagating particle bunches, a collision flash, projected particle shower, and glow effects. The artifact contains a compact 128-byte signed circle-coordinate table but no finished frame, texture, image, model, or video payload [\[H2\]](#). See [Figure 13](#).



- **V1:** executed but failed the requested visual gate because the result read as a flat event display rather than the intended 3D collider.
- **V2:** introduced the perspective tunnel but segfaulted because a late particle shower exceeded the vertical framebuffer bound.
- **V2.1:** corrected one literal byte at file offset 0x23c, changing 06 -> 07, which changed the shower Y-projection arithmetic shift from division by 64 to division by 128. V2.1 completed all 240 frames and passed the final evidence chain [H2]. See Figure 14.

FIGURE 14

LHC candidate lineage: V1 → V2 → V2.1

Failed candidates remain evidence. The accepted artifact is reached through explicit rejection, diagnosis, one-byte repair, and revalidation.

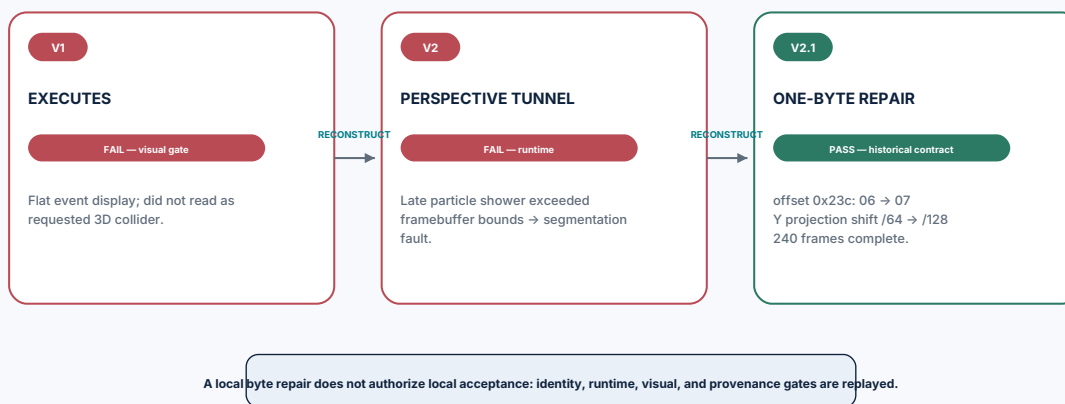


Figure 14. Preserved LHC candidate lineage. V1 fails the visual gate, V2 fails at runtime, and V2.1 applies the recorded one-byte repair at offset 0x23c before complete revalidation [H2].

This is a concrete example of PARS Reconstruction. A byte-level repair was followed by revalidation rather than treated as automatically safe.

Reconstruction and Runtime Round Trip

The visible frozen stream was independently materialized into `PARS_LHC_COLLIDER_V2_1_RECONSTRUCTED.elf`. Canonical and reconstructed binaries have identical SHA-256 values and `cmp` exits 0. The reconstructed artifact was executed independently and reproduced the exact same 55,296,000-byte runtime hash [H2]. See Figure 15.

FIGURE 15

Executable and runtime round-trip identity

The frozen representation must reproduce both the accepted executable and, for deterministic cases, the same observed runtime output.

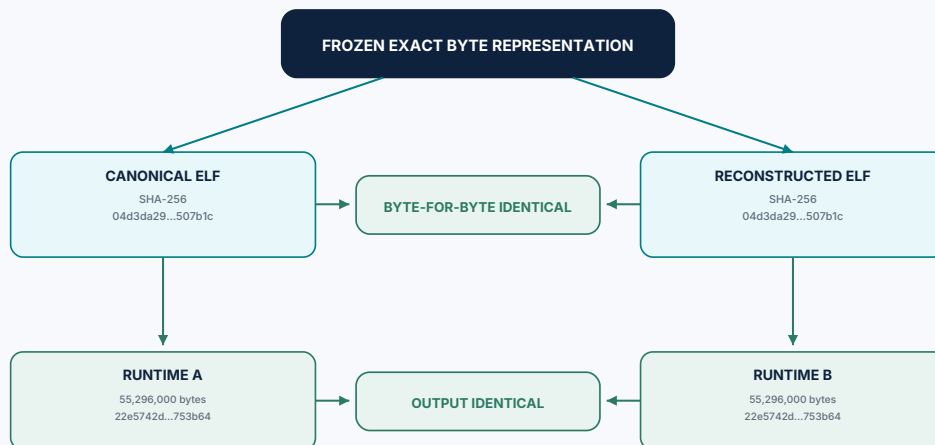


Figure 15. Executable and runtime round-trip identity for LHC v2.1: the frozen representation reconstructs the same ELF and the reconstructed executable reproduces the same deterministic runtime hash [H2].

The historical receipt classifies the artifact **BP2 / BV2** under its recorded provenance contract. This paper reports that historical classification; it does not retrospectively re-certify the complete upstream session as though it were a new prospective audit. The receipt explicitly distinguishes the sealed reconstruction path from development/QA helpers. Python was used during development to hold already-selected literal blocks, write candidates, export the final stream, hash files, and prepare media; the receipt states it did not calculate the final opcodes or branch displacements [H2].

The case therefore demonstrates not only a working procedural renderer but a rejection/repair/reconstruction lineage under explicit claim boundaries.

7.4 Case Study 2 - WASM Hyperlattice

The WASM Hyperlattice demonstrates that the method is not limited to ELF.

The authoritative artifact, `PARS_WASM_HYPERLATTICE_EXACT.wasm`, is a 9,433-byte WebAssembly v1 module with SHA-256: `c1d31cb6f7f052c5053697a1acaa9cb43f2b8ee2959d1df0274f5303bd37e7c3` [H3].

The receipt states that the module was serialized directly as exact module bytes without a WebAssembly compiler, assembler, linker, WAT translator, or source-language compiler. The module has zero imports and exports memory plus reset, update, render, width/height, framebuffer, and mutable mode state [H3].

The module owns its animation state, input transitions, camera/phase drift, warp-state control, procedural lattice calculations, color synthesis, and RGBA framebuffer generation. Its native framebuffer is 384 x 216 RGBA (331,776 bytes per frame). A runtime perturbation changed 82,944 of 82,944 pixels, providing discriminating evidence against a static framebuffer interpretation [H3].

Exact regeneration reproduced the same 9,433-byte artifact and SHA-256; direct cmp, WebAssembly validation, and regenerated execution all passed. The proof video visibly includes every byte through offset 0x24d8, followed by byte counts, hashes, identity, validation, exported API, execution, motion, and performance evidence [H3].

The host role is disclosed separately: Node.js instantiates the exact module and copies its exported framebuffer for capture because the sandbox browser policy prevents the ordinary local-browser route. The `.wasm` remains the authoritative generated artifact [H3].

This case supports cross-format applicability under the tested conditions without claiming arbitrary binary-format generality.

7.5 Case Study 3 - Binary Brains Gate 9

Binary Brains shifts the problem from media generation to stateful causal computation.

The historical Gate 9 artifact, `BINARY_BRAINS_v0.9.0_FROZEN_RECON.elf`, contains 34,175 bytes with SHA-256:

a8bd61cc7413bcb0392efa496a1e68f41320c197ce3ef850fd69428f576308cf [H4].

The retained record reports 583/583 independent audit checks passed, baseline exit code 0, counterfactual exit code 0, invalid-checksum exit code 2, unchanged pre/post executable hash, and byte-for-byte frozen-hex reconstruction [H4].

Counterfactual Test

In the baseline world, one Brain A payload byte is 36. The counterfactual changes exactly one bit, 36 -> 37, while Brain B's corresponding input remains unchanged. The resulting IDEA history diverges immediately at generation 0: baseline begins with 2D, counterfactual begins with D3, and later recursive state diverges accordingly [H4].

This provides a stronger causal test than merely observing dynamic animation.

Multiprocess Evidence

The accepted sampled run records observer PID 10574, Brain A PID 10576, and Brain B PID 10577. Host process inspection independently showed both child processes alive simultaneously with the expected parent [H4]. See Figure 16.

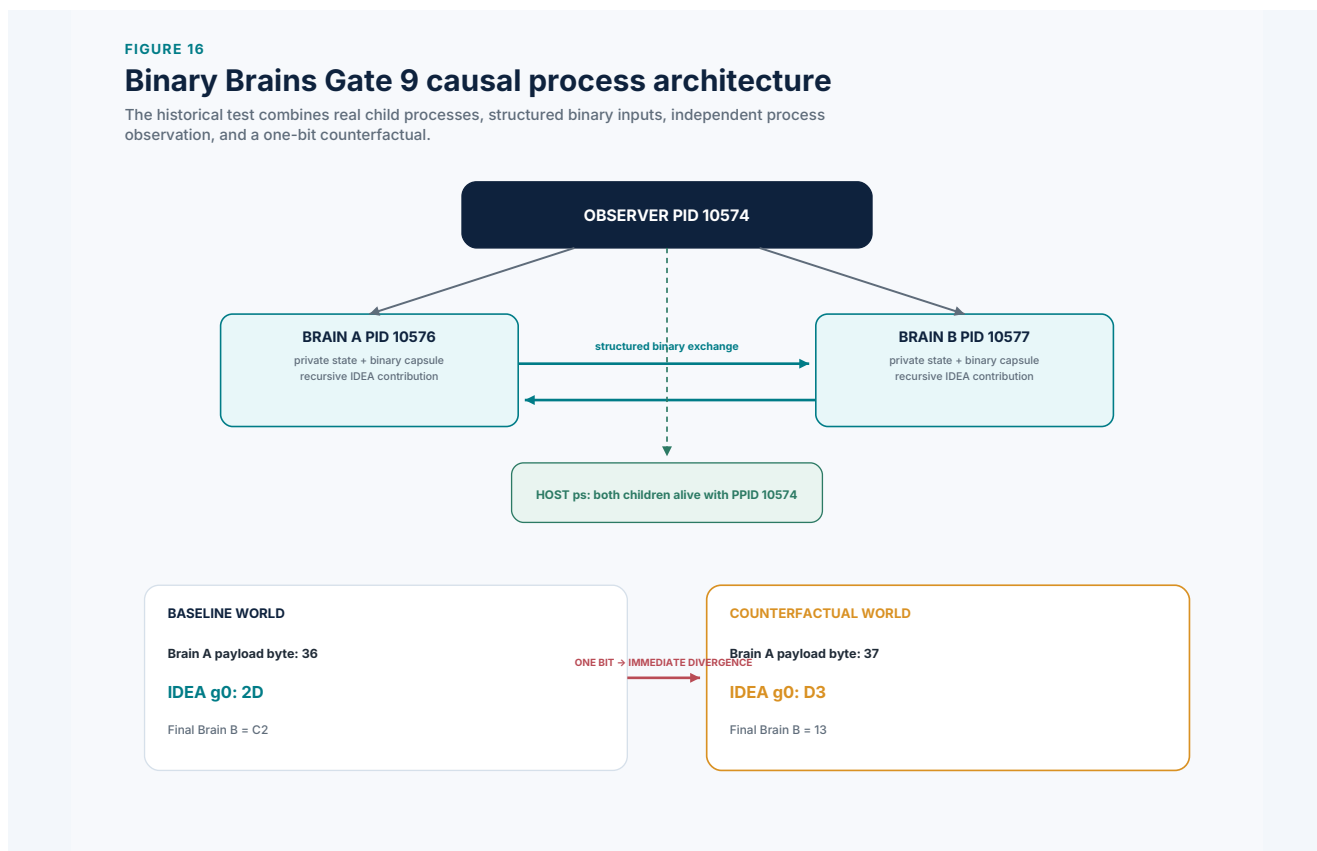


Figure 16. Binary Brains Gate 9 causal process architecture. The retained record combines host-observed child processes with a one-bit Brain A input perturbation that changes the IDEA sequence from generation 0 [H4].

The Gate 9 H.264 proof is 1920 x 1080, 60 fps, yuv420p, 18.10 seconds, and was recorded entirely inside the assistant sandbox [H4].

Provenance Nuance

The project rules exclude compiler, assembler, linker, object builds, prebuilt executable templates, and image generation. The same continuity record also discloses pre-freeze Python use for literal byte concatenation and address/displacement arithmetic [H4].

This distinction is important: **no conventional compiler/assembler/linker** and **no computational construction helper of any kind** are different provenance predicates. The paper therefore preserves the narrower historical claim rather than silently upgrading it.

7.6 Case Study 4 - PARS-VM/4

PARS-VM/4 demonstrates architectural depth within a compact artifact.

The frozen executable, `PARS_VM4_EXACT_BYTE_STREAM.elf`, is a 5,392-byte Linux x86-64 ELF with SHA-256:

5a002c60af45a59c392bffbcbbc26d82a6674f0dd7627ee90bbd25089d1908e1 [H5].

The runtime contains a purpose-built virtual computer with:

- 64 KiB mutable guest RAM;
- 16 signed 32-bit guest registers;
- guest program counter;
- a richer fixed four-byte guest ISA;
- arithmetic, LOAD/STORE, branches, SYS traps, and HALT;
- a 320 x 200 x 32-bit framebuffer;
- raw X11 display transport;
- a hardware-style virtual GPU filled-triangle rasterizer;
- guest-side fixed-point 3D rotation and projection;
- interactive input [H5]. See Figure 17.

FIGURE 17

PARS-VM/4 virtual-computer architecture

A 5,392-byte x86-64 host implements a custom guest CPU/ISA, RAM, devices, guest-side 3D math, and a software VGPU.

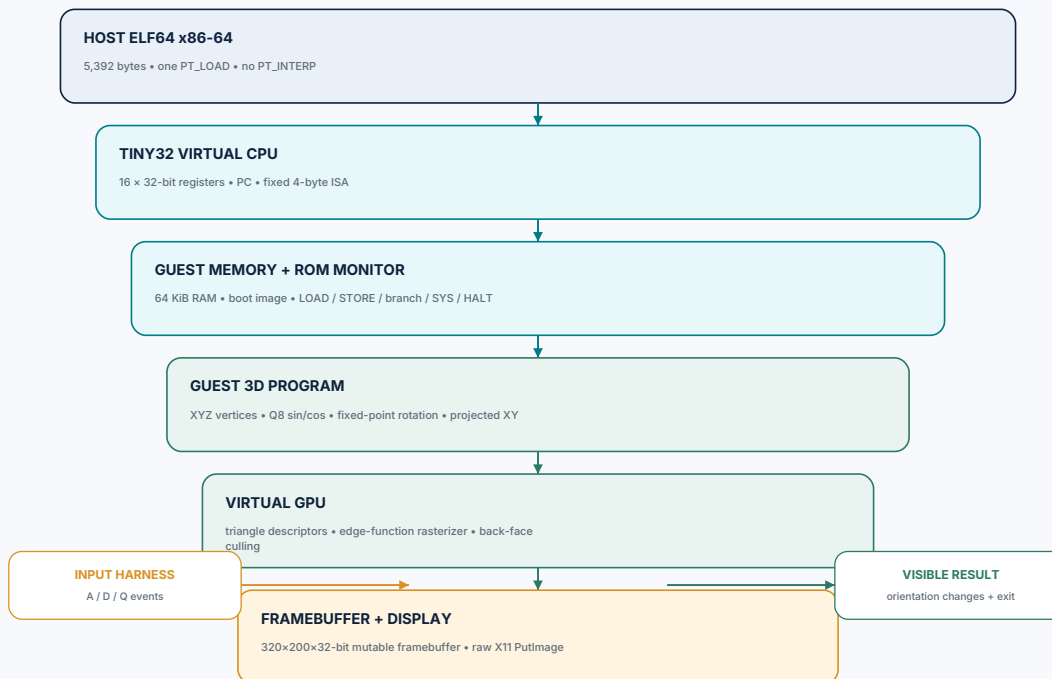


Figure 17. PARS-VM/4 virtual-computer architecture: a 5,392-byte x86-64 host implements a custom virtual CPU/ISA, guest memory and monitor, guest-side 3D computation, software VGPU, framebuffer, and raw X11 display transport [H5].

The host layer does not contain pre-rendered frames or pre-projected cube coordinates. The guest program computes projected vertices after launch and stores them in guest RAM. The virtual GPU then performs edge-function triangle rasterization and presents the framebuffer through raw X11 protocol bytes [H5].

A clean Xvfb runtime was driven by an external input harness that allowed autonomous rotation, injected A/left and D/right events, then Q. The frozen executable changed rendered orientation and exited after Q [H5].

Independent reconstruction from the frozen hexadecimal stream produced the identical SHA-256 and cmp passed [H5].

The receipt excludes C/C++/Rust source, assembly, compiler, assembler, linker, object file, executable template, graphics framework, or prebuilt guest emulator from the committed executable-generation path, while also disclosing development-time Python for byte-array and address/displacement arithmetic [H5].

The scope is deliberately limited: VM/4 is not an x86 PC emulator, general-purpose desktop OS, hardware GPU, or modern production 3D engine. It demonstrates the challenge at the scale of a purpose-built virtual computer.

7.7 Case Study 5 - House of the Rising Sun

House of the Rising Sun stresses byte-count, runtime, and proof-presentation scale.

The final frozen artifact, `RISING_SUN_HOUSE_FINAL_EXACT_BINARY_ELF`, is a 57,344-byte ELF64 x86-64 executable with one PT_LOAD, no section table, and no dynamic interpreter. SHA-256:

4d4fe5c98507c4a3bb4f852323bb3ed7dc4ce0612e2873f7b35a2591a4d1ff10 [H6].

The canonical emitter contains the complete executable as explicit three-digit octal byte escapes. Re-emission produces a byte-identical file with the same hash [H6].

The committed native run lasts approximately 211.5 seconds, exits 0, and retains the same pre/post executable hash. The executable generates a 210-second 48-kHz stereo signed-16-bit PCM WAV containing 40,320,044 bytes with SHA-256:

90be877fad9d99966b61a93c008a92bbb841a8e495c99c1e91b723f9b6379a3c [H6].

A capture-attached run again preserves the executable identity. The proof showcase is 280.021333 seconds and includes 112 complete byte pages, each 32 rows x 16 bytes, covering all 57,344 executable bytes with no omitted regions. It then presents committed hash/state verification, the full 210-second runtime, and post-run identity [H6].

The arithmetic is directly auditable:

112 pages x 32 rows/page x 16 bytes/row = 57,344 bytes.

This case shows that even when generation succeeds, **proof burden itself becomes a scaling problem.**

7.8 Case Study 6 - When a Working Binary Still Cannot Pass the Strict Contract

CASE FILE 06 / BVIS-001 INDEPENDENT AUDIT

ARTIFACT
228 BYTES

SUPPORTED
BP1

BLOCKER
UPSTREAM PROVENANCE

STATUS
INCONCLUSIVE

A separate BVIS-001 audit artifact, titled `Independent Audit`, provides the clearest historical illustration that the verification layer can return `INCONCLUSIVE` despite extensive technical success.

The package's own report said `PASS`. Independent replay confirmed nearly all of the technically impressive evidence:

- complete 228-byte executable representation;
- exact byte-stream -> ELF reconstruction;
- ELF -> byte-stream round trip;
- execution exit 0;
- exact runtime/reference identity;
- no embedded finished reference/pixel payload;
- successful rejection of a deliberately contaminated candidate;
- complete H.264 proof reel;
- procedural visual behavior;
- BP1 support [\[H7\]](#).

Yet the independent verdict was:

INCONCLUSIVE under the frozen strict-binary-first contract [\[H7\]](#).

The blocking gap was upstream provenance. The package proved that an already-frozen `byte_stream.hex` was literally converted into the ELF without compiler/assembler/linker participation. It did **not** contain evidence establishing how `byte_stream.hex` itself had originally been created [\[H7\]](#). See [Figure 19](#).

FIGURE 19

BVIS-001: technical success → provenance gap → INCONCLUSIVE

The strict contract fails at the missing upstream evidence boundary even though reconstruction, execution, output identity, and contamination checks succeed.

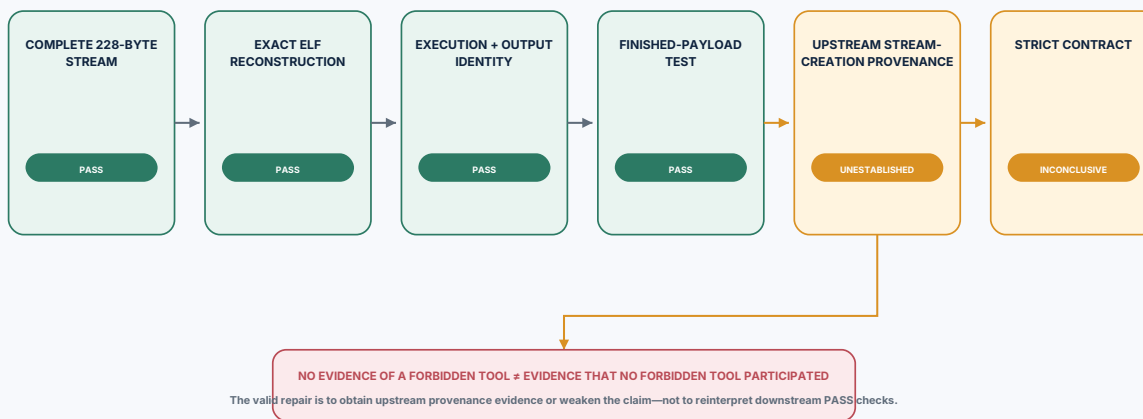


Figure 19. BVIS-001 evidence path. Reconstruction, execution, output identity, and contamination checks pass, but upstream byte-stream creation provenance is unestablished; the frozen strict contract therefore remains INCONCLUSIVE [H7].

The corrected claim boundary was therefore:

Supported: BP1 exact-byte materialization, procedural/reference-conditioned visual behavior, exact runtime/reference identity, no finished payload, contamination rejection, complete proof reel.

Not established: BP2 and no-toolchain involvement before the byte-stream freeze [H7].

Under the paper’s classification rules, this illustrates the intended behavior of the verification layer: technical success does not force a provenance PASS. A technically successful artifact can remain INCONCLUSIVE under a stronger experimental contract.

7.9 Cross-Case Synthesis
 See Figure 18.



Figure 18. Historical complexity/evidence ladder. The selected Type C cases stress different dimensions—serialized size, format, state, causal behavior, runtime scale, and proof burden—rather than constituting a statistical success sample. See Table 2.

Table 2. Historical case portfolio, artifact sizes, and primary methodological contribution.

Case	Bytes	Primary contribution
Starship	907	Minimal end-to-end exact-byte workflow
LHC v2.1	2,048	Failure preservation, one-byte repair, BV2, runtime replay
VM/4	5,392	Custom CPU/ISA/RAM/VGPU and guest-side 3D
WASM Hyperlattice	9,433	Non-ELF direct binary format
Binary Brains Gate 9	34,175	Multiprocessing, structured binary communication, causal counterfactual
Rising Sun	57,344	Long runtime and complete-proof scaling
BVIS-001	228	Separate audit classified the strict contract INCONCLUSIVE on a provenance gap

Taken together, the historical evidence supports a narrow statement: reconstructible exact-byte workflows have been applied to multiple machine-consumable artifact types and runtime behaviors, and the evidence base includes rejected candidates, byte repairs, provenance distinctions, and at least one technically successful artifact downgraded because the stronger generation-path claim was not established.

The cases do **not** establish a statistically measured PARS advantage over simpler prompting or universal exact-binary capability.

8. Discussion [Contents](#)

8.1 What PARS Appears to Contribute

The strongest architectural observation is that PARS changes exact-binary generation from an output-generation problem into a **contract-preservation and evidence-classification problem**.

A conventional success criterion might be simply “does the binary run?” PARS adds independent questions: were these the published bytes, can they reconstruct the artifact, did the reconstructed artifact run, did Build change identity, were forbidden paths excluded, is the evidence BP1 or BP2, is a visual BV0 or BV2, and did any active invariant fail?

Historical evidence cannot yet quantify how much this architecture improves success. It does show why acceptance discipline matters.

8.2 Constraint Preservation Versus Candidate Generation

Some of the most consequential PARS behavior occurs after a plausible candidate already exists. The LHC lineage illustrates this: V1 ran but failed the visual contract; V2 had the intended perspective behavior but failed at runtime; only V2.1 survived the full evidence chain [H2].

Thus, PARS may contribute as much by preventing premature acceptance as by helping generate bytes. This remains a prospective benchmark question rather than a settled performance claim.

8.3 Failure Preservation as Evidence

Ordinary development often treats failed candidates as disposable. PARS treats them as evidence. A preserved failure provides information about which representation, assumption, or invariant was wrong and whether the repair altered downstream state.

The LHC 0x23c repair is unusually concrete: one byte changed projection scale and eliminated a framebuffer-bound failure [H2]. Because the prior candidate remains visible, the final success has a documented causal repair story rather than a sanitized narrative.

8.4 Correctness and Provenance Are Orthogonal

BVIS-001 makes this distinction unavoidable. The artifact reconstructed, executed, reproduced its output, passed payload tests, and still remained INCONCLUSIVE under the strict upstream-provenance contract [H7].

The key principle is:

Artifact correctness and creation provenance are related but non-equivalent evidence dimensions.

8.5 Why “Freehand” Requires an Operational Definition

Historical cases expose ambiguity in the word *freehand*. Some exclude compiler, assembler, linker, object files, and prebuilt templates while still using arithmetic or byte-array helpers before freeze. Binary Brains and VM/4 disclose exactly such helper use [H4][H5].

Therefore, “no conventional compiler/assembler/linker” and “no computational helper of any kind” must not be treated as synonyms. The experimental contract must state which provenance predicate is actually required.

8.6 Materialization as an Information Boundary

The appropriate distinction is functional rather than tool-name-based. A shell or Python program may be a literal materializer if it only decodes already-complete bytes, or a generator if it computes opcodes, labels, addresses, or executable structure.

The question is not “was Python used?” but “what information did the step contribute?”

8.7 Exact-Byte Generation Is Not a Replacement for Conventional Toolchains

These experiments deliberately remove conveniences provided by compilers, assemblers, and linkers because doing so exposes a difficult reasoning and provenance problem. That does not imply direct byte authoring is a superior engineering practice.

Conventional toolchains provide optimization, portability, symbolic debugging, maintainability, relocation handling, and reproducible build systems. The research value of direct byte generation is precisely that it removes these layers and makes the final serialized representation part of the reasoning challenge.

8.8 Exact Bytes Do Not Reveal a Unique Internal Cognitive Mechanism

A model producing valid executable bytes does not prove that its internal cognition is equivalent to a compiler or assembler. Success may arise from learned binary-format knowledge, learned instruction patterns, arithmetic reasoning, context conditioning, iterative correction, external inspection, or PARS constraint management.

The experiments observe artifact behavior and evidence, not a unique hidden mechanism.

8.9 Exact-Binary Repair Does Not Establish Model-Weight Learning

Repeated repair within a conversation may demonstrate RL0 in-task adaptation. Reusable verified strategies may support RL1. A promoted search-controller mutation may support RL2. None of these establishes RL3 model-parameter learning without an actual training mechanism [S1].

8.10 Scaling the Artifact Versus Scaling the Proof

The selected artifacts span 907 to 57,344 bytes, but byte count is only one complexity dimension. VM/4 is smaller than the WASM Hyperlattice while containing a virtual CPU, RAM, VGPU, and guest-side 3D pipeline. Scale should therefore be considered across serialized size, state complexity, runtime duration, output volume, interaction, and proof burden.

Rising Sun also demonstrates that human-visible complete-byte proof does not scale indefinitely. At 57,344 bytes, 112 proof pages are already required [H6]. At multi-megabyte scale, cryptographic and machine-verifiable completeness may remain practical while literal human inspection of every byte does not.

9. Limitations, Alternative Explanations, and Threats to Validity [Contents](#)

9.1 Historical Selection Bias

The historical portfolio was assembled after many development experiments had already occurred. It is therefore subject to selection and survivorship bias. The selected cases illustrate diverse stresses; they cannot estimate overall success probability, average repair count, or expected work cost.

Only prospective testing can support those estimates.

9.2 Incomplete Historical Prompt Archives

For some early cases, exact byte streams, hashes, receipts, and videos were retained while the exact original prompt text was not recovered. Those cases can support artifact and provenance claims under their retained records but cannot establish the causal effect of particular prompt wording.

This is why they remain Type C historical evidence.

9.3 Generator/Verifier Correlation

Some historical verification was authored inside the same broader development environment that produced the candidates. This creates the possibility of correlated mistakes. Deterministic hashes, byte comparisons, parsers, known-output oracles, and evaluator/replay separation reduce but do not eliminate this risk.

Some retained records use the word *independent* for an audit or replay. Unless evaluator identity and separation are separately established, this paper does not treat that label alone as proof of external third-party validation. PARS's anti-self-confirmation rule explicitly prefers deterministic, held-out, independently measured, or otherwise separated evidence [\[S1\]](#).

9.4 Limits of Provenance Evidence

A package can establish published bytes → exact artifact → exact runtime without proving every upstream event that produced the published bytes. BVIS-001 demonstrates this directly [\[H7\]](#).

Provenance should therefore be described as supported by the observed/audited environment and procedure, not as absolute omniscient proof.

9.5 Runtime Environment Dependence

Not every artifact is independent of host infrastructure. WebAssembly requires a host; VM/4 requires an X11 display; some graphical artifacts require GLX/OpenGL or system libraries. Runtime dependencies must be disclosed separately from generation dependencies.

“Single binary” therefore does not mean “independent of all operating-system or hardware services.”

9.6 Capture and Presentation Distortion

Recording, scaling, frame duplication, muxing, and stitching can alter how runtime evidence is presented. The evidence artifact must therefore distinguish raw runtime observation from presentation encoding. Rising Sun explicitly distinguishes a lower-cadence uninterrupted evidence capture from 60-fps delivery [\[H6\]](#).

9.7 Subjective Visual Acceptance

Hashes can prove exact identity but not aesthetic quality. A scene may be deterministic yet unattractive. A model may be objectively 3D yet fail the requested artistic target. Visual criteria should therefore separate objective geometry/runtime properties from subjective aesthetic judgments.

9.8 Toolchain-Marker Absence Is Not BP2

No GCC, Clang, .comment, or linker marker may be useful supporting evidence, but missing metadata is not a creation-path audit. PARS explicitly rejects this inference [S1].

9.9 Model and Configuration Dependence

Observed success is conditional on the model, context, tools, and environment used. A different model or tool configuration may perform differently. Historical artifacts cannot be treated as model-independent properties of PARS.

9.10 Human-Readable Proof Does Not Scale Indefinitely

Complete visible byte pages are compelling for small and medium artifacts. For large binaries, literal visual display becomes unwieldy. Machine-verifiable completeness and human-readable exhaustive display are distinct goals.

9.11 Learned Binary Knowledge as an Alternative Explanation

The base model may already contain extensive learned knowledge of ELF, WebAssembly, x86-64, syscalls, and rendering algorithms. PARS may organize and constrain that knowledge rather than create new underlying capability.

9.12 Prompt Specificity as an Alternative Explanation

A sufficiently detailed ordinary prompt may capture some of the same benefits as PARS. This is why the prospective benchmark includes an explicit exact-binary prompt condition separate from full PARS-FEBP.

9.13 Iterative Debugging as an Alternative Explanation

A model allowed to inspect failures and repair candidates may improve even without PARS. PARS may add value principally through structured failure preservation, representation changes, Reconstruction, and invariant replay. Ablations are needed to separate these effects.

9.14 Task-Family Selection Effects

Minimal ELF, procedural framebuffer programs, and compact virtual machines may be unusually amenable to direct byte construction because the format can be simplified, dynamic linking removed, and behavior kept deterministic. Success on these families should not be generalized automatically to large ordinary desktop applications.

10. Conclusion [Contents](#)

PARS treats exact-binary generation as a **contract + search + reconstruction + verification** problem rather than as a prompt followed by a working executable.

Three distinctions are central.

First, exact execution, exact byte identity, and generation provenance are separate claims. A binary can run without its published representation being proven. Its published representation can reconstruct exactly without establishing the upstream creation path. A procedural visual can look convincing while remaining an emitter if it embeds the finished output.

Second, the strongest value of the architecture may lie in constraint preservation and correct classification, not simply candidate generation. Historical evidence includes artifacts that were rejected despite successful execution, one-byte repairs followed by complete revalidation, cross-format exact reconstruction, stateful causal systems, and an independently audited technically successful case that remained **INCONCLUSIVE** because BP2 provenance was not established.

Third, historical demonstrations are not a substitute for prospective comparative evidence. The paper therefore defines a frozen benchmark capable of distinguishing naive prompting, explicit exact-binary prompting, PARS-FEBP Minimal, and PARS-FEBP Strict under shared model/environment conditions. Until such a benchmark is executed, no claim of prospective superiority is made.

The current evidence supports a narrower conclusion: PARS specifies an explicit architecture intended to support reproducible exact-binary contracting, candidate search and repair, failure preservation, exact-identity and runtime verification, separation of generation from materialization and evidence, and refusal of claims stronger than the available evidence.

10.1 Public Disclosure Boundary

This publication is intentionally limited to the architecture, methods, artifacts, and evaluation procedures documented herein. **Unpublished extensions, implementation directions, proprietary research directions, and non-public experimental plans are outside the scope of this paper.**

Appendix A - Terminology and Glossary [Contents](#)

PARS - Constraint-driven reasoning and verification architecture used in this paper.

Parse - Freeze the evaluable task, facts, assumptions, unknowns, constraints, and claim boundary.

Branch - Generate mechanism-distinct candidate paths including NULL/OTHER where applicable.

Transform - Change representation to expose structure or failure.

Perturb - Apply hostile or adversarial stress capable of falsifying leading claims.

Test - Obtain discriminating evidence under an explicit pass/fail criterion.

Reconstruct - Rebuild affected reasoning/artifact dependencies after material evidence changes.

Build - Produce/materialize the requested artifact after applicable gates pass.

Hard invariant - Acceptance-critical predicate that cannot be silently weakened.

Final Invariant Gate - Mandatory replay of all active hard invariants against the exact selected final candidate.

Post-Build Acceptance Replay - Rechecking invariants whose truth may change during materialization.

Generation plane - Where semantic decisions about artifact bytes occur.

Materialization plane - Where an already-complete representation is converted losslessly into bytes.

Evidence plane - Where hashes, parsers, comparison, execution, recording, and audits evaluate the artifact.

Literal materializer - A mechanism that decodes a frozen representation without adding executable semantics.

BP0/BP1/BP2/BP3 - Binary provenance classes defined in Chapter 3.

BV0/BV1/BV2/BV3 - Visual exact-binary claim classes defined in Chapter 3.

RL0/RL1/RL2/RL3 - Adaptation/learning claim classes defined in Chapter 2.

CCEBS - Constraint-Compliant Exact-Binary Success, the composite prospective benchmark endpoint.

Claim boundary - Strongest conclusion supported by current evidence, plus explicit stronger conclusions not established.

Appendix B - PARS-FEBP Prompt Protocol v1.0 [Contents](#)

B.1 Minimal Freehand Prompt

Activate PARS-Deep.

Create [TARGET] as a complete exact executable byte representation for [FORMAT / ARCHITECTURE / PLATFORM].

The executable bytes themselves are the authored artifact.

Do not use a compiler, assembler, linker, prebuilt executable, prebuilt object file, or source-language build step to generate the claimed binary.

Treat these as hard constraints.

Freeze the complete byte representation before execution.

Materialize the artifact literally from that representation, verify byte count and SHA-256, reconstruct it independently, confirm byte-for-byte identity, and execute the reconstructed artifact.

Report only the strongest provenance claim supported by the evidence.

Return PASS, FAIL, or INCONCLUSIVE.

B.2 Strict Research Prompt

ACTIVATE PARS-DEEP.

Apply PARS to the actual artifact-generation task.

TARGET

Create [EXACT TARGET].

Platform: [PLATFORM]

Format: [FORMAT]

Architecture: [ARCHITECTURE]

Required observable behavior:

[BEHAVIOR]

REPRESENTATION CONTRACT

The complete executable byte representation is the artifact being authored. Source code, assembly source, pseudocode, or build instructions do not satisfy this experiment.

FORBIDDEN GENERATION PATHS

The following are hard constraints:

- no compiler;
- no assembler;
- no linker;
- no source-language build path generating the executable;
- no prebuilt executable;
- no prebuilt object file;
- no hidden binary substitution;
- no undisclosed binary-generation mechanism.

Do not weaken these constraints to obtain a working result.

MATERIALIZATION CONTRACT

A literal representation-to-byte writer may materialize an already-complete

frozen byte sequence. It must perform no semantic instruction translation, code generation, symbol resolution, assembly, linking, or binary substitution.

PROVENANCE CONTRACT

Successful execution is not sufficient provenance evidence. Do not infer absence of conventional tooling merely from missing metadata or from the binary appearing hand-constructed.

BYTE-IDENTITY CONTRACT

Freeze the complete accepted byte representation. Record byte count, SHA-256, format, architecture, and entry point when relevant. Reconstruct the binary exclusively from the frozen representation and require byte-for-byte equality.

EXECUTION CONTRACT

Execute the reconstructed binary and record exit status and observable behavior.

ADVERSARIAL CONTRACT

Perform at least one test capable of falsifying the strongest generation or provenance claim. Preserve failed candidates and meaningful failures.

FINAL INVARIANT GATE

Before acceptance, replay every active hard invariant on the exact candidate that will be built. FAIL blocks Build. UNVERIFIABLE does not count as PASS.

POST-BUILD GATE

After materialization, replay every invariant that Build could have changed.

CLAIM BOUNDARY

Return only the strongest evidence class actually established.

Final result must include:

Artifact identity

Byte count

SHA-256

Format

Architecture

Reconstruction identity

Execution result

Generation-path evidence

Failures/repairs

Unverified properties

BP class

BV class if applicable

PASS / FAIL / INCONCLUSIVE

B.3 Maximum-Assurance Prompt

Run this as a maximum-assurance PARS exact-binary experiment.

Do not optimize for obtaining a PASS.

Optimize for correctly distinguishing PASS from FAIL or INCONCLUSIVE.

Freeze the complete task contract before generation.

Maintain an explicit hard-invariant ledger.

Preserve every material failed candidate.

The final executable must be authored as a complete byte-level representation without a compiler, assembler, linker, source-language binary build, prebuilt executable, or prebuilt object artifact.

Freeze the full byte representation before accepted execution.

Reconstruct a clean executable solely from that frozen representation.

Verify exact byte count and SHA-256 identity.

Inspect the resulting format and architecture independently.

Execute only the reconstructed artifact.

Capture observable runtime evidence.

Audit the claimed generation path independently of runtime success.
Attempt to falsify BP2.

If BP2 cannot be established, downgrade to the strongest class that can be established.
If any hard requirement is unverifiable, return INCONCLUSIVE rather than converting uncertainty into PASS.

Preserve all verification receipts required for independent replication.

B.4 Procedural Visual / BV2 Prompt

Activate PARS-Deep and the Exact-Byte Visual / Binary Image rules.

TARGET

Create [VISUAL TARGET] using an exact executable binary.
Required claim class: BV2 - PROCEDURAL RUNTIME RENDERER.

HARD GENERATION CONSTRAINTS

NO compiler.
NO assembler.
NO linker.
NO source-language build path.
NO image-generation model/tool in the generation path.
NO embedded finished raster.
NO embedded finished frame sequence.
NO embedded finished video.
NO prebuilt executable implementing the renderer.

BV2 REQUIREMENT

The visible pixels/frames must be computed algorithmically at runtime by the sealed executable. Compact procedural parameters, geometry, primitives, algorithms, or structural data may be encoded. The accepted finished visual may not merely be stored and emitted.

CONTAMINATION TEST

Search the executable for evidence that the accepted finished raster/video is embedded directly or in disguised/compressed form. Use size/layout inspection, payload search, output-expansion analysis, or another discriminating method. A visually superior result with weaker provenance must be downgraded rather than promoted.

EXACT-BYTE PROOF

Freeze and preserve the complete executable byte representation. Record byte count and SHA-256. Reconstruct the binary from the frozen representation. Verify byte-for-byte identity. Execute the reconstructed binary. Hash/identify the runtime output. Preserve the relationship between executable identity and runtime result.

FINAL CLASSIFICATION

Return BP class, BV class, executable identity, runtime-result identity, forbidden-path status, contamination-test result, and PASS / FAIL / INCONCLUSIVE.

B.5 Contract-Repair Prompt

The current candidate is INVALID_FINAL_STATE.

Failure:

[EXACT VIOLATED INVARIANT]

Do not discard the rest of the valid task contract.
Preserve all unaffected evidence and failure history.
Return to Reconstruct.

Remove the invalid generation path and rebuild every downstream dependency affected by that change. Do not solve the failure by weakening or deleting the violated invariant.

Before Build, replay the complete ACTIVE hard-invariant set against the repaired candidate.
After Build, rerun every materialization-sensitive check.

A repaired candidate is not a PASS until all gates pass.

B.6 Independent Audit Prompt

Audit the supplied exact-binary experiment.
Do not attempt to rescue or improve the artifact.
Your task is classification.

Evaluate independently:

1. artifact identity;
2. completeness of the disclosed byte representation;
3. reconstruction identity;
4. byte count and SHA-256;
5. executable format;
6. architecture;
7. runtime execution evidence;
8. output identity;
9. prohibited generation-path evidence;
10. claimed BP class;
11. claimed BV class where applicable;
12. hard-invariant compliance.

Actively search for evidence that would downgrade the claimed class.

Do not infer BP2 from BP1.

Do not infer BV2 from a visually convincing result.

Return the strongest class actually supported.

Final classification:

PASS / FAIL / INCONCLUSIVE

Appendix C - PARS-FEBP Evidence Receipt [Contents](#)

PARS-FEBP EVIDENCE RECEIPT

Experiment ID:
Date:
PARS specification/version:
Model/configuration:

TARGET
Artifact:
Platform:
Format:
Architecture:
Required behavior:

CONTRACT
Claim target:
Forbidden generation paths:
Active hard invariants:

FROZEN REPRESENTATION
Representation format:
Representation identity:
Complete representation preserved: YES / NO

MATERIALIZATION
Materializer:
Materializer role:
Semantic transformation detected: YES / NO / UNRESOLVED

ARTIFACT
Byte count:
SHA-256:
Format verification:
Architecture verification:
Entry point:

RECONSTRUCTION
Independent reconstruction:
Reconstructed byte count:
Reconstructed SHA-256:
Direct byte comparison:
IDENTICAL / NOT IDENTICAL

EXECUTION
Environment:
Network state:
Exit status:
Observed behavior:
Expected behavior satisfied:

RUNTIME OUTPUT
Output type:
Output byte count:
Output SHA-256:
Accepted-result identity:

PROVENANCE
Compiler evidence:
Assembler evidence:
Linker evidence:
Prebuilt artifact evidence:
Generation-path audit:

Hostile provenance test:

CLASSIFICATION

BP:

BV:

Unverifiable properties:

Material failures preserved:

Final status:

PASS / FAIL / INCONCLUSIVE

Appendix D - PARS-ECS Case Study Template [Contents](#)

PARS-ECS (PARS Exact-Binary Case Study Standard) is a paper-defined reporting standard. It is derived from the evidence-preservation principles in [\[S1\]](#) and the historical evidence inventory; it is not a named standard in the supplied PARS specification.

Every substantive case study should contain:

1. Case identification and immutable experiment ID.
2. Objective.
3. Claim target (BP/BV and forbidden paths) frozen before result.
4. Hard-invariant ledger.
5. Exact prompt identity or explicit statement that prompt text was not recovered.
6. Externally relevant generation strategy.
7. Complete byte-representation availability.
8. Materialization record.
9. Accepted artifact byte count/hash.
10. Independent reconstruction identity.
11. Structural inspection.
12. Execution evidence.
13. Runtime-result identity.
14. At least one hostile test.
15. Failure/repair ledger.
16. Final BP/BV classification.
17. What the case establishes.
18. What it does not establish.
19. Reproducibility-package contents.

Appendix E - Benchmark Preregistration Template [Contents](#)

PARS-FEBP BENCHMARK PREREGISTRATION

Benchmark version:

Date frozen:

Model:

Model configuration:

Tool availability:

Execution environment:

Task bank identity:

Development cohort:

Held-out cohort:

Conditions:

Exact frozen prompt text:

Retry policy:

Repair policy:

Resource limits:

Primary endpoint:

CCEBS

Secondary endpoints:

M1-M11

Failure taxonomy:

F1-F10

Ablations:

ABL-1 through ABL-8

Predicted ablation failures:

Severe-regression vetoes:

Scoring implementation identity:

Verifier identity/hash where applicable:

Exclusion rules:

Promotion rule:

Known limitations:

Frozen by:

Appendix F - Historical Artifact Identity Index

[See Table 3.](#)
[Contents](#)

Table 3. Historical artifact identity index and source-key mapping.

Source ID	Artifact	Bytes	Format	SHA-256 / identity	Role
H1	Starship ELF	907	ELF64 x86-64	1df35f8f...030a354	Worked example
H2	LHC Collider v2.1	2,048	ELF64 x86-64	04d3da29...507b1c	Flagship historical case
H3	WASM Hyperlattice	9,433	WebAssembly v1	c1d31cb6...37e7c3	Cross-format case
H4	Binary Brains Gate 9	34,175	ELF64 x86-64	a8bd61cc...308cf	Causal multiprocess case
H5	PARS-VM/4	5,392	ELF64 x86-64	5a002c60...908e1	Virtual-computer case
H6	House of the Rising Sun	57,344	ELF64 x86-64	4d4fe5c9...d1ff10	Scaling case
H7	BVIS-001	228	ELF64 x86-64	See audit package	INCONCLUSIVE audit case

Appendix G - Mechanical Reproduction Examples [Contents](#)

These examples illustrate only literal materialization and evidence operations. They are not unpublished binary-construction techniques.

Literal Hex to Bytes

A frozen representation consisting of hexadecimal byte pairs may be decoded mechanically into bytes. The decoder must not interpret mnemonics, labels, symbols, or executable semantics.

Identity

After materialization:

1. record byte count;
2. compute SHA-256;
3. reconstruct independently from the frozen representation;
4. compare byte-for-byte;
5. inspect format/architecture;
6. execute the reconstructed artifact under containment;
7. identify runtime output;
8. replay post-Build invariants.

Appendix H - Evidence Package Layout [Contents](#)

```
/PARS-EXPERIMENT-ID
|
|-- 00_README.md
|-- contract/
|   |-- target.md
|   |-- invariants.json
|   `-- prompt.txt
|-- generation/
|   |-- frozen_byte_stream.hex
|   |-- byte_stream.sha256
|   `-- generation_receipt.json
|-- artifact/
|   |-- artifact.bin
|   `-- artifact.sha256
|-- reconstruction/
|   |-- reconstructed.bin
|   |-- reconstructed.sha256
|   `-- identity_receipt.txt
|-- inspection/
|   |-- format.txt
|   |-- architecture.txt
|   `-- structural_receipt.json
|-- runtime/
|   |-- execution_receipt.json
|   |-- stdout.bin
|   |-- stderr.bin
|   `-- output/
|-- provenance/
|   |-- generation_path.md
|   |-- prohibited_path_checks.json
|   `-- hostile_tests/
|-- failures/
|   |-- candidate_001/
|   |-- candidate_002/
|   `-- failure_ledger.json
|-- media/
|   |-- screenshots/
|   `-- evidence_video.mp4
`-- results/
    `-- PARS_FEBP_EVIDENCE_RECEIPT.json
```

Appendix I - References and Primary Source Artifacts [Contents](#)

The integrated manuscript uses compact source identifiers. Final publication formatting may convert these into footnotes or a formal references section.

[S1] PARS-Deep Model Execution Specification, v1.25.0-candidate.6 — Final Invariant Gate Repair Edition. Primary architectural specification used in this paper. The source labels candidate.6 a non-authoritative experimental repair candidate and preserves its own authority/promotion boundaries.

[H1] SHOWCASE_RECEIPT.md; STARSHIP_FREEHAND_BYTE_STREAM.txt. Historical 907-byte Starship exact-binary evidence package: frozen byte representation, artifact identity, reconstruction, runtime output, and recorded provenance claim.

[H2] PARS_LHC_COLLIDER_V2_I_SHOWCASE_RECEIPT.md; PARS_LHC_COLLIDER_V2_I_EXACT_BYTE_STREAM.txt. Historical LHC-inspired exact-binary renderer evidence package: candidate lineage, one-byte repair, reconstruction, deterministic runtime replay, and recorded BP2/BV2 classification.

[H3] PARS_WASM_HYPERLATTICE_VERIFICATION_RECEIPT.txt. Historical WebAssembly exact-byte evidence package: direct module serialization claim, zero-import module contract, regeneration, perturbation, runtime capture, and proof-video receipt.

[H4] BINARY_BRAINS_CONTINUITY(3).md and associated Gate 3/5 verification records. Historical Binary Brains evidence: Gate 9 authority, frozen-stream reconstruction, causal counterfactual, process evidence, H.264 proof, and provenance caveats.

[H5] PARS_VM4_PROOF_RECEIPT.md. Historical PARS-VM/4 evidence: exact-binary virtual computer, guest CPU/ISA/RAM/VGPU architecture, interaction proof, reconstruction, runtime scope, and provenance boundary.

[H6] RISING_SUN_HOUSE_FINAL_RELEASE_RECEIPT.md. Historical 57,344-byte exact executable evidence: canonical re-emission, 210-second runtime, generated PCM, complete byte-page proof coverage, capture separation, and final release gate.

[H7] INDEPENDENT_AUDIT.md. Separate BVIS-001 audit artifact: BP1 reconstruction and runtime evidence reproduced, but strict upstream BP2 provenance remained unestablished; final audit verdict INCONCLUSIVE.