The Trust Framework for Computational Physics v1.0: Hash-Verified Trajectories and Reproducible Dynamical Systems

Bradley C. Peter

November 29, 2025

Abstract

Reproducibility is a cornerstone of scientific validity, yet numerical simulations in physics often lack the rigorous reproducibility and falsifiability of traditional experiments. We present the *Trust Framework for Computational Physics*, a formal methodology that contributes to reproducible and testable computational experiments through trajectory-level hashing, strict precision controls, and canonical data serialization. We develop the theoretical foundations of this framework, introducing formal definitions (trust anchors, canonical trajectories, precision/integrator/serialization boundaries, σ-stability, falsification criteria) and establishing two core principles: the Trust Equivalence Principle and the Canonical Trajectory Invariance Principle. These principles, under explicit assumptions, establish that within fixed trust conditions (e.g., specified precision and integrator settings), independent simulations of the same dynamical system will produce identical hashed trajectories, providing a protocol-specific signature of the numerical result. We discuss how trust boundaries (on precision, discretization error, etc.) delineate the regime where numerical results are reliable and scientifically falsifiable. To demonstrate generality, we present three case studies drawn from existing trust datasets: (1) linear mechanical stability in the canonical bicycle model (the Whipple–Meijaard bicycle), (2) nonlinear gravitational dynamics in the three-body problem (high-precision periodic orbits), and (3) a curvature-dependent diffusion model illustrating a gravitational arrow-of-time. In each case, the Trust Framework yields reproducible results with cryptographic hash anchors and cross-verification scripts. We show how trajectory-level hashing acts as a numerical signature, and how exceeding trust boundaries leads to divergent outcomes that signal genuine physical disagreement or numerical error – thereby contributing to the falsifiability of computational results. Finally, we outline the architecture of an ideal "trust dataset" (including notebooks, summary data, hash manifests, and verification scripts) and argue that the Trust Framework lays a foundation for an open ecosystem of hash-verified dynamical models.

1 Introduction

Reproducibility in science—the ability for independent researchers to obtain the same results given the same conditions—is essential for trust and validation. In recent years, concerns over a "reproducibility crisis" have emerged across disciplines, including computational science, where the complexity of codes and the sensitivity of numerical models can make exact reproduction challenging. Computational physics, in particular, faces unique reproducibility issues: simulations often depend on machine-specific arithmetic, algorithmic choices, and implicit parameters that may not be fully specified in publications. The consequence is that

two groups attempting to simulate the "same" physical scenario might obtain results that quantitatively (or even qualitatively) disagree, without an obvious way to determine which (if either) result is correct.

The scientific method demands not only reproducibility but also falsifiability—the possibility to definitively test whether a given result or theory is correct or not.Traditional experiments achieve falsifiability by comparisons to nature: an independent experiment can confirm or refute a measurement within error bounds.Numerical experiments, however, often lack a clear falsification criterion: if two simulations disagree, is the physics in question falsified, or is it simply a matter of different numerical error? How can one prove that a given computational result truly reflects an underlying physical truth rather than a numerical artifact?

This paper proposes a solution in the form of a formal architecture for reproducible computational experiments, which we call the Trust Framework for Computational Physics (version 1.0).The core idea is to treat a computational result—for example, a trajectory of a dynamical system—as a well-defined, hash-verifiable scientific artifact.By imposing strict conditions on the simulation (fixed precision arithmetic, specified integrator schemes and tolerances, and canonical data output formats), we ensure that any faithful implementation of the same experiment yields an identical data object (to within a deterministic tolerance) which can be verified by a cryptographic hash. This approach elevates a computed trajectory to a digital signature of the physical system under the specified conditions: any deviation indicates a failure to reproduce the physics or a violation of the stated "trust" conditions, thereby flagging a potential error or new physics. In effect, the framework allows computational results to be treated with similar rigor as analytical solutions or experimental measurements—unique, stable, and testable.

The Trust Framework builds upon recent progress in reproducible research and numerical analysis.Efforts in other fields have highlighted the importance of open-source code, data sharing, and environment capture (e.g., containers) for reproducibility. Our contribution is to formalize reproducibility at the level of the dynamical trajectory and to introduce hash-based verification as a quantitative test of equivalence. Drawing inspiration from cryptographic integrity checks (widely used to verify data and software integrity), we treat a numerical trajectory's hash (e.g., SHA-256) as a*trust anchor*—a fixed reference against which future computations can be compared.This approach is conceptually similar to using invariants in mathematics or physics, where conserved quantities can verify correctness, though hashes are protocol-specific rather than universal.

The remainder of this paper is structured as follows. Section 2 provides background and motivation. Section 3 introduces the Trust Framework formally, with precise definitions of key concepts such as trust boundaries (limits on precision, integrator error, etc.), canonical trajectories (the definitive trajectory data under those limits), and falsification conditions (criteria to declare a result inconsistent or "wrong"). Section 4 presents the theoretical core: the Trust Equivalence Principle and the Canonical Trajectory Invariance Principle. Section 5 discusses trust boundaries and falsification criteria. Section 6 elaborates on stability and numerical error. Section 7 details serialization and hash anchors. Section 8 addresses limitations and operational considerations. Section 9 describes the architecture of a trust dataset. Section 10 presents three case studies. Finally, Section 11 provides a discussion of broader implications, and Section 12 concludes with future outlook.We maintain a formal tone throughout, with rigorous definitions and clear principles, striving for a self-contained exposition.

2 Background and Motivation

Modern computational physics simulations range from simple ODE integrations to massive high-performance computing (HPC) codes solving PDEs for climate, astrophysics, or plasma physics. In all cases, reproducibility has multiple facets: *numerical reproducibility* (getting the same numerical results on different runs or platforms) and*physical reproducibility*(getting results consistent with the true physical solution or experiment). Both aspects can be undermined by subtle computational issues. Floating-point arithmetic is not associative or distributive, meaning that even tiny round-off differences can accumulate and lead to divergent outcomes in long or chaotic simulations.For example, running the same chaotic climate model on two different architectures may yield climate projections that decorrelate after some simulation time due to minute differences in arithmetic – a disconcerting phenomenon when results are used for high-stakes predictions. Likewise, in molecular dynamics or$N$-body simulations, different integrator step sizes or random number sequences can lead to statistically different outcomes if not carefully controlled.These issues have prompted increasing attention to the reliability of computational results.Researchers have begun to ask:how can we trust that a published simulation result is "correct" or representative of physical truth?

Several studies highlight the severity of the problem. For instance, Nazar´e*et al.*(2020) explicitly demonstrated that finite precision effects can limit the reproducibility of chaotic circuit simulations: using the same circuit model on different computers produced measurably different trajectories beyond a certain time horizon, even when nominally running the same experiment. They conclude that simulation reliability must be evaluated carefully, especially for chaotic dynamics.Rounding-error analyses by other authors have similarly shown that the long-term behavior of chaotic maps or systems is strongly influenced by floating-point rounding choices, to the point where na¨ıve simulations can produce misleading results. These works underline a key point: theoretical chaos amplifies computational uncertainty. What can be done about this? Approaches like interval arithmetic or arbitrary-precision arithmetic have been suggested to obtain reliable bounds on trajectories. Indeed, the so-called "Clean Numerical Simulation" approach uses very high precision to effectively eliminate numerical chaos over a finite duration, allowing one to get reproducible trajectories for a chaotic system up to a known time limit. Our Trust Framework embraces arbitrary precision as needed, but embeds it in a larger architecture of data validation.

Another line of motivation comes from the philosophy of science and the notion of falsifiability. In principle, a numerical model is falsifiable if one can demonstrate that no exact solution of the model corresponds to the reported numerical solution. However, in practice it is hard to "disprove" a numerical result unless an analytical solution or an obviously better-resolved numerical solution is available. There is a gap in how we treat numerical evidence versus experimental evidence. The Trust Framework aims to close this gap by making the comparison of independent computations a meaningful test. If two teams integrate the same equations under agreed-upon conditions and obtain different results (e.g. different trajectory hashes), then at least one simulation must be wrong – a situation analogous to two physical experiments producing discrepant measurements, which would prompt investigation. By formalizing the conditions (the trust boundaries) under which results should agree, we create a concrete falsification criterion: violation of the Trust Equivalence Principle (hash mismatch) falsifies the assumption that both simulations were correct implementations of the model. This provides an actionable test for correctness:when a mismatch occurs, it is not "just one of those things" but evidence of a problem (be it a bug, inadequate precision, or even a

fundamentally new physical effect if one simulation included factors the other missed).

Even outside of highly chaotic systems, issues of numerical consistency can plague simulations. For example, different solver algorithms for a stiff system can converge to different stable states if parameters or tolerances are not aligned, and if authors do not fully specify their numerical methods, exact replication becomes impossible. The Trust Framework insists on specifying a canonical numerical method or at least a tightly constrained family of methods (with fixed error tolerances), effectively treating the numerical method as part of the experiment's protocol. This is analogous to specifying an experimental procedure in a lab:if one researcher heats a sample at 1°C/min and another heats at 10°C/min, they may get different outcomes; the protocol must be standardized. Likewise, we standardize the "protocol" of a numerical experiment via trust boundaries.

A further motivation comes from the concept of digital signatures and data integrity, which is well-established in computer science but underutilized in computational science. Cryptographic hash functions (such as SHA-256) are routinely used to verify data integrity (for example, to ensure a software download has not been corrupted). The key property of such hashes is that even a one-bit difference in the input produces a completely different hash, and it is (with extremely high probability) infeasible to find two distinct inputs with the same hash. Thus, a hash can serve as a unique fingerprint for data. In our framework, we use hashing to fingerprint an entire simulation output (or some critical subset of data) and to enforce an all-or-nothing equivalence criterion.If one group publishes the hash of their canonical trajectory, another group can run their own code and compute the hash of their trajectory; if it matches, they have automatically validated that their result is identical to the original (within the prescribed tolerances), with no need to compare every data point manually. This shifts the burden of comparison from a subjective or detailed analysis ("Do these plots look the same?")to an objective binary decision ("Do these results hash the same?").

There have been important efforts in recent years toward more reproducible computational science. Reproducibility networks in various countries and new journal policies (for example, some journals now require authors to provide code and data as supplemental materials) are steps toward better practices. However, to date, the emphasis has been on openness and repeatability (sharing code/data so others can rerun the exact same program) rather than on guaranteeing that independent re-implementations yield identical outcomes.The Trust Framework goes a step further by not relying on the exact same code or computing environment, but only on the same agreed specifications. In doing so, it addresses a deeper notion of reproducibility often called *replicability* or independent verification. It is this higher standard – akin to rebuilding an experiment from scratch and still getting the same result – that scientific knowledge ultimately demands.Our framework provides a concrete pathway to achieve it in the computational realm.

In summary, the motivation for the Trust Framework arises from addressing the reproducibility crisis, accounting for finite precision and chaos, and establishing falsifiability for numerical results.

3 Formal Definitions

We now introduce the key definitions that form the language of the Trust Framework. These definitions are constructed to be precise and unambiguous, suitable for use in principle statements and proofs. Throughout, consider a well-posed dynamical system (continuous- or

discrete-time) with specified model equations, parameters, and initial/boundary conditions that define a unique solution trajectory (or a distribution of trajectories, if the system is stochastic). For concreteness, one may think of an ODE or PDE with given initial/boundary conditions, but the framework is general.

Definition1(Trust Anchors). A *trust anchor* is a piece of information that is fixed and agreed upon as the basis for reproducibility of a computational experiment. In the context of a simulation, the trust anchors typically include: (a) the exact specification of the physical model (e.g., the equations of motion, material properties, parameters, etc.); (b) the initial conditions and boundary conditions for the simulation (and any other input data defining a particular scenario); (c) the computational protocol: the numerical method details, including the prescribed precision, integrator scheme and settings, time-step size or solver tolerances, and any other settings that can affect the results. Together, these anchors define the experiment so completely that any variation outside them is considered a different experiment. Additionally, once a canonical trajectory (see Definition) has been established for a given experiment, the final reference hash of that trajectory can itself serve as a trust anchor for future comparisons. In summary, trust anchors are the fixed reference specifications that all parties must use identically in order for their results to be considered comparable.By design, trust anchors are publicly shareable and verifiable. (For example, publishing the exact ODE and initial state, the integration scheme and parameters, and the resulting SHA-256 hash of the trajectory data provides all the necessary anchors to reproduce and verify the experiment.)

Definition 2 (Precision Boundary)**.** The *precision boundary* specifies the numerical precision at which all calculations in the experiment must be carried out. It can be defined in terms of floating-point format (e.g., 64-bit double precision, 80-bit extended, or a specified number of decimal digits in arbitrary precision) or in terms of an upper bound on rounding error. The precision boundary ensures that any implementation uses arithmetic that is at least as precise as the agreed standard. For instance, a precision boundary might state: "All computations must be done with at least 70 decimal digits of precision (rounding to nearest), guaranteeing rounding error $< 10^{-70}$." This puts an upper bound on truncation or round-off error due to arithmetic. Using lower precision than specified constitutes a violation of the precision boundary, as it could alter results beyond the tolerances.

Definition 3 (Integrator Boundary)**.** The *integrator boundary* specifies the numerical integration or solver method to be used (or a narrowly defined family of acceptable methods) and its key settings. This includes constraints such as a fixed time-step size or an adaptive step size with a given local error tolerance, and may also cover algorithmic details (e.g., requiring a symplectic integrator for a Hamiltonian system, or a particular regularization for handling singularities). The integrator boundary effectively bounds the discretization error by ensuring all implementations solve the same numerical problem. For example, an integrator boundary might require: "Use a 4th-order Runge–Kutta method with fixed time step $h = 10^{-3}$," or "Use an adaptive 8th-order integrator with local error tolerance 10^{-12}." Any implementation deviating from the specified method or using a step size outside the allowed range has crossed the integrator boundary, and its results are not considered equivalent under the framework.

Definition 4 (Serialization Boundary)**.** The *serialization boundary* defines how results (the trajectory data) are recorded, rounded, and formatted for comparison. It includes the output grid (e.g., the set of time points at which the solution is recorded), the data format (for

example, a JSON file with specified structure, or a CSV with designated columns and units), and the numeric rounding or truncation rule for output values. The serialization boundary ensures that two identical trajectories produce bitwise-identical output files. For instance, a serialization boundary might specify: "Output the trajectory as a JSON array of $(t, x(t))$ pairs with time step $\Delta t = 0.01$. Represent each floating-point number in scientific notation with 14 significant digits, rounding to the nearest 10^{-12}, and use UTF-8 encoding with Unix newline conventions." This determines a unique representation for the data. Any deviation (such as using 8 significant digits or a different output ordering) would produce a different raw output file and hence a different hash. In effect, the serialization boundary bounds the *representation error*: differences smaller than the chosen rounding increment are deemed irrelevant (they will be rounded away and not appear in the output), whereas any difference larger than that increment will manifest in the output and thus be detectable via the hash.

Definition 5 (Canonical Trajectory)**.** Given a specified dynamical system and a set of trust anchors (including fixed precision, integrator, and serialization boundaries), the *canonical trajectory* is defined as the unique numerical trajectory obtained when the system is simulated under those conditions. Formally, one can think of the canonical trajectory as a function $T(t)$ (in the continuous-time case) or sequence of states $\{T_n\}$ (in a discrete or time-stepped case) that results from applying the agreed-upon numerical procedure with no degrees of freedom left unspecified. In practice, we obtain the canonical trajectory by actually running the simulation with the given trust anchors (precision, integrator, etc.). If the integrator boundary involves taking a sufficiently small time step or tight tolerance, we assume this yields a trajectory close to the true solution of the equations; any remaining numerical deviation is accepted as part of the canonical result. The canonical trajectory thus serves as the standard of comparison (the "ground truth" within the trust framework). By definition, it is deterministic and reproducible: any correct implementation of the simulation that adheres to the same trust anchors should produce the same canonical trajectory (or a trajectory indistinguishable from it within the specified output rounding). We treat the canonical trajectory as a data object (for example, a table of times and state vectors) which can be serialized and hashed. The file or data object containing this trajectory in the agreed format is sometimes called the *canonical output* of the experiment. Importantly, the canonical trajectory is not necessarily the exact analytic solution of the equations (which may be unknown), but it is the definitive numerical solution under the chosen conditions. If one later decides to improve the method (e.g., uses higher precision or smaller steps beyond the current trust boundaries), one might define a new canonical trajectory (with a new version number, perhaps). But within a fixed framework version, the canonical trajectory is the final authoritative result that all participants aim to reproduce.

Definition 6 (Trajectory Hash (Hash Anchor))**.** Given a canonical trajectory data object D (for example, a file containing the trajectory in the canonical serialization format), we define $H = \mathcal{H}(D)$ as the cryptographic hash of that data. Here $\mathcal{H}(\cdot)$ is a chosen hash function (such as SHA-256). The *trajectory hash* H serves as a compact *hash anchor* for the result: any change in the trajectory data D (even a single bit) will, with overwhelming probability, produce a completely different hash value $\mathcal{H}(D') \neq H$. In practice, floating-point computations on different hardware might yield tiny differences in least significant bits; the serialization boundary (see Definition) is set up to eliminate such differences by rounding/truncating to an agreed precision.Thus, two independent implementations that produce numerically identical trajectories within the allowed tolerance will produce bitwise-identical serialized data D, and therefore the same hash H. The trajectory hash is the unique

digital fingerprint of the canonical trajectory, and it is treated as a conserved quantity under the Trust Framework. Once H is published as part of the trust anchors, anyone can recompute the hash of their own result and compare to H to verify whether they obtained the same canonical trajectory.

Definition 7 (σ-Stability)**.** We define *σ-stability* as a property of a trajectory (or outcome) indicating that it is robust to small perturbations (in initial conditions or intermediate calculations) up to a level σ, over the time (or iteration) interval of interest. More concretely, a trajectory $T(t)$ is σ-stable (under the given trust conditions) if any other trajectory $T'(t)$ generated with perturbations within the trust boundaries (for example, a slightly perturbed initial state within $|\Delta x_0| < \sigma$, or numerical round-off differences within the allowed precision) stays within a distance σ of $T(t)$ for all t in the simulation range $[0, T_{\max}]$. Intuitively, σ-stability means that the trajectory does not diverge significantly when subjected to small disturbances up to size σ. If a system is highly sensitive or chaotic, achieving a small σ may require extremely tight trust boundaries (such as very high precision) to control numerical perturbations. In practice, one chooses trust boundaries such that the expected numerical differences (due to round-off, step size, etc.) remain below a chosen σ threshold, ensuring that all runs yield effectively the same result. Sigma-stability can be empirically tested by verifying that slight variations in the simulation (for instance, using a marginally smaller time step within the allowed range, or running on a different machine with the same precision) do not change the output beyond σ. If they do, the chosen trust boundaries are insufficient (the system is less stable than assumed), and one would tighten those boundaries (increase precision, etc.) until the results become σ-stable for the desired σ. In summary, σ-stability quantifies reproducibility: a highly σ-stable result (with very small σ) means the outcome is extremely insensitive to implementation details, which is precisely what we enforce.

Definition 8 (Falsification Conditions)**.** Within the Trust Framework, a *falsification condition* is a well-defined criterion that, if met, indicates the failure of the hypothesis that a given computational result is correct (or that the underlying model holds in the scenario tested). We distinguish two levels of falsification: **Numerical falsification:** This is triggered when the reproducibility assumptions are violated. The primary condition is a *hash mismatch* between independent implementations: if two reputable parties produce different trajectory hashes under the agreed trust anchors, at least one result is falsified as a trustworthy solution. In other words, failure to achieve matching results under identical conditions means the computational claim is not confirmed. Another numerical falsification condition can be an internal inconsistency: for example, if a model is supposed to conserve energy and the trusted trajectory exhibits energy drift beyond the allowed numerical error, that would falsify the assumption that the integrator was sufficiently accurate or that the implementation followed the model correctly. Essentially, any breach of the prescribed trust boundaries (such as detecting lower precision or larger error than specified) falsifies the result as being within the agreed regime. **Physical falsification:** This refers to falsifying the scientific model or theoretical prediction itself (as opposed to just the numerical execution). For example, if a trust-verified simulation makes a prediction about a physical observable and a laboratory experiment measures a significantly different value, that discrepancy falsifies (or at least challenges) the model or hypothesis underlying the simulation. While this goes beyond purely computational verification, the Trust Framework assists by ensuring the computational prediction is solid and reproducible—so that any disagreement with experiment is due to physics, not numerical issues. In the context of trust datasets, one might also consider cross-model falsification: for instance, if two different theoretical models (say Newtonian gravity versus a modified gravity

theory) are both simulated under trust conditions for the same scenario, their trajectory outputs can be directly compared. If they yield clearly different predictions beyond known uncertainties, then at least one model will be falsified when confronted with empirical data. The Trust Framework only certifies that a numerical result is a self-consistent solution of the stated model under fixed conditions. It does nothing to address whether the model itself is appropriate; physical falsification still requires comparison with experiment and sound model selection. In practice, the core falsification criterion provided by the Trust Framework is: a computational result is not considered fully trustworthy (i.e., it remains unvalidated) until an independent reproduction under the same trust anchors yields a matching trajectory or hash. Conversely, if such an independent reproduction fails to match (hash mismatch occurs), the result is deemed *falsified* as a unique outcome under those conditions; either one of the implementations is incorrect or the trust anchors were not properly respected. This approach mirrors the role of independent experimental reproduction in traditional science: new computational discoveries (e.g., a new orbit or a new phenomenon in simulation) are not fully accepted until independently verified in this manner.

Definition 9(Trajectory Equivalence)**.** Two simulation outputs (trajectories) are considered *equivalent under the Trust Framework* (or *trust-equivalent*) if they produce identical serialized trajectory data under the agreed trust anchors, and hence yield the same trajectory hash (see Definition).Equivalently, at every specified output time or grid point, the state values from the two simulations differ by no more than the allowed rounding tolerance of the serialization.Trajectory equivalence defines an equivalence relation between results: all simulations that satisfy the same trust anchors and are trajectory-equivalent effectively reproduce the same physical outcome. The Trust Equivalence Principle (see Principle) formalizes this notion by asserting that under the given conditions, all correct simulations are guaranteed to be trajectory-equivalent.

Definition 10 (Trust Dataset)**.** A *trust dataset* is a self-contained collection of digital artifacts that encapsulates a computational experiment in the Trust Framework. It includes all information and files needed to reproduce and verify the experiment's results. Specifically, a trust dataset typically contains: (i) a formal specification of the model and trust anchors (often as a document or README detailing the equations, parameters, precision, integrator, serialization format, etc.), (ii) the canonical output data (the canonical trajectory or result, provided in the agreed-upon serialization format, often accompanied by plots or summaries for convenience), (iii) a manifest listing the cryptographic hashes of the data files (to ensure file integrity and allow quick hash-based comparisons), and (iv) scripts or notebooks for verification and analysis (for example, code that recomputes the trajectory using independent software or that reads the data and checks the hash and key properties like conservation laws). A trust dataset is typically archived in an archival repository (such as Zenodo) with a DOI, to ensure long-term accessibility. The combination of the trust dataset and the whitepaper describing it (such as this document) provides a complete package: the dataset contains the raw materials and instructions for reproduction, while the paper provides the theoretical context, interpretation, and broader implications.

Definition 11(Trust-Verified Result)**.** We say a computational result (trajectory or other output) is*trust-verified* once it has been independently reproduced under the same specified trust anchors and found to be trajectory-equivalent to the original result. In other words, at least one independent implementation has generated a matching output (yielding the same hash, per Definition) as the original implementation. A trust-verified result is considered

confirmed within the framework. Until such verification, a result (even if produced with internal rigor) remains unverified in the sense that it could, in principle, be a numerical artifact or an implementation error. The requirement of independent reproduction with matching hashes ensures that the result is not tied to a single code or hardware instance. Once a result is trust-verified, it can serve as a reliable reference for further studies (and may be added to a library of trust datasets for the community). This notion parallels the idea in experimental science that a measurement becomes firmly accepted only after it has been independently replicated.

4 Theoretical Core

Having laid out the definitions, we now present two fundamental principles that guarantee the reproducibility and uniqueness of outcomes under the Trust Framework, under explicit assumptions. These are the *Trust Equivalence Principle*, which formalizes the fact that all simulations obeying the same trust anchors yield the same result, and the *Canonical Trajectory Invariance Principle*, which states that once trust boundaries are set, further tightening them does not change the outcome (up to the prescribed resolution). These principles assume deterministic arithmetic semantics, fully specified integrator behavior, and no nondeterministic elements such as parallelism or random seeds unless fixed as anchors. They are engineering patterns supported by empirical evidence from the case studies, not rigorous theorems in the mathematical sense.

Principle 1 (Trust Equivalence Principle)**.** *Consider a dynamical system S and a fixed set of trust anchors specifying the model equations, initial conditions, precision boundary P, integrator boundary I, and serialization boundary Z. Let T be the canonical trajectory obtained under these conditions, with canonical serialized output D and hash $H = \mathcal{H}(D)$. Now let $\mathcal{A}$ and $\mathcal{B}$ be any two independent implementations (two different codes, possibly written in different languages or running on different machines) that both aim to simulate S under the same trust anchors (P, I, Z). Suppose both $\mathcal{A}$ and $\mathcal{B}$ execute without error and adhere strictly to these anchors. Then the outputs $D_{\mathcal{A}}$ and $D_{\mathcal{B}}$ produced by $\mathcal{A}$ and $\mathcal{B}$ will be bitwise identical. In particular, they produce the same canonical trajectory T and thus $D_{\mathcal{A}} = D_{\mathcal{B}} = D$. Consequently, their trajectory hashes will satisfy $\mathcal{H}(D_{\mathcal{A}}) = \mathcal{H}(D_{\mathcal{B}}) = H$. In other words, under the given trust conditions, all correct simulations are equivalent and yield the unique trusted result. Conversely, if $D_{\mathcal{A}} \neq D_{\mathcal{B}}$ (hash mismatch), then at least one of $\mathcal{A}$ or $\mathcal{B}$ has failed to meet the trust conditions or contains an implementation error.*

Proof Sketch: The principle formalizes the intuitive claim that the trust conditions define a unique numerical experiment whose outcome is deterministic. The proof can be broken into parts corresponding to each trust boundary: *Integrator and precision determinism:* Because both implementations use the same numerical method with the same step sizes (or adaptive criteria) and the same arithmetic precision and rounding mode, the sequence of computed states x_n produced by each should be identical at each step. This can be shown inductively. Assume after n steps, $\mathcal{A}$ and $\mathcal{B}$ have identical states. Then their states after step $n + 1$ will also coincide, since each computed update is a deterministic function of the previous state (and the floating-point operations, given the same inputs and rounding, produce the same result bit-for-bit [**?**]). Starting from the identical initial conditions (a trust anchor), it follows by induction that $x_n^{(\mathcal{A})} = x_n^{(\mathcal{B})}$ for all n up to the final step. *Serialization consistency:* Even if $\mathcal{A}$ and $\mathcal{B}$ generate the same sequence of states, one must ensure they produce the

same output representation. The serialization boundary Z guarantees this by prescribing an unambiguous output format and rounding procedure. Thus, both simulations will serialize the identical state sequence in exactly the same way (same output times, same numeric formatting, etc.). Any minute differences (e.g., in least significant bits) are eliminated by rounding to the agreed precision. Therefore, the output files $D_{\mathcal{A}}$ and $D_{\mathcal{B}}$ are identical bit-for-bit. Given these points, it follows that $D_{\mathcal{A}} = D_{\mathcal{B}} = D$, the canonical output. Consequently, $\mathcal{H}(D_{\mathcal{A}}) = \mathcal{H}(D_{\mathcal{B}}) = \mathcal{H}(D)$, meaning the hashes match and equal the canonical hash H. □

Principle 2 (Canonical Trajectory Invariance Principle)**.** *Within a fixed set of trust boundaries, the canonical trajectory is invariant under any further refinements of the numerical procedure that stay within those boundaries. More concretely: suppose we have trust anchors (P, I, Z) defining a canonical trajectory T with output D and hash H. Now consider any modification to the numerical process that (i) does not violate the boundaries and (ii) potentially improves accuracy within those boundaries (for example, using higher internal precision than required, or a smaller step size than required, or an alternative but mathematically equivalent integrator that still respects the error tolerance). Then the resulting trajectory T' will coincide with T in the sense that after serialization (with the same Z) it will produce the same output D and hash H. In other words, the canonical trajectory T is the unique fixed point: refining the numerical parameters further does not change the trajectory within the agreed resolution. Moreover, if one does enlarge the trust boundaries (e.g., requiring even higher precision), one obtains a new canonical trajectory T^+ that converges toward T as the boundaries tighten, and the hashes remain invariant up to the point of change in least significant digits.*

Proof Sketch: There are two parts to this principle: invariance under allowed changes, and convergence under tightening. *Invariance under allowed changes:* If a change is within the trust boundaries, it means we have not actually changed the defined numerical task. For example, if the integrator boundary says "step size $h = 0.001$ or smaller," then using 0.001 or 0.0005 are both allowed. But all allowed methods are supposed to produce the same serialized output. Why? Because if a smaller step is used but one still rounds the output to the given format, then any differences introduced by the smaller step are below the rounding threshold. By definition, the serialization boundary Z truncates any differences smaller than a certain δ in each value. A stricter integrator (smaller step) would produce a trajectory closer to the true solution, but since the original step already produced results accurate to $\delta/2$, the new results differ by less than δ; hence after rounding they yield the same output data. More formally, assume the original integrator had local error $O(h^p)$ and it met the global error tolerance such that the output error was below $\delta/2$ in each value. If a new integrator uses a step $h/2$, its error might be $\sim O((h/2)^p) \approx \frac{1}{2^p} O(h^p)$, which is even smaller, say $< \delta/4$. Both trajectories T and T' are within $\delta/2$ of the exact solution at output points. Thus, the difference between T and T' is at most $\delta/2 + \delta/4 < \delta$. Rounding both to nearest δ (or 12 decimal places, etc.) will yield the same result. In essence, T was already at the resolution limit of the output; further improvements are invisible in the output space defined by Z. This argument can be extended to any other refinement: higher internal precision beyond P yields differences smaller than what P (the original precision) could produce, hence they vanish upon rounding to the original precision's output format (if the output format was aligned with the original precision). Using a different but higher-order integrator is trickier, but if it's within tolerance, then by definition it should not deviate beyond the allowed error band of the original integrator. As long as both produce outputs within the $\pm\delta/2$ band of the exact solution (and δ is the rounding unit), they will produce identical rounded output. Therefore, the canonical output D doesn't change. Consequently, H remains the same. *Convergence*

under tightening: Now consider if we change the trust boundaries themselves to be stricter – say require 2× higher precision or 2× smaller step. We get a new canonical trajectory T^+ and output D^+. This new trajectory will generally be "closer" to the true continuum solution. For a well-behaved physical problem (assuming it's stable and convergent), as we tighten these parameters arbitrarily, T^+ will converge to the exact solution trajectory (in so far as numerical analysis guarantees convergence for the given scheme). More relevantly, there will come a point beyond which tightening further does not change certain digits of the output. For example, if going from 16-digit to 32-digit precision changes the trajectory at the 10^{-14} level, and we are rounding output at 10^{-12}, then beyond 32-digit, nothing changes up to 10^{-12}. Thus, there is an invariant hash beyond a certain threshold. Practically, one would increase precision until the hash of the output stops changing – that hash corresponds to a trajectory stabilized to the output resolution. The principle asserts that the originally chosen trust boundaries, if chosen such that errors are below output resolution, already achieve that stability. If not, one might revise the trust boundaries (hence a new version of trust dataset) until invariance is observed. □

5 Trust Boundaries and Falsifiability

The concept of *trust boundaries* provides a clear demarcation of the regime in which numerical results are considered reliable and comparable. By strictly specifying the precision, integrator, and serialization parameters, we define a "safe zone" in which trust in reproducibility is guaranteed. So long as results are obtained within these boundaries, any disagreement between independent simulations is highly significant. In fact, a hash mismatch under identical trust anchors is treated as a falsification of the computational result: it indicates that at least one simulation is flawed (either through a bug or because it did not truly meet the stated conditions). Thus, trust boundaries turn reproducibility into a binary criterion—either a result can be reproduced within the trusted regime, or it cannot, in which case the discrepancy must be resolved before the result can be trusted.

In this way, the Trust Framework introduces a Popperian falsifiability test for computational science. If two groups integrate the same equations with the same specified (P, I, Z) and get different outcomes, it is analogous to two experimentalists measuring the same quantity and obtaining conflicting results: at least one of them is wrong, or some hidden condition is not controlled. The formal structure (the trust anchors and boundaries) ensures that there is *no ambiguous middle ground*: under the framework, one should not dismiss a simulation discrepancy as merely a result of "different numerical choices," because all key choices have been standardized. Either the simulations match, or a trust boundary has been crossed or violated. This elevates disagreements to actionable evidence of a problem. In practice, when a mismatch occurs, the researchers would investigate whether one implementation failed to follow the protocol (e.g., used insufficient precision or an incorrect algorithm). If not, the mismatch could point to a deeper issue, such as an undiscovered software bug or an inconsistency in the model itself.

It is important to note that the Trust Framework's falsifiability primarily addresses numerical correctness, not physical validity. A successful cross-verification (matching hashes) does not prove that the simulation's prediction is true in nature; it only proves that the prediction is a consistent solution of the given model. However, by securing this consistency, we set the stage for meaningful physical tests. Once a result is trust-verified computationally, any subsequent disagreement between simulation and experiment is a direct challenge to the

model (physical falsification) rather than a question of numerical error. In other words, the Trust Framework cleanly separates numerical uncertainty from scientific uncertainty. We first eliminate (or strictly control) the former, so that if a simulation fails to match reality, one can confidently attribute it to the physics. This is particularly powerful when comparing different theoretical models: if two competing models are both implemented as trust datasets for the same scenario, their trajectories might be directly compared. If they yield clearly different predictions beyond known uncertainties, then one model will be falsified when compared to nature, with the assurance that the discrepancy isn't due to sloppy numerics.

In summary, trust boundaries delineate the domain where computational experiments are reproducible and thus falsifiable. They enforce a rigorous standard: so long as one stays within the specified boundaries, any independent replication should succeed (and any failure signals a serious issue). The existence of these well-defined boundaries turns reproducibility from a qualitative goal into a quantitative test, and it provides a basis for treating computational findings with a rigor comparable to experimental findings.

6 Stability and Numerical Error

The ability to reproduce a trajectory across independent simulations is intimately tied to the dynamical stability of the system and the control of numerical errors. We introduced σ-stability (Definition **??**) as a measure of how robust a trajectory is to small perturbations. Intuitively, if a system is *unstable* or chaotic, tiny differences (from rounding error or slightly different initial data) will grow exponentially, eventually exceeding any fixed σ and leading to divergent trajectories. Conversely, if a system is *stable* (e.g., dissipative or linear with negative feedback), small perturbations may damp out or stay bounded, making reproducibility much easier.

Consider a chaotic system with Lyapunov exponent $\lambda > 0$. Two trajectories starting with an initial difference Δx_0 will typically diverge at a rate $|\Delta x(t)| \sim |\Delta x_0| e^{\lambda t}$, until nonlinear saturation. This means that if we want two runs to remain within a tolerance σ for time T, the initial numerical differences must be on the order of $e^{-\lambda T}\sigma$ or smaller. In practical terms, to achieve σ-stability over a long integration time T, one must greatly reduce numerical error (perhaps using extremely high precision and small step sizes) so that the inevitable round-off and truncation errors remain exponentially small compared to the system's divergence rate. This is precisely what the Trust Framework facilitates: by tightening the precision and integrator boundaries, we can push Δx_0 (the effective perturbation introduced by numerics at each step) below the threshold required for a desired σ. In effect, the framework leverages numerical rigor (arbitrary precision, etc.) to counteract chaos.

The theoretical justification for this approach comes from the concept of *shadowing* in dynamical systems theory. The shadowing lemma states that for many hyperbolic chaotic systems, a numerical trajectory that stays sufficiently close to some true trajectory will have a nearby exact trajectory that it tracks for an interval of time [**?**]. By minimizing numerical errors, we ensure the computed trajectory shadows an actual trajectory of the differential equations for the duration of interest. The Trust Equivalence Principle then guarantees that all implementations are following the same shadowed trajectory. Of course, if one attempts to integrate indefinitely far into the future, even extremely small errors can eventually grow; thus there is always a time horizon beyond which trajectories decohere. But the framework allows us to extend this horizon as far as needed by choosing appropriate trust boundaries (or, if necessary, explicitly acknowledging a limit on $T_{\max}$ beyond which results are not trusted

without further refinements).

In contrast, for systems that are not chaotic (for example, a damped harmonic oscillator, or the linear bicycle model in Case 1), double precision arithmetic and modest step sizes might be sufficient to achieve σ values on the order of machine epsilon for very long times, because perturbations do not grow significantly. In such cases, the trust boundaries needed for reproducibility can be relatively loose and still guarantee matching results. Indeed, our bicycle stability case study required only standard double precision and an eigen-solver to produce identical stability outcomes across platforms, since the problem is well-conditioned and not sensitive to minute computational differences.

One must also consider systems with stochastic elements. In a purely stochastic simulation (e.g., Monte Carlo or molecular dynamics with random forces), reproducibility of a single trajectory requires controlling the random number generation (for instance, using a fixed random seed) so that independent runs produce the same sequence of random events. The Trust Framework can accommodate this by treating the random seed as part of the trust anchors (within the model specification) and thereby making the simulation pseudo-deterministic. However, doing so means we are verifying one particular realization of the stochastic process. If the quantity of interest is an ensemble-average or probabilistic outcome, a different approach (beyond the current scope of the framework) would be required to define reproducibility of distributions. In our Case 3 (arrow-of-time toy model), we address this by fixing the random seed to obtain a representative trajectory, while acknowledging that extending the trust concept to statistical outcomes is an area for future exploration.

In summary, the Trust Framework compels us to analyze the stability of our simulations and adjust numerical settings accordingly. For a given system and time span, σ-stability provides a target for how indistinguishable runs should remain. If the system is chaotic, achieving a tiny σ demands high precision and careful integration; if the system is stable, the requirements are more modest. By empirically testing variations (e.g., halving the time step or using higher precision) and observing whether the results change beyond σ, one can validate that the chosen trust boundaries are appropriate. This process ensures that the numerical error is effectively under control and that the simulation's inherent dynamics (not numerical noise) dominate the trajectory.

7 Serialization and Hash Anchors

A crucial but sometimes underappreciated aspect of reproducibility is ensuring that two matching trajectories are represented in exactly the same way. Even if two simulations follow the same dynamics and produce the same sequence of states, a discrepancy in how the results are written to disk (different formatting, ordering, or precision) can lead to binary differences. The *serialization boundary* in our framework eliminates this ambiguity by rigidly specifying the output format. By agreeing in advance on things like the time points for output, the number format (scientific notation with a fixed number of significant digits, etc.), sorting of data entries, and even file encoding details, we guarantee that identical trajectories yield identical files.

In practice, implementing the serialization boundary requires careful attention to potential platform and language differences. For example, one programming environment might by default print floating-point numbers as `0.123456789012345E+01` whereas another prints `1.23456789012345e+00`, and even slight differences (like a different capitalization of the exponent or a different rounding of the last digit) would result in non-identical files. The

solution is to explicitly format all output according to a contract. In our trust datasets, we use routines that format floating-point numbers to a specified number of digits and enforce consistent notation. We also ensure that data structures (like JSON or CSV files) have a deterministic ordering of entries. For instance, if output is in JSON format, we require that arrays and object keys are always written in the same order (alphabetical or a specified sequence) and that no extraneous whitespace or variation in syntax is allowed. By constraining these degrees of freedom, we make the serialization process itself deterministic.

The cryptographic hash then acts as the final arbiter of equivalence for the output files. We choose a hash function such as SHA-256, which produces a 256-bit fingerprint of the file. A single-bit difference in the file (for example, one digit being different in one output) will completely change the SHA-256 hash value, thanks to the avalanche effect in cryptographic hashes. Conversely, if two files have the same SHA-256 hash, it is overwhelmingly likely (with probability $\approx 2^{-256}$ of a false positive) that the files are exactly identical bit-for-bit. In practice, the chance of an accidental hash collision is so astronomically low that we treat a matching hash as proof of matching data. Using such hashes is standard in software verification and data integrity applications.

One advantage of hash-based comparison is efficiency and convenience. Rather than manually inspecting large output files or doing a line-by-line comparison, one can simply compute the hash of the file and compare it to the published hash H. If the hashes match, the files are identical; if not, some difference exists. The hash thus serves as a concise *trust anchor* for the result: it condenses perhaps megabytes of trajectory data into a short string (e.g., 64 hexadecimal characters for SHA-256) that can be listed in a paper or on a dataset webpage. Anyone who recomputes the trajectory can verify agreement by recomputing and checking this string.

In our framework, the hash is typically recorded in a manifest along with context (e.g., file name, dataset version). For example, a trust dataset might include a 256*hash should be a specific value say,*

```
cc59025c07830f49b3d8aa21779b6b940d4feeed9b877f75d8d4419c83c03
```

8Limitations and Operational Considerations

While the Trust Framework provides a structured approach to reproducibility, it has limitations and requires operational choices that users must consider.

A primary limitation is scalability. High-precision computations (e.g., 80-digit arithmetic) are computationally expensive and best suited for small systems or short time windows. For large-scale simulations like climate models or high-resolution PDEs, full bitwise agreement may be impractical. In such cases, the framework can be applied to subsystems, diagnostics, or shorter segments. Future extensions could include streaming hashes (e.g., rolling hashes over time windows or Merkle trees) for large outputs, but these are beyond v1.0.

Another consideration is choosing the serialization precisionδ(the rounding unit in the serialization boundary).A heuristic is to setδlarger than estimated numerical errors but small relative to physical scales of interest. For example, if physical distances are O(1) and errors are controlled to$<$ 10-8, rounding to 10 -12 ensures physical features are resolved while sub-10-8 differences are rounded away. Users can empirically validate$\delta$by tightening boundaries and checking if hashes stabilize.

Performance tradeoffs are also key: higher precision extends the reliable time horizon in chaotic systems but scales runtime exponentially. For stable systems, double precision often suffices. We recommend starting with modest boundaries and tightening as needed based on σ-stability tests.

The framework assumes deterministic environments; stochastic systems require fixed seeds for single trajectories, but ensemble statistics need future extensions. Tools like containers (Docker) and CI pipelines are natural complements for automation, though not required in v1.0.

In summary, the Trust Framework is scoped to low-to-medium dimensional deterministic or pseudo-deterministic systems. Operational decisions like δ and precision should balance accuracy, runtime, and physical relevance.

9 Dataset Architecture

We have repeatedly referred to *trust datasets* as the vehicle for sharing and verifying results. Here we describe how a trust dataset is structured and what it contains. The goal is to package everything needed to independently reproduce the computational experiment and to validate the outcome via hash comparison.

At a high level, a trust dataset includes:

- **Specifications and documentation:** A detailed description of the model and trust anchors (often provided as a PDF or README file). This document states the equations being solved, the parameter values, initial conditions, and the exact trust boundaries (precision, integrator, serialization format) that were used. Essentially, this is a formal record of the experiment's design, much like the methods section of a paper. In many cases, a significant portion of this text overlaps with the descriptions found in an accompanying whitepaper (such as this one), and the dataset documentation might even include or reference the whitepaper.

- **Canonical data output:** The primary results of the simulation, saved in the canonical serialization format. For example, in our case studies this could be a file like `eigenvalues` *benchmark.csv*

- `Hash manifest: A file (or section of the documentation) that lists each output data file and its hash (e.g., the SHA-256 value). This manifest is the key for verification: after recomputing, one can compare the new hashes to those in the manifest. The manifest may also include hashes of any important input files or configuration files, to ensure those were not altered.`

- `Verification and analysis scripts: Code that can be used to either recompute the results or at least verify them. In some cases, this might be a Jupyter notebook or a Python/Matlab script that reads the provided data and, for instance, regenerates the plots from the associated paper or checks conservation laws. Ideally, it may also contain an independent re-computation (for example, using a different algorithm or library) to cross-check the results within the dataset itself. At minimum, these scripts demonstrate how to load the data and compute the hashes, making it straightforward for a user to perform the verification.`

- `Optional reference code: While not strictly required (since the idea is that independent implementations might be used), authors often include the actual source code or notebooks they used to generate the canonical data. This can be helpful for transparency and for users who want to understand`

or modify the original simulation. However, the trust framework does not rely on this code being used by others; it is provided as a convenience. The core requirement is that the code, if run, produces the included canonical data and hashes.

- Figures or summary plots: Many trust datasets include a few illustrative figures (e.g., a plot of the trajectory or a stability diagram) as quick visual evidence that the results look as expected. These also serve as a check; if someone's reproduction is off, they might see it visually even if the hash didn't match. For example, the bicycle dataset might include a graph of eigenvalues vs. speed showing where it's stable, and the three-body dataset might include an image of the figure-8 orbit. These help users quickly confirm \this looks right" and also act as a sanity check (if someone's reproduction looks qualitatively different, they know something is wrong even before computing the hash).

To ensure longevity, trust datasets are typically archived in public data repositories such as Zenodo. Each dataset is given a DOI (digital object identifier), and often versions are tracked: for instance, a dataset might be published as version 1.0 (with a DOI specific to that version). If later a minor error is found or improvements are made, a version 1.1 or 2.0 can be published with a new DOI, while still linking to the original (Zenodo allows a concept of a DOI for the "latest version" and DOIs for each specific version). By versioning the dataset, the framework accommodates updates while preserving the exact records of what was done initially. In our case, this whitepaper and the associated trust datasets are labeled "v1.0" to indicate the initial release under the Trust Framework for Computational Physics. Future updates (say, to correct a mistake in the data or to tighten a trust boundary upon discovering a need for higher precision) would result in new dataset versions (and possibly a revised paper version), clearly documented as such.

It's also important that users cite these datasets when they make use of them. Just as one would cite a traditional paper, we envision citing the dataset DOI and the whitepaper together, so that credit is given for the effort of making a reproducible dataset. The dataset's documentation typically includes a suggested citation format.

In essence, a trust dataset is the practical implementation of the framework's principles. It is the container that holds the "evidence" of a computational result and all auxiliary information needed to scrutinize and regenerate that evidence. By examining the contents of a trust dataset, an independent researcher should be able to understand exactly what was done and to repeat the experiment (on their own system, possibly with their own code) to see if they obtain the same outcomes. If they do, the hashes (and science) agree; if they do not, the dataset provides the clues (specs and code) to investigate where the discrepancy may have arisen.

Summary of Referenced Trust Datasets

This whitepaper draws on three previously archived trust datasets:

- **Trust Dataset 1 (Bicycle Stability):** DOI: 10.5281/zenodo.17634914.

- **Trust Dataset 2 (Three-Body Orbits)**: DOI: 10.5281/zenodo.17635887.

- **Trust Dataset 3 (Arrow-of-Time Model)**: DOI: 10.5281/zenodo.17652928.

10 Case Studies

To illustrate the Trust Framework in action, we present three case studies drawn from different areas of physics. Each case corresponds to a trust dataset that has been openly archived (with a Zenodo DOI) and demonstrates how the framework handles a particular type of problem. The cases increase in complexity from a simple linear stability analysis to a chaotic three-body system to a speculative diffusion model.

Case 1: Linear Mechanical Stability – Bicycle Self-Stability

Our first case study involves the classical problem of an unridden bicycle's self-stability. Meijaard*et al.*(2007) derived the linearized equations of motion for a bicycle (the canonical "Whipple" bicycle model) and identified the velocity range over which an unridden bicycle is self-stable. In other words, there is a finite band of forward speeds in which the bicycle can automatically steer itself upright after a perturbation, without rider input; at speeds below or above this band, the bicycle becomes unstable and will tip over unless actively controlled.

The trust dataset for this case reproduces the benchmark results of Meijaard*et al.* by fixing the model parameters and solving the linear stability eigenvalue problem as a function of forward speed. The trust anchors include using double-precision arithmetic and a specific generalized eigen-decomposition algorithm to compute the linearized dynamics matrix eigenvalues.The output of the dataset is a table of eigenvalues (or equivalently, damping rates and oscillation frequencies) vs. speed, along with identification of the critical speeds at which stability is lost or regained. Independent implementations (e.g., one group's MATLAB script and another's Python code) that use the same model and numerical approach will yield identical eigenvalue curves and thus identical hashes for this table. The dataset confirms the known result that the example bicycle is self-stable only within an intermediate range of speeds (on the order of a few m/s).In our simulation, for instance, the bicycle's weave and capsize modes are damped for speeds between 3.6500000000000004 and 5.300000000000001 m/s (34 consecutive speeds satisfying the stability criterion), indicating self-stability in that range, and become unstable outside it. Because this problem is linear and well-conditioned, reproducibility is relatively straightforward – even standard double precision suffices to produce matching results across platforms. Thus, this case serves as a baseline test of the framework's efficacy. The trust dataset (Zenodo DOI: 10.5281/zenodo.17634914) includes the model specification, eigenvalue data, and a verification script.Any failure to reproduce these results exactly would immediately suggest a violation of the conditions (e.g., a different coordinate convention or a numerical precision issue).

Case 2: Nonlinear Gravitational Dynamics – Periodic Three-Body Orbits

Our second case study tackles a famous problem in nonlinear dynamics:the Newtonian three-body problem. While generic three-body motion is chaotic, there exist special initial conditions that result in periodic orbits. The "figure-8" three-body choreography discovered by Moore and proven to exist by Chenciner and Montgomery is a celebrated example, involving three equal masses chasing each other around a planar figure-eight trajectory. Subsequently, researchers have found many more intricate periodic orbits (nicknamed the "butterfly," "moth," and so on) through numerical searches. These orbits are delicate: a tiny perturbation or numerical error will cause the bodies to drift off the periodic path (since the surrounding dynamics are chaotic with positive Lyapunov exponents).

In this case, the trust dataset (Zenodo DOI: 10.5281/zenodo.17635887) focuses on reproducing a set of known periodic three-body orbits with very high precision, effectively eliminating numerical drift over the simulated time.The trust anchors specify the use of arbitrary-precision arithmetic (we employed 80-digit decimal precision) and a symplectic or high-order integrator with a very small step size to control local truncation error.For the figure-8 orbit, for example, our canonical simulation follows the system through one full period (T= 6.32591398 in suitably normalized units) and returns each body to its starting position with an accumulated error on the order of 7.544683316874829e–08. The trajectory data – positions and velocities at many points over the period – is recorded to 12 decimal places, so this error is far below the output resolution. An independent implementation that adheres to the same precision and method (for instance, one team might use a Python library for arbitrary precision ODE integration while another writes a C++ code with GMP multi-precision) will produce the exact same sequence of states at the output times, as verified by matching hashes of the trajectory files.

This case demonstrates the framework's value: even a chaotic system can be made reproducible if one pushes numerical errors below the shadowing threshold.In standard double precision, two simulations of the figure-8 orbit would typically dephase after a short time – their bodies would trace the same curve initially but gradually fall out of sync due to slight numerical differences.With 80-digit precision and stringent error control, however, any divergence is suppressed below the 10^{-12} output tolerance for the duration of the run; all observers see essentially the same trajectory. We also ensure our results match those in the literature: for example, we compute the figure-8 orbit's period to high accuracy (matching Moore's and others' published values to many significant figures) and confirm its choreographic symmetry (all three masses follow the same path with 120° phase offsets). The trust dataset includes additional checks such as energy and momentum conservation – with 80-digit precision, energy drift per period is negligible (2.4008838248938523e–33), whereas a double-precision simulation might show a drift around 10^{-14}.Such comparisons illustrate how pushing into the "trusted" regime bridges the gap between numerical integration and the theoretical existence of an orbit.In practical terms, this trust dataset can serve as a benchmark:anyone developing a new integrator or code for the three-body problem can test it by attempting to reproduce these orbital trajectories.Any discrepancy (hash mismatch) would pinpoint an implementation issue or insufficient numerical accuracy. The canonical summary includes details for additional orbits:Butterfly I (I.A.1) with period 6.2356, energy drift 4.341138083617139e-29, closure error 0.0018286927833090138, SHA-256 e8905fd4f6df1211c69b64221be54052fa4e253238725476766e0b3e96c39ba5; Moth I (I.B.1) with period 14.8939, energy drift 5.520769246398254e-31, closure error 0.00036082474667521665, SHA-256 cf26581e2d62c9db6d4a26bac7ccc5f62bea81a243289bfd65457eb771fe8783; Yin-Yang

I (II.C.2a) with period 17.3284, energy drift 2.5074428910107846e-29, closure error 0.0015154295086739296,
SHA-256 c0e9036a29b7268ff816bcf889fec6713e28610d13ef657b85d506acf566a380.

Case 3: Curvature-Dependent Diffusion – A Gravitational Arrow-of-Time Toy Model

Our final case study is more conceptual, illustrating that the Trust Framework can even be applied to stochastic or speculative models outside conventional physics. We constructed a toy model to mimic a possible "gravitational arrow of time" scenario, where entropy-like behavior arises from gravitational clumping. The model is a 1D diffusion process in an asymmetric double-well potential with a curvature-dependent diffusion coefficient $D(x) = D_0(1+\alpha|V''(x)|)$, where diffusion strengthens in high-curvature regions, and a drift term from the potential gradient. This draws from ideas in gravitational entropy, where time's direction emerges from complexity growth in N-body systems without special initial conditions.

The trust dataset for this model (Zenodo DOI: 10.5281/zenodo.17652928) defines the rules of the stochastic process, including fixing the random number generator seed to ensure reproducibility of the random events. We discretize the evolution equation with finite differences, periodic boundaries, and mass renormalization, producing hash-verified JSON trajectories and summaries with diagnostics like entropy ranges.As expected, the probability density relaxes asymmetrically, illustrating emergent time asymmetry. Because we fix the random seed and use the same pseudorandom sequence for every run, the trajectory and diagnostics are deterministic outputs that can be hashed.Any other implementation that uses the same rules and random seed (we supply it as part of the trust anchors) produces bitwise identical data and thus the same hash.In our dataset, two different codes (one in Python, one in C++) were used to simulate the process, and they yielded matching entropy curves, strengthening confidence that the effect is not an artifact of one particular code.

This case is admittedly outside mainstream physics and serves mainly to stress-test the framework's generality.It shows that even when dealing with randomness, one can enforce determinism (via fixed seeds) to apply a hash-based equivalence test.It also highlights the distinction between verifying a single trajectory and verifying statistical claims: our trust verification covers the former (exact reproducibility of one random realization), but not the latter (which would require agreement on distributions across many runs, a more complex notion of reproducibility). Nevertheless, by casting a qualitative idea (time's arrow from gravitational-like bias) into a concrete simulation that anyone can rerun, we make the idea falsifiable at least in a computational sense.If someone doubted our results, they could reimplement the biased diffusion; a mismatch with our hash would expose an implementation discrepancy, while a match would confirm that, given the assumptions, the arrow-of-time behavior is indeed obtained.We deliberately avoid overinterpreting this toy model's significance – it's a proof-of-concept rather than an astrophysical statement – and we label any speculation clearly as such.The Trust Framework's discipline encourages this clarity:since every assertion can be checked, there is less temptation to overclaim.

In terms of limitations, this case study underscores that the Trust Framework is most straightforward when randomness is made repeatable. In real-world applications (like climate simulations or Monte Carlo particle physics), fixing seeds may be impractical or insufficient to address reproducibility concerns.There, one might need to extend the framework with statistical measures of reproducibility (e.g., verifying that two sequences of random num-

bers produce statistically indistinguishable outcomes). Those considerations are beyond our present scope, but they represent a frontier for future work. For now, the important takeaway is that even a rather abstract scenario can be packaged into a trust dataset, allowing the community to examine, test, and debate it on equal footing.

11 Discussion

The case studies demonstrate that the Trust Framework is capable of elevating computational experiments to a new level of transparency and rigor. We have shown examples from linear, nonlinear, and even stochastic contexts where results that could have been contentious or irreproducible are instead turned into shared, verifiable knowledge.We now reflect on the broader implications, limitations, and prospects of this approach.

One philosophical point worth highlighting is the emergence of*trajectory-level signatures* in our framework.When all independent observers (simulations) agree on a trajectory's hash, that hash becomes an objective property of the system under the given conditions – a signature that transcends any particular implementation. This is conceptually similar to how in physics certain quantities are the same no matter who measures them, though hashes are protocol-specific rather than universal.

Practically speaking, the framework pushes us toward a culture of checking and cross-checking.If a result comes out that is surprising or counter-intuitive, the first response in a trust-based workflow is to verify it independently. This is similar to experimental science, where a shocking claim would prompt replication attempts in other labs. In computational work, this hasn't been standard practice – often due to difficulties in replication. But with trust datasets, one can much more easily attempt to reproduce others' findings. We suspect that, over time, this could improve the*collective memory*of the field: important results could be preserved in executable form, not just as figures in papers. Re-implementations years later (say, on new hardware or using new languages) could be validated against the original data, ensuring that insights are not lost or distorted over time.

We should also consider the framework's current limitations. One clear limitation is scalability.Not every simulation can be pushed to 80-digit precision or ultra-fine resolution – computational resources are finite. For extremely large or complex simulations (climate models, high-resolution turbulence, etc.), a full bitwise agreement may be impractical to attain. In such cases, the framework might need to be relaxed or supplemented with other methods:for example, one might verify smaller components of the model or shorter segments of a long simulation, or use interval arithmetic to bound differences in lieu of hashing full outputs. Another limitation is that we have focused on deterministic equivalence. In inherently stochastic modeling, insisting on bitwise identical outputs for one random sequence is a limited notion of reproducibility; what one really cares about are distributional or ensemble properties. Adapting the framework to cover that – perhaps by comparing statistical summaries or requiring consistency of random number streams across implementations – is an open problem.

A further challenge is integrating trust-verification into the publication and peer-review process.If journals expect authors to provide trust datasets, what standards and training are needed? Will reviewers rerun the simulations or just examine the procedures? Perhaps new roles (like "reproducibility reviewers") could emerge. Encouragingly, the community is already moving in this direction with reproducibility checklists and data/code availability requirements. Our proposal provides concrete criteria (hash matching) which could simplify

the task: a reviewer doesn't necessarily need to examine every line of code, but can focus on whether the claimed result has been independently reproduced (perhaps by a dedicated verification team or as part of a multi-group collaboration). Over time, as these practices become routine, we might see a virtuous cycle: researchers know their results will be checked, so they are more careful and thorough from the outset, leading to fewer false leads and retractions.

Finally, let us briefly speculate on a future where trust-verified simulations are the norm. One could imagine a shared repository – an analog of the Particle Data Group in high-energy physics – that maintains a list of important computational results, along with their hash signatures and dataset DOIs. When someone solves a classic problem (say, a new solution to Navier–Stokes in some regime), they contribute the trust dataset to this repository. Others can then reliably use those results, or even treat them as "ground truth" for testing new methods. Such a repository might also record instances of mismatch: if two groups try to simulate the same scenario and get different hashes, that would be flagged as requiring investigation. In this sense, the framework could facilitate a more systematic mapping of the landscape of computational physics knowledge – including its uncertainties and boundaries.

In summary, while version 1.0 of the Trust Framework is just a beginning, it demonstrates a tangible path toward more reproducible and trustworthy computational science. The case studies show that, with some effort, one can transform numerical experiments into durable, sharable assets. The potential rewards in terms of credibility and efficiency in research are significant. By focusing on clear, modest claims and providing all the means for others to verify them, we avoid grandstanding and instead contribute incrementally but solidly to the edifice of science. We view this as very much in the spirit of Popper and Feynman – an ethos of "utter honesty" about computational results, where the harshest tests are not only welcomed but facilitated. The Trust Framework, in our opinion, provides one viable implementation of that ethos.

12 Conclusion and Future Work

In this paper, we introduced the Trust Framework for Computational Physics v1.0, a formal architecture for ensuring that computational results are reproducible and verifiable. We set forth rigorous definitions of concepts like trust anchors and boundaries that precisely delineate what it means to perform the "same" computational experiment. We established core principles guaranteeing that within these boundaries, all correct simulations will agree (Trust Equivalence) and that further refinement of numerical parameters will not change the outcome (Canonical Trajectory Invariance). We demonstrated the framework on three diverse case studies, turning previously challenging simulations into shareable trust datasets with cryptographic hash signatures.

The immediate payoff of this framework is a higher degree of confidence in computational findings. When a result is accompanied by a trust dataset, one does not have to take it on faith or personal expertise – any researcher can independently verify it by reproducing the hashed output. This is a marked step toward treating computational science with the same empirical rigor as experimental science. Moreover, by making the "hidden variables" of simulations (like precision and algorithmic details) explicit and standardized, the framework reduces the incidence of irreproducible outcomes and eases the burden of starting new work from old results.

Looking ahead, there are several directions to extend this work. One direction is to develop better tools and software infrastructure to automate the creation of trust datasets.

Integrating the necessary instrumentation (hashing, output formatting, etc.) into widely-used simulation libraries could greatly lower the barrier. Another direction is handling systems at scale: exploring hybrid approaches where one can verify parts of a large simulation or use statistical equivalence tests when bitwise equivalence is out of reach. There is also room to integrate more sophisticated formal verification methods (e.g., proof assistants or certified compilers) with our approach, to cover cases where floating-point nondeterminism or parallelism might otherwise break reproducibility.

We also see an opportunity to connect with experimental science. While directly hashing experimental data is generally infeasible (due to noise and continuous variation), the mentality of precisely specifying "trust conditions" carries over. For instance, a detailed experimental protocol with error bounds plays a role analogous to our trust anchors and boundaries. One could imagine simulation and experiment working hand-in-hand:a simulation's trust dataset could be used to generate predicted measurement hashes (within tolerances), and any experimental result falling outside those tolerances would indicate new physics or a flaw in the model.

In conclusion, we have taken a step toward solidifying the foundation of computational physics. By formalizing how to make simulations reproducible and by providing examples of success, we hope to inspire further adoption and refinement of these practices.The Trust Framework v1.0 is not the final word – it is a starting point that others can build upon. As more researchers engage with these ideas, we anticipate a future where sharing a trust dataset becomes as natural as sharing a graph, and where computational results become lasting contributions that need not be constantly reconfirmed. In that future, computational physics will be more reliable, more collaborative, and ultimately more impactful in its contributions to our understanding of the universe.

Acknowledgments

The author acknowledges the open computational physics community whose published trust datasets enabled systematic cross-verification and reproducibility testing.

References

[1] M. Baker, "1,500 scientists lift the lid on reproducibility," *Nature* **533**, 452–454 (2016).

[2] J. P. A. Ioannidis, "Why most published research findings are false," *PLoS Med.* **2**(8), e124 (2005).

[3] M. Hutson, "Artificial intelligence faces reproducibility crisis," *Science* **359**, 725–726 (2018).

[4] V. Stodden *et al.*, "Enabling the verification of computational results: An empirical evaluation of computational reproducibility," in *Proc. 1st Int. Workshop on Practical Reproducible Evaluation of Computer Systems (P-RECS'18)*, ACM, 2018.

[5] T. E. Nazaré *et al.*, "A Note on the Reproducibility of Chaos Simulation," *Entropy* **22**(9), 953 (2020).

[6] E. Ott, *Chaos in Dynamical Systems*, 2nd ed. (Cambridge Univ. Press, 2002), pp. 18–19.

[7] J. P. Meijaard *et al.*, "Linearized dynamics equations for the balance and steer of a bicycle: a benchmark and review," *Proc. R. Soc. A* **463**, 1955–1982 (2007).

[8] C. Moore, "Braids in classical dynamics: A periodically time-symmetric three-body orbit," *Phys. Rev. Lett.* **70**, 3675–3679 (1993).

[9] A. Chenciner and R. Montgomery, "A remarkable periodic solution of the three-body problem in the case of equal masses," *Ann. Math.* **152**, 881–901 (2000).

[10] M. Šuvakov and V. Dmitrašinović, "Three classes of Newtonian three-body planar periodic orbits," *Phys. Rev. Lett.* **110**, 114301 (2013).

[11] J. Vanderbei, "New orbits for the n-body problem," *Ann. N. Y. Acad. Sci.* **1017**, 422–433 (2004).

[12] B. Brik et al., "Formal Definition and Implementation of Reproducibility Tenets for Computational Astronomy Workflows," arXiv:2406.01146 [cs.DC] (2024).

[13] R. E. Moore, R. B. Kearfott, and M. J. Cloud, *Introduction to Interval Analysis* (SIAM, 2009).

[14] S. Yu. Pilyugin, *Shadowing in Dynamical Systems* (Springer, 1999).

[15] J. Barbour et al., "Identification of a Gravitational Arrow of Time," *Phys. Rev. Lett.* **113**, 181101 (2014).

[16] Bradley C. Peter, "Bicycle Stability Trust Dataset v1.0," Zenodo (2025). DOI: 10.5281/zenodo.17634914.

[17] Bradley C. Peter, "Periodic Three-Body Orbits Trust Dataset v1.0," Zenodo (2025). DOI: 10.5281/zenodo.17635887.

[18] Bradley C. Peter, "Curvature-Dependent Diffusion Arrow-of-Time Trust Dataset v1.0," Zenodo (2025). DOI: 10.5281/zenodo.17652928.