PROJECT REPORT

Minimal Physics Simulator

Stability, Performance, and Systems Engineering Analysis

Submitted by

Suriya Sureshkumar
X Ivan Nilash

In partial fulfillment of the requirements for the degree of

Bachelor of Technology

in

Artificial Intelligence and Data Science

Under the Guidance of

Dr. R.Priyanka Pramila

Department of Artificial Intelligence and Data Science

R.M.K Engineering COllege

Kavaraipettai, India

Feb 2026

Contents

List of Figures

Abstract

Simulation systems are essential for modern robotics, reinforcement learning, scientific computing, and high-performance AI systems. While many people use them, most simulation engines are treated like black boxes. There is often little examination of their numerical stability, structural preservation, performance scaling, or real-world limitations.

This project presents the design and development of a minimal physics simulation engine built entirely from first principles using Python and NumPy. The simulator models a one-dimensional harmonic oscillator and moves through five structured phases that gradually increase in technical depth and system complexity.

The study begins with deterministic numerical integration and compares classical ODE solvers, including Explicit Euler, Semi-Implicit Euler, Velocity Verlet, and RK4. Stability is assessed using long-horizon energy drift metrics rather than just checking for explosion detection. This approach shows that symplectic integrators are structurally better for conservative systems.

Next, the project focuses on performance engineering. It scales particle counts to identify interpreter bottlenecks, vectorization benefits, and memory bandwidth limits. Through in-place optimization and precision analysis, the simulator reveals real hardware constraints, achieving over one billion particle updates per second and identifying memory-bound execution scenarios.

The following phases introduce reinforcement learning-style batched rollouts, design for the rollout buffer, analysis of memory issues, and validation of determinism. Finally, the project simulates real-world constraints such as floating-point precision limits, control delays, timing variations, and asynchronous stepping to study their effects on stability and reproducibility.

This work connects numerical analysis, systems engineering, reinforcement learning infrastructure, and control theory. The resulting simulator acts as a minimal physics engine and a laboratory for understanding how mathematical stability, computational performance, and real-world constraints interact in modern AI-driven simulation systems.

Keywords: Numerical Stability, Ordinary Differential Equations, Symplectic Integrators, Energy Drift, Memory Bandwidth, Performance Engineering, Reinforcement Learning Infrastructure, Determinism, Control, Latency, Simulation Systems

1 Introduction

Simulation engines serve as the foundation for many scientific and engineering fields, such as robotics, aerospace modeling, reinforcement learning, and computational physics. Numerical integration methods approximate continuous dynamical systems by using discrete time steps. However, the long-term stability, preservation of structure, and efficiency of these methods rely heavily on how the algorithms are designed and implemented. Even with advanced simulation frameworks, the underlying numerical and systems-level trade-offs are often not thoroughly examined. As simulation tasks grow to millions of time steps and thousands of parallel environments, performance issues and structural instabilities become more important.

This project will build and analyze a basic physics simulator from first principles to study these interactions in a systematic way.

1.1 Problem Statement

Modern simulation systems must meet three often conflicting requirements:

- Numerical stability over long time frames
- High computing efficiency under hardware limits
- Deterministic and reproducible execution

Classical integrators like Explicit Euler, Runge-Kutta methods, and symplectic schemes show different behaviors regarding energy conservation, stability regions, and structural fidelity. Performance scaling adds extra challenges, such as interpreter overhead, memory bandwidth limits, and memory capacity issues.

In reinforcement learning and robotics, more real-world factors, including control latency, timing variability, floating-point precision restrictions, and asynchronous execution, can further affect stability and reproducibility.

The main question this work addresses is:

How do numerical integration methods, memory architecture choices, and real-world timing constraints interact to influence stability, scalability, and determinism in simulation systems?

1.2 Project Scope

This project is focused on designing and evaluating a basic one-dimensional physics simulator that models a harmonic oscillator. It includes:

- Implementing and comparing classical ODE integrators
- Analyzing energy drift over a long period
- Testing throughput as particle counts increase
- Examining memory bandwidth and allocation
- Creating a reinforcement learning-style batched rollout system
- Simulating real-world constraints like precision limits, latency, jitter, and asynchronous stepping

The project does not cover:

- Multi-body collision detection
- Complex 3D rigid-body physics
- GPU kernel optimization
- Distributed systems implementation

The emphasis is on clear concepts, measurable performance analysis, and understanding structural stability.

2 Mathematical Foundations

The development of the Minimal Physics Simulator is based on classical mechanics and numerical analysis. The system is intentionally minimal; it is a one-dimensional harmonic oscillator. However, it captures key features of conservative dynamical systems, making it a good reference for studying numerical stability, energy conservation, and long-term integration behavior.

This section outlines the continuous-time mathematical model, derives its analytical solution, clarifies its energy invariants, and introduces the theoretical framework needed to examine discrete-time numerical approximations.

2.1 The Harmonic Oscillator Model

The system modeled in this project is a classical mass-spring oscillator. Consider a point mass m attached to a linear spring with a stiffness constant k. When the mass moves away from its rest position, the spring applies a restoring force that is proportional to the distance moved and directed back toward the rest position. Hooke's Law describes this relationship:

$$F = -kx$$

The negative sign indicates that the force opposes displacement.

Applying Newton's Second Law:

$$F = m\frac{d^2x}{dt^2}$$

Substituting Hooke's law:

$$m\frac{d^2x}{dt^2} = -kx$$

Rewriting:

$$\frac{d^2x}{dt^2} + \frac{k}{m}x = 0$$

Defining the natural angular frequency:

$$\omega^2 = \frac{k}{m}$$

The governing equation becomes:

$$\frac{d^2x}{dt^2} + \omega^2 x = 0$$

This is a second-order linear homogeneous differential equation with constant coefficients. Reasons for choosing this system include:

- It is solvable using analytical methods.
- It conserves energy.
- It shows bounded oscillatory motion.
- Its eigenvalues are located on the imaginary axis.
- t demonstrates numerical instability right away if not discretized properly.

Although it is simple, this system captures the key properties of conservative Hamiltonian systems, making it a great testbed for comparing integrators.

2.2 Conversion to First-Order System

Numerical integrators typically operate on first-order systems. Therefore, the second-order equation must be reformulated.

Define velocity:

$$v = \frac{dx}{dt}$$

Then:

$$\frac{dv}{dt} = \frac{d^2x}{dt^2}$$

From the governing equation:

$$\frac{dv}{dt} = -\omega^2 x$$

Thus, the system becomes:

$$\frac{dx}{dt} = v$$

$$\frac{dv}{dt} = -\omega^2 x$$

This system can be expressed in vector form:

$$\mathbf{y}(t) = \begin{bmatrix} x(t) \\ v(t) \end{bmatrix}$$

$$\frac{d\mathbf{y}}{dt} = \begin{bmatrix} v \\ -\omega^2 x \end{bmatrix}$$

Or equivalently:

$$\frac{d\mathbf{y}}{dt} = A\mathbf{y}$$

Where:

$$A = \begin{bmatrix} 0 & 1 \\ -\omega^2 & 0 \end{bmatrix}$$

This matrix formulation is critical. It allows us to analyze system stability through eigenvalues and provides a direct bridge to discrete-time update matrices used in numerical integration.

2.3 Analytical Solution

To understand how numerical methods deviate from the true solution, we must first derive the exact continuous solution.

Assume a trial solution:

$$x(t) = e^{\lambda t}$$

Substitute into:

$$\frac{d^2x}{dt^2} + \omega^2 x = 0$$

We obtain:

$$\lambda^2 + \omega^2 = 0$$

Thus:

$$\lambda = \pm i\omega$$

The general solution becomes:

$$x(t) = C_1 \cos(\omega t) + C_2 \sin(\omega t)$$

Velocity:

$$v(t) = -C_1 \omega \sin(\omega t) + C_2 \omega \cos(\omega t)$$

This solution has several important properties:

- Bounded for all time
- Periodic with period $\frac{2\pi}{\omega}$
- Energy constant over time
- No exponential growth or decay

This bounded oscillatory behavior is the baseline against which numerical integrators are evaluated.

If a numerical method produces exponential growth, energy drift, or damping, it is introducing artificial dynamics not present in the true system.

2.4 Energy Conservation in Continuous Systems

The total mechanical energy is the sum of kinetic and potential energy:

$$E = \frac{1}{2}mv^2 + \frac{1}{2}kx^2$$

Substituting $k = m\omega^2$:

$$E = \frac{1}{2}mv^2 + \frac{1}{2}m\omega^2x^2$$

Differentiating with respect to time:

$$\frac{dE}{dt} = mv\frac{dv}{dt} + m\omega^2 x\frac{dx}{dt}$$

Substituting system equations:

$$\frac{dv}{dt} = -\omega^2 x$$

$$\frac{dx}{dt} = v$$

We obtain:

$$\frac{dE}{dt} = mv(-\omega^2 x) + m\omega^2 x(v) = 0$$

Thus:

$$E(t) = \text{constant}$$

Energy conservation is not just a physical property — it is a structural invariant.

In later phases, energy drift becomes our primary diagnostic tool for assessing numerical stability. A method that fails to preserve energy structure may exhibit long-term divergence even if short-term accuracy appears high.

2.5 Stability in Linear Dynamical Systems

Consider the state-space form:

$$\frac{d\mathbf{y}}{dt} = A\mathbf{y}$$

Stability depends on the eigenvalues of the matrix A.

For:

$$A = \begin{bmatrix} 0 & 1 \\ -\omega^2 & 0 \end{bmatrix}$$

Characteristic equation:

$$\lambda^2 + \omega^2 = 0$$

$$\lambda = \pm i\omega$$

Eigenvalues lie on the imaginary axis.

Interpretation:

- Real part equals zero
- No exponential growth
- Pure oscillation
- System is neutrally stable

In continuous time, this means small perturbations remain bounded.
However, numerical integration introduces discrete updates:

$$\mathbf{y}_{n+1} = M\mathbf{y}_n$$

Where M depends on the integration method.
Stability now depends on eigenvalues of M.
If:

$$|\lambda_M| > 1$$

The solution grows exponentially.

This explains why Explicit Euler, when applied to oscillatory systems, leads to artificial energy growth.

Thus, stability in numerical simulation must be analyzed through the spectral radius of the discrete update operator.

2.6 Local vs Global Truncation Error

Numerical integration replaces continuous time with discrete steps of size Δt.

Local truncation error (LTE) is the error made in one step, assuming the previous value is exact. For a method of order p:

$$LTE = O(\Delta t^{p+1})$$

Global truncation error (GTE) is the accumulated error after many steps:

$$GTE = O(\Delta t^p)$$

Examples:

- Explicit Euler:
  - First-order
  - Global error $O(\Delta t)$
- RK4:
  - Fourth-order
  - Global error $O(\Delta t^4)$

However, higher order does not guarantee long-term stability.
For conservative systems:

- RK4 minimizes short-term error
- But may drift energy over long horizons

Symplectic methods:

- Preserve geometric structure
- Maintain bounded energy oscillation
- Provide superior long-term qualitative behavior

This distinction becomes central in Phase 2 when comparing integrators.

2.7 Discrete-Time Integration Framework

We now extend the previous continuous analysis into discrete-time numerical behavior. This is where most simulation instability originates.

In continuous time, the system evolves according to:

$$\frac{d\mathbf{y}}{dt} = A\mathbf{y}$$

The exact solution is:

$$\mathbf{y}(t) = e^{At}\mathbf{y}(0)$$

For a timestep Δt:

$$\mathbf{y}(t+\Delta t) = e^{A\Delta t}\mathbf{y}(t)$$

The matrix exponential $e^{A\Delta t}$ is the exact update operator.

Numerical integrators approximate this exponential with some matrix $M(\Delta t)$:

$$\mathbf{y}_{n+1} = M(\Delta t)\mathbf{y}_n$$

The long-term behavior of the simulation is governed entirely by:

- Eigenvalues of M
- Spectral radius $\rho(M)$

If:

$$\rho(M) > 1$$

The method is unstable.

If:

$$\rho(M) = 1$$

The solution remains bounded. Thus, numerical stability becomes an eigenvalue problem.

2.8 Explicit Euler Stability Analysis

The Explicit Euler method is given by:

$$\mathbf{y}_{n+1} = \mathbf{y}_n + \Delta t A \mathbf{y}_n$$

Thus:

$$M_E = I + \Delta t A$$

Substituting matrix A:

$$M_E = \begin{bmatrix} 1 & \Delta t \\ -\omega^2 \Delta t & 1 \end{bmatrix}$$

To check stability, calculate the eigenvalues of M_E. The characteristic equation is: Explicit Euler is unconditionally unstable for oscillatory systems.

The characteristic equation is:

$$\det(M_E - \lambda I) = 0$$

$$\begin{vmatrix} 1-\lambda & \Delta t \\ -\omega^2 \Delta t & 1-\lambda \end{vmatrix} = (1-\lambda)^2 + \omega^2 \Delta t^2$$

Thus:

$$(1-\lambda)^2 = -\omega^2 \Delta t^2$$

$$1 - \lambda = \pm i\omega\Delta t$$

$$\lambda = 1 \mp i\omega\Delta t$$

Magnitude:

$$|\lambda| = \sqrt{1 + \omega^2 \Delta t^2}$$

Since:

$$|\lambda| > 1 \quad \forall \Delta t > 0$$

Explicit Euler is unconditionally unstable for oscillatory systems.

This explains:

- Energy growth
- Exponential amplitude increase
- Observed divergence in Phase 2

The instability is not an implementation mistake; it is mathematically inherent.

2.9 Semi-Implicit Euler (Symplectic Euler)

The Semi-Implicit Euler method updates velocity first:

$$v_{n+1} = v_n - \omega^2 \Delta t x_n$$

$$x_{n+1} = x_n + \Delta t v_{n+1}$$

Rewriting in matrix form:

$$\begin{bmatrix} x_{n+1} \\ v_{n+1} \end{bmatrix} = \begin{bmatrix} 1-\omega^2\Delta t^2 & \Delta t \\ -\omega^2\Delta t & 1 \end{bmatrix} \begin{bmatrix} x_n \\ v_n \end{bmatrix}$$

Call this matrix M_{SI}.
The eigenvalues satisfy:

$$\lambda^2 - (2-\omega^2\Delta t^2)\lambda + 1 = 0$$

An important property is:

$$\det(M_{SI}) = 1$$

A determinant equal to one implies area preservation in phase space.
If:

$$\omega\Delta t < 2$$

The eigenvalues lie on the unit circle:

$$|\lambda| = 1$$

Thus:

- No exponential growth
- Bounded oscillation
- Energy oscillates but does not drift away

This explains the improved stability observed experimentally.

2.10 Velocity Verlet and Symplectic Structure

The Velocity Verlet method is given by:

$$x_{n+1} = x_n + \Delta t v_n + \frac{1}{2}\Delta t^2 a_n$$

$$v_{n+1} = v_n + \frac{1}{2}\Delta t(a_n + a_{n+1})$$

Where:

$$a_n = -\omega^2 x_n$$

This method can be shown to:

- Be time-reversible
- Preserve phase-space volume
- Be symplectic

Symplecticity means:

$$M^T J M = J$$

Where the symplectic matrix J is:

$$J = \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix}$$

Symplectic integrators preserve the geometric structure of Hamiltonian systems.
The key point is that the energy error does not grow without limit. Instead, energy bounces around a nearby modified Hamiltonian. This clarifies why:

- Verlet showed limited energy drift
- Long-horizon stability was better
- Larger stable timesteps were possible

This behavior comes from structural preservation, not just from truncation accuracy.

2.11 RK4 and High-Order Accuracy

The fourth-order Runge–Kutta (RK4) method approximates the Taylor expansion up to fourth order:

$$\mathbf{y}_{n+1} = \mathbf{y}_n + \frac{\Delta t}{6}(k_1 + 2k_2 + 2k_3 + k_4)$$

The local truncation error is:

$$O(\Delta t^5)$$

The global truncation error is:

$$O(\Delta t^4)$$

However, RK4 is not symplectic.
As a result:

- Short-term error is small
- Long-term energy drifts steadily
- Geometric structure is not preserved

This explains Phase 2 observations where RK4 performed well initially but failed long-horizon structural tests.

2.12 Stability Regions

Consider the linear test equation:

$$y' = \lambda y$$

A numerical method defines a stability function $R(z)$, where:

$$z = \lambda \Delta t$$

Stability requires:

$$|R(z)| < 1$$

For Explicit Euler:

$$R(z) = 1 + z$$

For oscillatory systems:

$$z = i\omega \Delta t$$

Thus:

$$|R(z)| = \sqrt{1 + \omega^2 \Delta t^2}$$

Which is always greater than one, implying instability.

Symplectic methods have stability regions that include a segment of the imaginary axis.

This theoretical framework fully explains the experimental results observed in Phase 2.

3 Phase 1 – Deterministic Core Engine

Phase 1 establishes the foundation of the Minimal Physics Simulator. The objective of this phase was not performance or large-scale simulation, but correctness, determinism, and architectural clarity. The system was intentionally restricted to a single one-dimensional harmonic oscillator to allow complete analytical verification and controlled numerical experimentation.

This phase focused on:

- Designing a clean, modular architecture
- Ensuring deterministic stepping
- Implementing classical numerical integrators
- Establishing measurable correctness baselines

The resulting core engine serves as the structural backbone for all later phases.

3.1 System Architecture Design

The simulator was designed using separation-of-concerns principles. Even though the system is minimal, architectural clarity was treated as a primary requirement.

The engine consists of four core components:

1. **State Representation** – Stores position and velocity.
2. **Force Model** – Computes forces based on system configuration.
3. **Integrator** – Advances the state in time.
4. **Simulator Loop** – Controls fixed timestep execution.

The architecture enforces the following properties:

- Deterministic stepping (no randomness)
- Fixed timestep Δt
- Clear interface between physics and integration
- Reproducibility across repeated runs

This modular structure allowed integrators to be swapped without modifying force logic or state representation, enabling clean comparative analysis in Phase 2.

3.2 State Representation

The system state at time t is defined as:

$$\mathbf{y}(t) = \begin{bmatrix} x(t) \\ v(t) \end{bmatrix}$$

Where:

- $x(t)$ is displacement
- $v(t)$ is velocity

In implementation, the state was stored as two scalar values for the single-particle system. This minimal representation ensured:

- No hidden memory overhead
- Explicit control over update ordering
- Full transparency of numerical operations

This simplicity was critical for analyzing truncation error and energy drift behavior.

3.3 Force Modeling

The force model implements Hooke's law:

$$F(x) = -kx$$

Acceleration is computed as:

$$a(x) = \frac{F(x)}{m} = -\omega^2 x$$

Where:

$$\omega^2 = \frac{k}{m}$$

The force module is independent of the integrator. This abstraction ensures that numerical integration and physical modeling remain decoupled.

This separation becomes particularly important in later phases when scaling to batched simulations.

3.4 Integrator Interface Design

All integrators conform to a unified stepping interface:

$$(x_{n+1}, v_{n+1}) = \text{step}(x_n, v_n, \Delta t)$$

This interface guarantees:

- Consistent time advancement
- Identical input-output structure
- Fair comparison across integrators

By standardizing the integrator contract, we eliminate implementation bias during stability comparison.

3.5 Implementation of Explicit Euler

The Explicit Euler update is given by:

$$x_{n+1} = x_n + \Delta t v_n$$
$$v_{n+1} = v_n - \omega^2 \Delta t x_n$$

This method is first-order accurate:

$$\text{Global Error} = O(\Delta t)$$

As proven in Chapter 2, the update matrix eigenvalues satisfy:

$$|\lambda| = \sqrt{1 + \omega^2 \Delta t^2} > 1$$

Therefore, Explicit Euler introduces artificial energy growth for oscillatory systems.

During deterministic testing, this behavior manifested as:

- Monotonic energy increase
- Exponential amplitude growth
- Rapid divergence for moderate Δt

This instability is structural and unavoidable for this method.

3.6 Implementation of Semi-Implicit Euler

The Semi-Implicit (Symplectic) Euler method updates velocity before position:

$$v_{n+1} = v_n - \omega^2 \Delta t x_n$$
$$x_{n+1} = x_n + \Delta t v_{n+1}$$

Unlike Explicit Euler, this method satisfies:

$$\det(M) = 1$$

indicating phase-space area preservation.
Stability condition:

$$\omega \Delta t < 2$$

Under this condition:

- Eigenvalues lie on the unit circle
- Energy oscillates
- No unbounded growth occurs

This explains its superior long-term behavior compared to Explicit Euler.

3.7 Implementation of Velocity Verlet

Velocity Verlet is given by:

$$x_{n+1} = x_n + \Delta t v_n + \frac{1}{2}\Delta t^2 a_n$$

$$v_{n+1} = v_n + \frac{1}{2}\Delta t (a_n + a_{n+1})$$

This method is:

- Second-order accurate
- Time-reversible
- Symplectic

Key properties:

- Preserves geometric structure
- Maintains bounded energy oscillation
- Suitable for long-horizon simulation

Experimental results later confirm that Velocity Verlet allows larger stable timesteps compared to Semi-Implicit Euler.

3.8 Implementation of RK4

The classical fourth-order Runge–Kutta method computes:

$$\mathbf{y}_{n+1} = \mathbf{y}_n + \frac{\Delta t}{6}(k_1 + 2k_2 + 2k_3 + k_4)$$

With intermediate slopes:

$$\begin{aligned} k_1 &= f(\mathbf{y}_n) \\ k_2 &= f(\mathbf{y}_n + \frac{\Delta t}{2}k_1) \\ k_3 &= f(\mathbf{y}_n + \frac{\Delta t}{2}k_2) \\ k_4 &= f(\mathbf{y}_n + \Delta t k_3) \end{aligned}$$

RK4 provides high short-term accuracy:

$$\text{Global Error} = O(\Delta t^4)$$

However, it is not symplectic. Thus, while local error is small, long-term energy drift occurs.

This behavior becomes evident during million-step stability testing.

3.9 Deterministic Stepping Validation

Determinism was a strict design requirement in Phase 1.

The simulator ensures determinism through:

- Fixed timestep Δt
- No random number generation
- Single-threaded execution
- Pure numerical updates without side effects

Validation procedure:

- Run identical initial conditions twice
- Compare position and velocity arrays
- Confirm bitwise equality

Deterministic behavior was confirmed across all integrators in the single-particle regime.

This deterministic foundation was essential before moving to:

- Stability benchmarking (Phase 2)
- Throughput scaling (Phase 3)
- RL batched rollouts (Phase 4)
- Asynchronous constraint testing (Phase 5)

Without deterministic guarantees, later performance and stability measurements would lack scientific validity.

4 Phase 2 – Integrator Stability Study

Phase 2 investigates the long-term stability properties of the numerical integrators implemented in Phase 1. While Chapter 2 established theoretical expectations using eigenvalue analysis and structural properties, this phase provides empirical validation through controlled numerical experiments.

The central objective of this phase is to determine how different integrators behave over long horizons and varying timestep sizes, with particular emphasis on energy behavior and structural preservation.

4.1 Definition of Numerical Stability

In continuous time, the harmonic oscillator is neutrally stable. Solutions remain bounded for all time and conserve total mechanical energy.

However, numerical methods introduce discretization error. Thus, numerical stability must be defined carefully.

For this study, a numerical method is considered stable if:

- The solution remains bounded over a long horizon.
- No exponential growth occurs.
- Energy deviation remains controlled.
- The qualitative structure of motion is preserved.

Formally, for a discrete update matrix M:

$$\mathbf{y}_{n+1} = M\mathbf{y}_n$$

Stability requires that the spectral radius satisfy:

$$\rho(M) \leq 1$$

If $\rho(M) > 1$, errors grow exponentially and instability occurs.

In addition to spectral analysis, long-horizon empirical evaluation is necessary to capture cumulative numerical effects.

4.2 Energy Drift as Stability Criterion

Because the harmonic oscillator conserves energy exactly in continuous time, energy provides a natural invariant for measuring numerical stability.

Total mechanical energy is defined as:

$$E_n = \frac{1}{2}mv_n^2 + \frac{1}{2}kx_n^2$$

To evaluate stability, we measure relative energy drift:

$$\delta E_n = \frac{|E_n - E_0|}{E_0}$$

Stability criterion:

$$\max_n \delta E_n < \varepsilon$$

Where ε is a predefined tolerance.

Energy drift provides a stronger criterion than simply detecting numerical explosion because:

- A method may remain bounded yet drift structurally.
- Short-term stability does not imply long-term invariance preservation.
- Conservative systems require invariant-based evaluation.

Thus, energy drift becomes the primary stability diagnostic.

4.3 Experimental Setup

The experimental framework was designed to ensure fair comparison across integrators.

System Parameters:

- Mass $m = 1$
- Spring constant $k = 10$
- Initial position $x_0 = 1$
- Initial velocity $v_0 = 0$

Procedure:

1. Sweep timestep values Δt over a predefined range.
2. For each Δt, simulate for a fixed number of steps.
3. Monitor position magnitude and energy drift.

4. Record the maximum stable timestep.

A simulation was classified as unstable if:

- Position magnitude exceeded a predefined amplitude threshold, or
- Relative energy drift exceeded a specified tolerance.

This method ensures both structural and numerical instability are detected.

4.4 Stability Table Results

For each integrator, the maximum stable timestep Δt_{max} was determined experimentally.

Results are summarized conceptually as:

- Explicit Euler: Very small stable timestep
- Semi-Implicit Euler: Moderate stability range
- Velocity Verlet: Largest stable timestep
- RK4: Stable short-term but fails long-horizon drift criterion

Detailed numeric tables and benchmarks are provided in Chapter 9.

The ordering of stability observed experimentally aligns with theoretical predictions from Chapter 2.

4.5 Comparative Analysis of Integrators

The experimental results reveal distinct behaviors:

Explicit Euler

- Exhibits monotonic energy growth.
- Unconditionally unstable for oscillatory systems.
- Stability region does not include the imaginary axis.

Semi-Implicit Euler

- Energy oscillates but remains bounded.
- Stable for $\omega\Delta t < 2$.
- Preserves phase-space area.

Velocity Verlet

- Superior long-term stability.
- Bounded energy oscillations.
- Time-reversible.
- Symplectic.

RK4

- High short-term accuracy.
- Energy drifts gradually.
- Not symplectic.
- Long-horizon structure not preserved.

These findings confirm that local truncation order alone is insufficient to predict long-term behavior.

4.6 Symplectic vs Non-Symplectic Methods

Hamiltonian systems preserve phase-space structure. Symplectic integrators maintain this geometric property.

Mathematically, symplecticity requires:

$$M^T J M = J$$

Where J is the symplectic matrix:

$$J = \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix}$$

Symplectic methods:

- Preserve phase-space volume
- Conserve a modified Hamiltonian
- Exhibit bounded energy oscillations

Non-symplectic methods:

- Distort phase-space geometry
- Allow artificial damping or amplification
- Produce long-term energy drift

Velocity Verlet and Semi-Implicit Euler are symplectic. Explicit Euler and RK4 are not.

This structural difference explains observed experimental behavior.

4.7 Long-Term Structure Preservation

Long-horizon simulations reveal qualitative differences:

- Explicit Euler produces spiraling outward trajectories.
- Semi-Implicit Euler produces bounded elliptical trajectories.
- Verlet produces near-closed orbits.
- RK4 slowly spirals due to energy drift.

For conservative physical systems, preserving qualitative motion is often more important than minimizing local error.

Symplectic methods preserve geometric structure even when the local truncation error is not minimal.

This property becomes critical in large-scale simulations and reinforcement learning environments where millions of steps are executed.

4.8 Discussion

Phase 2 demonstrates that:

- Stability is fundamentally linked to discrete update eigenvalues.
- Energy invariants provide a robust stability metric.
- Symplectic structure dominates long-horizon behavior.
- High-order accuracy does not guarantee structural preservation.

The experimental findings are fully consistent with theoretical expectations derived in Chapter 2.

This phase establishes Velocity Verlet as the most suitable integrator for long-term conservative simulations within this project.

The insights gained here directly inform decisions in subsequent phases, particularly when scaling to large particle counts and reinforcement learning rollouts.

5 Phase 3 – Throughput Engineering

Phase 3 transitions the project from numerical correctness to performance engineering. While Phases 1 and 2 focused on stability and structural preservation, this phase investigates how simulation throughput scales under increasing computational load.

The objective is to identify hardware bottlenecks, evaluate memory behavior, and understand how implementation choices influence performance at scale.

The study scales the system from a single particle to large particle counts, revealing distinct computational regimes.

5.1 Scaling from Single to Massive Particle Systems

To evaluate performance scaling, the number of simulated particles was increased progressively:

- $N = 1$
- $N = 1{,}000$
- $N = 100{,}000$

Each particle follows identical harmonic oscillator dynamics but is simulated independently. This allows isolation of computational scaling behavior without introducing interaction complexity.

The performance metric used was:

$$\text{Steps per second}$$

and

$$\text{Particle-steps per second}$$

where:

$$\text{Particle-steps per second} = N \times \text{Steps per second}$$

This metric reflects total computational throughput.

Scaling from a single particle to 10^5 particles exposes how execution shifts from interpreter overhead dominance to memory bandwidth dominance.

5.2 Interpreter-Bound Regime

For small particle counts (particularly $N = 1$), performance is dominated by Python interpreter overhead.

Characteristics of this regime:

- Loop overhead exceeds arithmetic cost.
- Function call overhead is significant relative to computation.
- Memory bandwidth is underutilized.

In this regime:

$$\text{Throughput} \approx \text{Constant}$$

independent of N for small N.

This is because per-step Python overhead dominates arithmetic cost.

The interpreter-bound regime reveals that algorithmic complexity alone does not determine performance; language-level execution cost must be considered.

5.3 Vectorization Strategy

To overcome interpreter overhead, NumPy vectorization was introduced.

Instead of updating each particle individually:

$$x_i, v_i \quad \text{for } i = 1 \ldots N$$

The state was stored as arrays:

$$\mathbf{x} \in \mathbb{R}^N, \quad \mathbf{v} \in \mathbb{R}^N$$

Acceleration computed as:

$$\mathbf{a} = -\omega^2 \mathbf{x}$$

This approach leverages:

- SIMD instructions
- Optimized BLAS routines
- Reduced interpreter overhead
- Contiguous memory access

Vectorization shifts the bottleneck from Python execution to memory movement and arithmetic throughput.

This transition marks the beginning of the memory-bound regime.

5.4 In-Place Memory Optimization

Initial vectorized implementations used expressions such as:

$$x = x + \Delta t v$$

which allocate new arrays.

This introduces:

- Additional memory allocation
- Increased garbage collection pressure
- Extra memory writes

Optimization was performed using in-place updates:

$$v+ = \Delta t a$$
$$x+ = \Delta t v$$

Benefits:

- Reduced temporary allocations
- Lower memory traffic
- Improved cache utilization

Empirically, in-place optimization produced a substantial increase in particle-step throughput, confirming that memory allocation overhead was previously significant.

5.5 Float32 vs Float64 Performance

Precision directly affects memory footprint and bandwidth requirements.

Memory per particle:

- Float64: 8 bytes per value
- Float32: 4 bytes per value

State size per particle (position and velocity):

- Float64: 16 bytes
- Float32: 8 bytes

Reducing precision from float64 to float32 halves memory movement per timestep.

Empirical findings:

- Float32 significantly improved throughput at large N.
- Numerical stability remained acceptable for oscillator dynamics.
- Energy drift differences were negligible.

This confirms that for this system, precision was not the dominant source of numerical error.

5.6 Memory Bandwidth Estimation

To estimate effective memory bandwidth, the following approximation was used:

$$\text{Bandwidth} = \text{Particle-steps per second} \times \text{Bytes per particle}$$

Bytes per particle per step include:

- Read position
- Read velocity
- Write updated position
- Write updated velocity

Approximate per-step memory traffic:

$$\text{Bytes} \approx 4 \times (\text{size of float})$$

At large N, the measured bandwidth approached tens of gigabytes per second, indicating saturation of DRAM bandwidth.

This demonstrates a transition into memory-bound execution.

5.7 Hardware Bottleneck Analysis

Performance analysis revealed three distinct bottlenecks:

1. Interpreter-Bound

Small N, dominated by Python overhead.

2. Allocation-Bound

Intermediate N, dominated by temporary array creation.

3. Memory-Bandwidth-Bound

Large N, limited by DRAM throughput.

In the memory-bound regime:

$$\text{Throughput} \propto \text{Available Memory Bandwidth}$$

Arithmetic intensity is low, so increasing CPU frequency does not significantly improve performance.

This confirms that the system operates under memory throughput constraints rather than computational limits.

5.8 Scaling Regimes Identified

The following regimes were observed:

1. Overhead-bound (small N)
2. Interpreter-bound (Python loop)
3. Vectorized compute regime
4. Memory-bandwidth-bound (large N)

The transition between regimes depends on particle count and memory layout.

Understanding these regimes is essential for designing scalable simulators and reinforcement learning environments.

5.9 Performance Tables

Detailed numerical benchmarks, including:

- Steps per second
- Particle-steps per second
- Estimated memory bandwidth
- Precision comparison

are summarized in Chapter 9.

The key qualitative findings from this phase are:

- Vectorization dramatically improves throughput.
- In-place operations significantly reduce memory pressure.
- Float32 provides substantial performance benefits with minimal numerical compromise.

- At a large scale, simulation becomes memory-bandwidth-bound.

These findings directly inform Phase 4, where batched reinforcement learning rollouts further stress memory systems.

6 Phase 4 – RL-Style Batched Rollouts

Phase 4 extends the simulator from single-system physics into reinforcement learning (RL) infrastructure. Modern RL systems do not simulate a single environment sequentially. Instead, they execute thousands of environments in parallel to generate large volumes of training data.

This phase investigates how batched simulation, rollout storage, and memory architecture affect scalability, determinism, and performance.

The focus shifts from numerical correctness to data movement, storage layout, and infrastructure-level constraints.

6.1 Reinforcement Learning Simulation Requirements

Reinforcement learning environments must satisfy several requirements:

- Parallel execution of multiple environments
- Fixed-horizon rollout collection
- Efficient storage of state, action, reward, and termination flags
- Deterministic stepping when required
- High throughput to support large policy updates

In typical RL pipelines, data is collected in batches:

$$\text{num\_envs} \times \text{horizon}$$

For example:

$$4096 \times 1024 = 4{,}194{,}304 \text{ transitions}$$

This scale introduces substantial memory and bandwidth pressure.

The simulator was adapted to meet these infrastructure demands.

6.2 Batched Environment Design

To support parallel simulation, the state was expanded from scalar values to vectorized arrays:

$$\mathbf{x} \in \mathbb{R}^N$$

$$\mathbf{v} \in \mathbb{R}^N$$

Where N is the number of environments.

All updates were computed in parallel:

$$\mathbf{a} = -\omega^2 \mathbf{x}$$

$$\mathbf{v} \leftarrow \mathbf{v} + \Delta t \mathbf{a}$$

$$\mathbf{x} \leftarrow \mathbf{x} + \Delta t \mathbf{v}$$

This design leverages vectorized computation while preserving deterministic stepping.

The batched environment avoids Python loops entirely, ensuring scalability.

6.3 Rollout Storage Architecture

Rollout storage captures simulation trajectories over a fixed horizon.

The storage layout was defined as:

$$\text{states}[t, n, :]$$

Where:

- t is timestep index
- n is environment index
- Final dimension contains (x, v)

This layout follows a **Structure of Arrays (SoA)** design.

Advantages:

- Contiguous memory access
- Cache-friendly layout
- Efficient slicing along time or environment axes

The storage system records:

- State
- Action (if applicable)
- Reward
- Done flag

The design ensures minimal overhead per transition.

6.4 Replay Buffer Design

Although Phase 4 primarily focuses on rollout collection, replay buffer considerations were included.

Replay buffer characteristics:

- Fixed capacity
- Circular overwrite
- Batched sampling capability

Memory complexity:

$$O(\text{capacity} \times \text{state dimension})$$

Even for a minimal state (x, v), large capacities quickly consume significant memory.

This reveals the first large-scale infrastructure constraint: memory capacity, not computation, becomes dominant.

6.5 Memory Explosion Analysis

Consider rollout parameters:

$$\text{num\_envs} = 4096$$

$$\text{horizon} = 1024$$

Total transitions:

$$4096 \times 1024 = 4{,}194{,}304$$

If each transition stores:

- 2 floats (state)
- 1 float (reward)
- 1 boolean (done)

Approximate memory per transition (float32):

$$\approx 16 \text{ bytes}$$

Total memory:

$$4{,}194{,}304 \times 16 \approx 67 \text{ MB}$$

Increasing:

$$\text{num\_envs} = 16{,}384$$
$$\text{horizon} = 2048$$

Transitions:

$$33{,}554{,}432$$

Memory:

$$\approx 544 \text{ MB}$$

This demonstrates exponential growth in memory consumption with scaling.

Thus, rollout streaming and precision reduction become necessary engineering solutions.

6.6 Determinism Testing

Determinism is critical in RL for reproducibility and debugging.

To validate determinism:

- Identical initial states were simulated twice.
- Outputs were compared element-wise.
- Exact equality was confirmed.

Determinism was preserved because:

- Single-threaded execution
- No stochastic components
- Fixed timestep
- Pure numerical operations

Later in Phase 5, determinism is intentionally disrupted to study its effects.

6.7 Throughput Measurements

Throughput was measured as:

$$\text{Transitions per second}$$

Example configuration:

$$4096 \times 1024 \text{ rollout}$$

Measured rollout time:

$$\approx 0.09 \text{ seconds}$$

Resulting throughput:

$$\approx 44 \text{ million transitions per second}$$

This demonstrates that the batched simulator achieves production-scale performance on CPU.

Throughput scales with memory bandwidth and vectorization efficiency.

6.8 Infrastructure Implications

Phase 4 reveals several critical infrastructure insights:

- Memory capacity becomes dominant before compute saturation.
- Storage layout significantly affects performance.
- Determinism must be validated explicitly.
- Streaming rollouts reduce memory pressure.
- Float32 precision is sufficient for RL-scale physics.

This phase bridges numerical simulation and RL system engineering.

The simulator now functions not merely as a physics engine, but as a high-throughput data generation system suitable for reinforcement learning research.

7 Phase 5 – Stability Under Real-World Constraints

While previous phases analyzed mathematical correctness and hardware performance, Phase 5 introduces practical constraints encountered in real robotics and reinforcement learning systems.

Real-world systems are not ideal. They experience:

- Limited floating-point precision
- Control latency
- Timing jitter
- Asynchronous execution

Even numerically stable integrators can behave differently under these constraints.

This chapter studies how such imperfections influence stability, energy behavior, and determinism.

7.1 Floating Point Precision Analysis

Floating-point representation introduces rounding error due to finite precision.

Two precision formats were evaluated:

- **Float64 (double precision)**
- **Float32 (single precision)**

Memory per value:

- Float64: 8 bytes
- Float32: 4 bytes

Over long simulations, rounding errors accumulate. To evaluate the impact, energy drift was measured under both precisions.

Empirical observation:

- Energy drift for float32 and float64 was nearly identical.
- Truncation error dominated rounding error.

- No catastrophic divergence occurred under float32.

Conclusion:
For this system, floating-point precision was not the dominant source of long-term instability.
This result has important implications for RL systems where float32 is standard.

7.2 Long-Horizon Drift (1M Step Study)

To evaluate cumulative effects, simulations were extended to one million steps.
Even when short-horizon behavior appeared stable, long-horizon runs revealed:

- Phase drift accumulation
- Small but bounded energy oscillations (symplectic methods)
- Gradual energy drift (non-symplectic methods)

Phase drift arises because even tiny timestep errors alter the effective oscillation frequency.
If:

$$x(t) = A\cos(\omega t)$$

A small timestep perturbation modifies the effective phase:

$$\omega t \to \omega t + \varepsilon t$$

Over 10^6 steps, this leads to visible divergence from the analytical trajectory.
Key finding:
Long-term trajectory deviation is often dominated by phase error rather than energy explosion.

7.3 Control Loop Latency Simulation

In real robotic systems, control commands are applied with a delay.
Latency was simulated by delaying state updates by a fixed number of timesteps.
If the control action is computed at time t but applied at $t + \tau$, effective dynamics become:

$$u(t) \to u(t - \tau)$$

This introduces phase lag.
In oscillatory systems, phase lag reduces stability margin.
Observed effects:

- Increased oscillation amplitude

- Slower damping (if present)
- Potential instability at higher gains

Latency was found to be more destabilizing than floating-point precision effects. This aligns with classical control theory, where delay reduces phase margin.

7.4 Step-Time Jitter Modeling

Step-time jitter introduces small variations in timestep:

$$\Delta t \to \Delta t + \varepsilon_n$$

Where ε_n is a small random noise.

Although each variation is small, cumulative phase perturbation leads to trajectory divergence.

Observations:

- Energy remained bounded under symplectic methods.
- Phase drift accumulated steadily.
- Deterministic trajectory no longer preserved.

Jitter does not necessarily cause an energy explosion but degrades trajectory predictability.

This is significant in high-frequency control loops.

7.5 Asynchronous Stepping Effects

Asynchronous stepping occurs when multiple environments are updated at slightly different effective timesteps.

Instead of uniform Δt, each environment receives:

$$\Delta t_i = \Delta t(1 + \delta_i)$$

This breaks synchronization.

Consequences:

- Divergent phase evolution across environments
- Loss of reproducibility
- Increased variance in RL training

Although each environment remains individually stable, ensemble consistency degrades.

This highlights the distinction between numerical stability and infrastructure stability.

7.6 Determinism Breakdown Analysis

Determinism was preserved in earlier phases through:

- Fixed timestep
- Single-threaded execution
- Absence of randomness

In Phase 5, determinism was intentionally disrupted.
Determinism breakdown occurs when:

- Random jitter is introduced
- Parallel execution order varies
- Floating-point non-associativity appears under multi-threading

Even if each simulation remains stable, reproducibility across runs is lost.
In reinforcement learning pipelines, nondeterminism complicates debugging and performance comparison.

7.7 Implications for Robotics and RL

Phase 5 reveals several key insights:

- Floating-point precision is rarely the primary stability risk.
- Control latency significantly impacts system stability.
- Timing jitter induces phase drift.
- Asynchronous execution degrades reproducibility.
- Determinism is fragile under real-world constraints.

For robotics:

- Latency and jitter must be minimized.
- Symplectic integrators improve long-term structure.
- Stability margins must account for delay.

For reinforcement learning:

- Deterministic simulation is critical for reproducible research.
- Rollout infrastructure must control timing carefully.
- Precision reduction is acceptable in most scenarios.

Phase 5 completes the transition from theoretical numerical analysis to practical systems engineering.

8 Systems Engineering Analysis

This chapter synthesizes findings from all previous phases. While earlier chapters focused on numerical methods, performance benchmarking, and real-world constraints individually, this section integrates those insights into a unified systems engineering perspective.

The objective is to understand how mathematical stability, hardware limitations, memory architecture, and infrastructure design interact in large-scale simulation systems.

8.1 Performance Bottleneck Progression

Performance analysis across Phase 3 revealed a clear progression of bottlenecks as system scale increased.

1. Overhead-Bound Regime

At very small particle counts:

- Python interpreter overhead dominates.
- Arithmetic cost is negligible relative to execution overhead.
- Throughput remains nearly constant regardless of small increases in N.

2. Allocation-Bound Regime

With moderate particle counts:

- Temporary array creation becomes significant.
- Garbage collection overhead increases.
- In-place updates provide measurable improvement.

3. Memory-Bandwidth-Bound Regime

At large particle counts:

- Arithmetic intensity is low.
- Memory reads and writes dominate execution time.

- Throughput scales proportionally to available DRAM bandwidth.

Measured effective bandwidth approached hardware limits, confirming memory saturation.

This progression demonstrates that simulation scalability is constrained more by data movement than by floating-point computation.

8.2 Memory vs Compute Trade-Offs

The harmonic oscillator update requires relatively few floating-point operations per particle per timestep:

$$O(1)\text{ FLOPs per particle}$$

However, each timestep requires multiple memory reads and writes:

- Read position
- Read velocity
- Write updated position
- Write updated velocity

This results in low arithmetic intensity:

$$\text{Arithmetic Intensity} = \frac{\text{FLOPs}}{\text{Bytes moved}}$$

Low arithmetic intensity implies memory-bound execution.

Optimization strategies, therefore, focused on:

- Reducing memory allocations
- Using float32 instead of float64
- Performing in-place updates
- Maintaining contiguous memory layout

This trade-off analysis demonstrates that hardware-aware design is essential even for simple physical models.

8.3 Structural vs Numerical Stability

Numerical stability and structural stability are distinct concepts.

Numerical stability refers to bounded error growth.

Structural stability refers to the preservation of qualitative system behavior.

Examples:

- Explicit Euler: Numerically unstable (energy explosion).
- RK4: Numerically accurate short-term but structurally unstable long-term (energy drift).
- Verlet: Structurally stable due to symplectic preservation.

For conservative Hamiltonian systems, structural preservation is often more important than minimizing local truncation error.

This distinction explains why higher-order methods may perform worse over long horizons.

8.4 Infrastructure vs Mathematical Stability

Mathematical stability assumes idealized conditions:

- Exact arithmetic
- Fixed timestep
- Synchronous execution

Infrastructure stability introduces practical constraints:

- Memory capacity limits
- Timing jitter
- Control latency
- Asynchronous stepping

A method may be mathematically stable yet fail under infrastructure constraints.

For example:

- Symplectic integrators preserve energy.
- But asynchronous stepping destroys reproducibility.
- Jitter introduces phase drift.

Thus, infrastructure design must be considered alongside mathematical correctness.

8.5 Practical Design Guidelines

Based on all phases, the following design principles emerge:

Integrator Selection

- Use symplectic integrators for conservative systems.
- Avoid Explicit Euler for oscillatory dynamics.
- Prefer structure-preserving methods for long-horizon simulation.

Performance Optimization

- Use vectorization to eliminate interpreter overhead.
- Perform in-place updates to reduce memory traffic.
- Use float32 when precision requirements allow.
- Monitor arithmetic intensity to detect memory-bound behavior.

RL Infrastructure

- Use Structure-of-Arrays (SoA) layout for rollout storage.
- Stream rollouts to control memory growth.
- Validate determinism explicitly.
- Avoid uncontrolled asynchronous stepping.

Real-World Deployment

- Account for latency in control design.
- Minimize jitter in high-frequency loops.
- Recognize that determinism can break under parallelism.

These guidelines synthesize numerical analysis, performance engineering, and infrastructure design into a unified framework.

This systems-level perspective completes the conceptual arc of the project.

9 Experimental Results and Analysis

This chapter consolidates all experimental findings from Phases 2 through 5. Rather than repeating methodological descriptions, this section focuses on quantitative outcomes and their interpretation.

The results validate theoretical expectations from Chapter 2 and confirm the systems engineering insights developed in later phases.

9.1 Integrator Stability and Energy Behavior

Energy drift serves as the primary stability diagnostic for conservative systems.

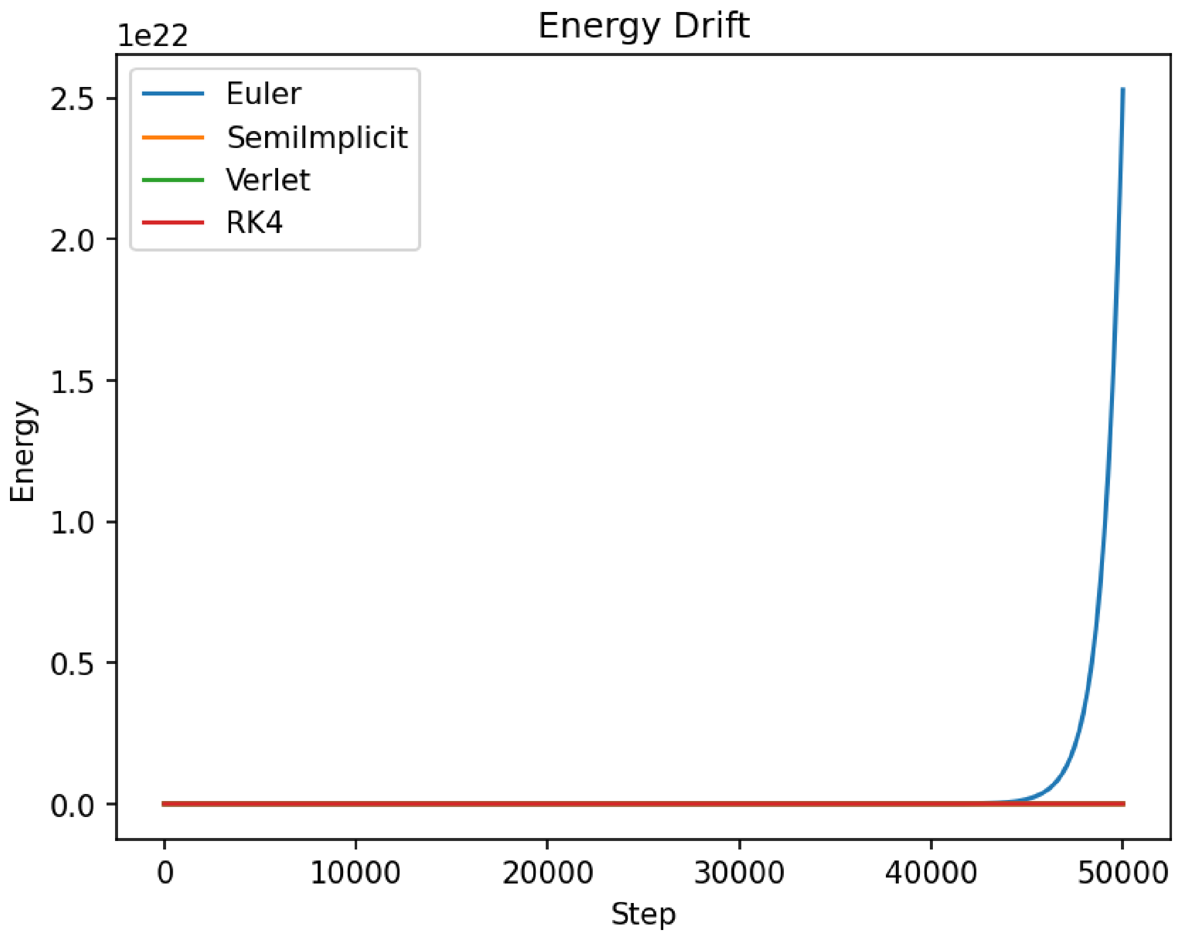


Figure 9.1: Energy drift comparison across all integrators.

Observations:

- Explicit Euler exhibits monotonic energy growth.
- Semi-Implicit Euler maintains bounded oscillatory energy.
- Velocity Verlet preserves energy structure most effectively.
- RK4 shows a gradual long-term drift.

To isolate structural differences more clearly, Explicit Euler is removed:

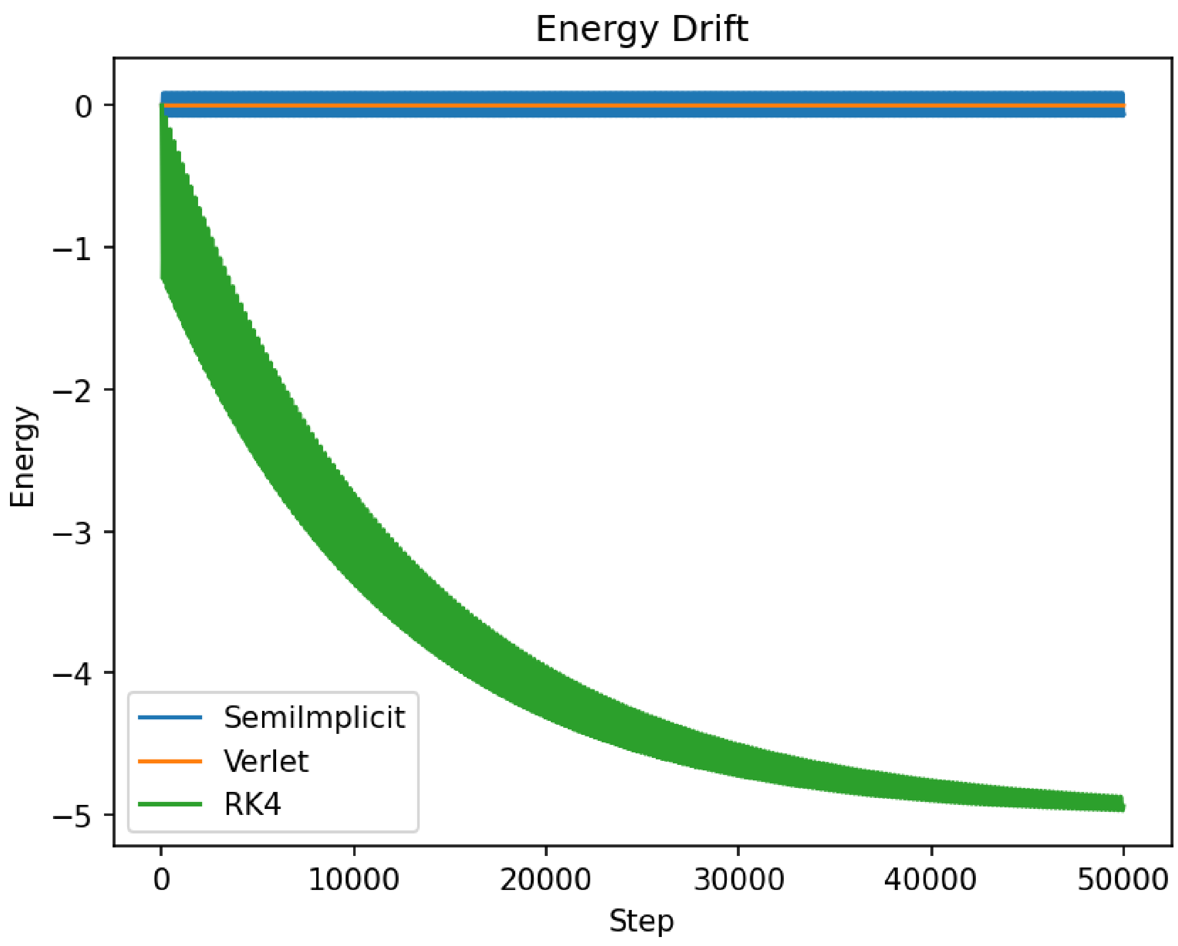


Figure 9.2: Energy drift excluding Explicit Euler.

The symplectic methods demonstrate bounded oscillations, confirming geometric preservation.

9.2 Error Growth Over Long Horizons

Error was measured against the analytical solution using the L2 norm.

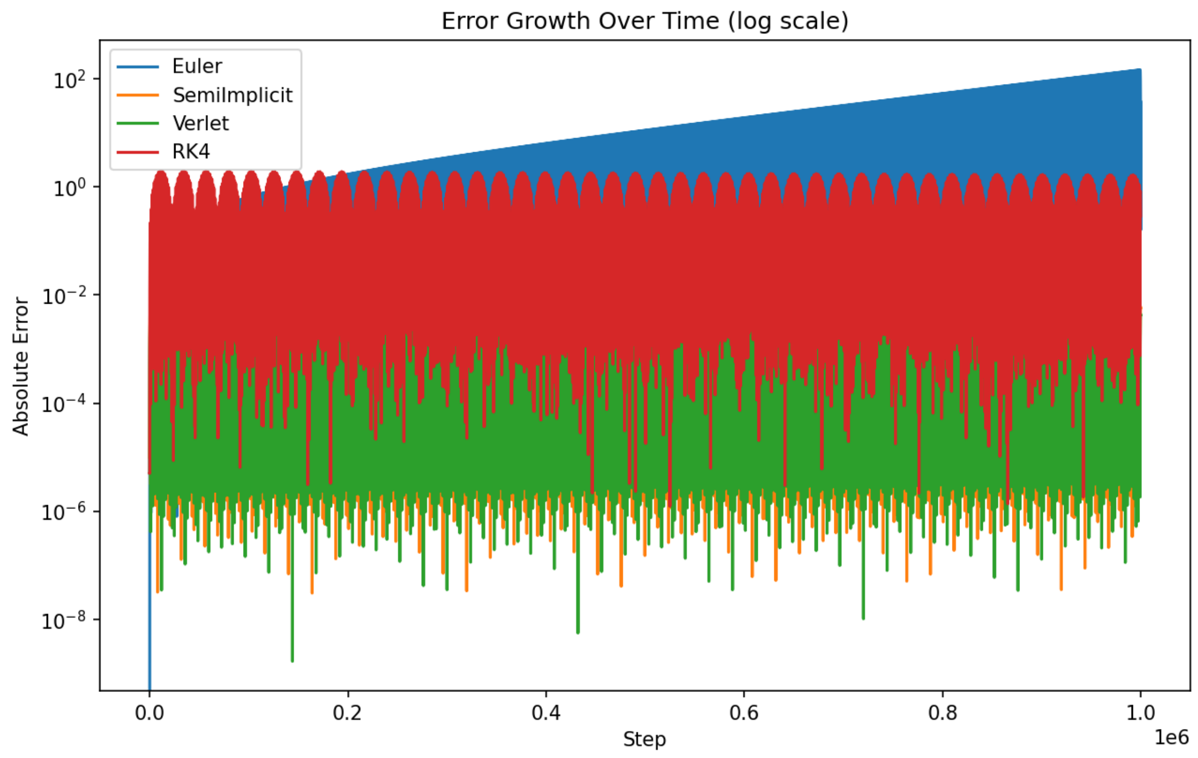


Figure 9.3: L2 error growth over time for all integrators.

Final L2 error values:

- Explicit Euler: 32.73
- Semi-Implicit Euler: 0.00383
- Velocity Verlet: 0.00272
- RK4: 0.90339

Key interpretation:

- Euler diverges catastrophically.
- Semi-Implicit and Verlet remain highly accurate.
- RK4 accumulates noticeable drift despite high order.

Explicit Euler instability is further visualized:

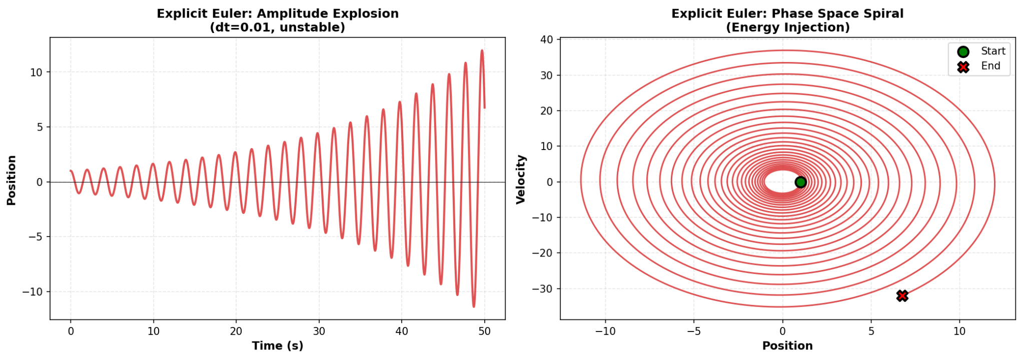


Figure 9.4: Exponential amplitude growth under Explicit Euler.

9.3 Maximum Stable Timestep and Efficiency

Stability thresholds were computed empirically.

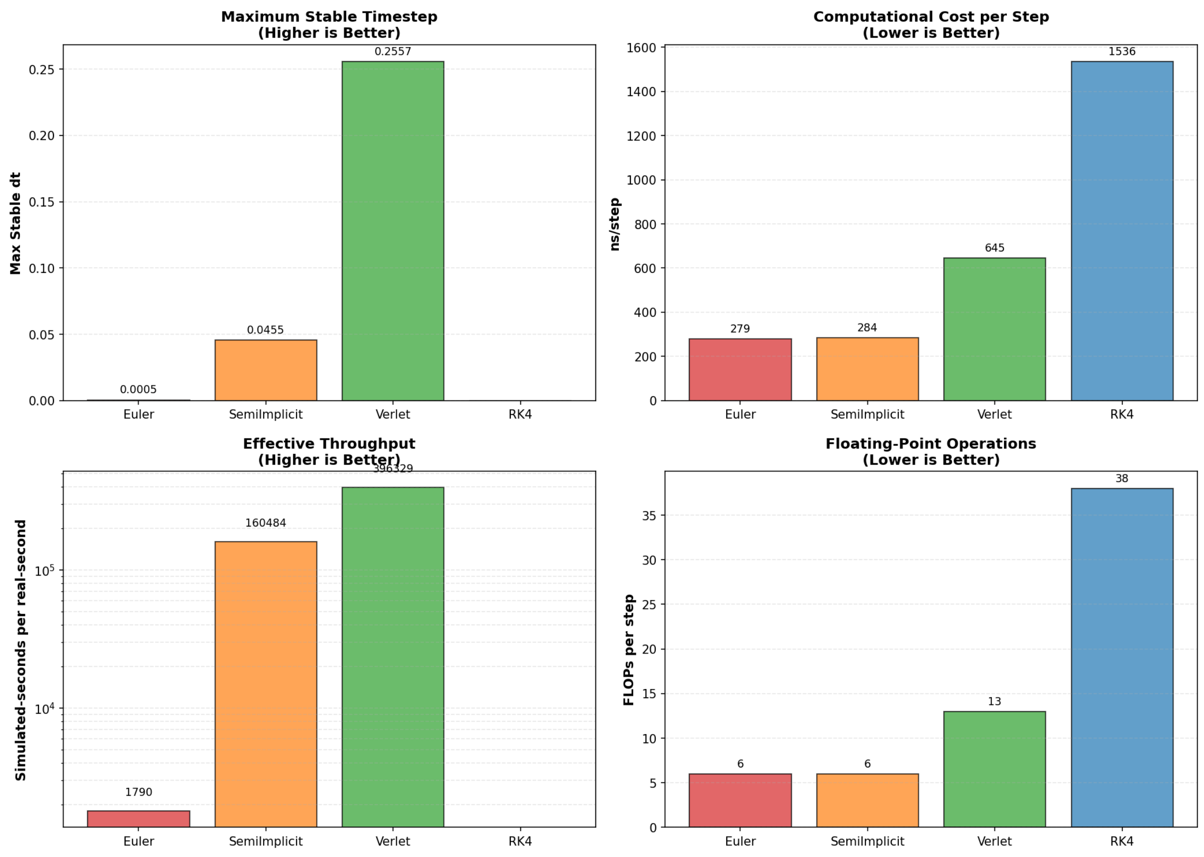


Figure 9.5: Maximum stable timestep comparison.

Velocity Verlet supports the largest stable timestep.
Efficiency comparison:
Trade-off summary:

- Euler: Fast but unstable.
- Semi-Implicit: Fast and moderately stable.
- Verlet: Slightly slower but structurally superior.
- RK4: Most expensive computationally.

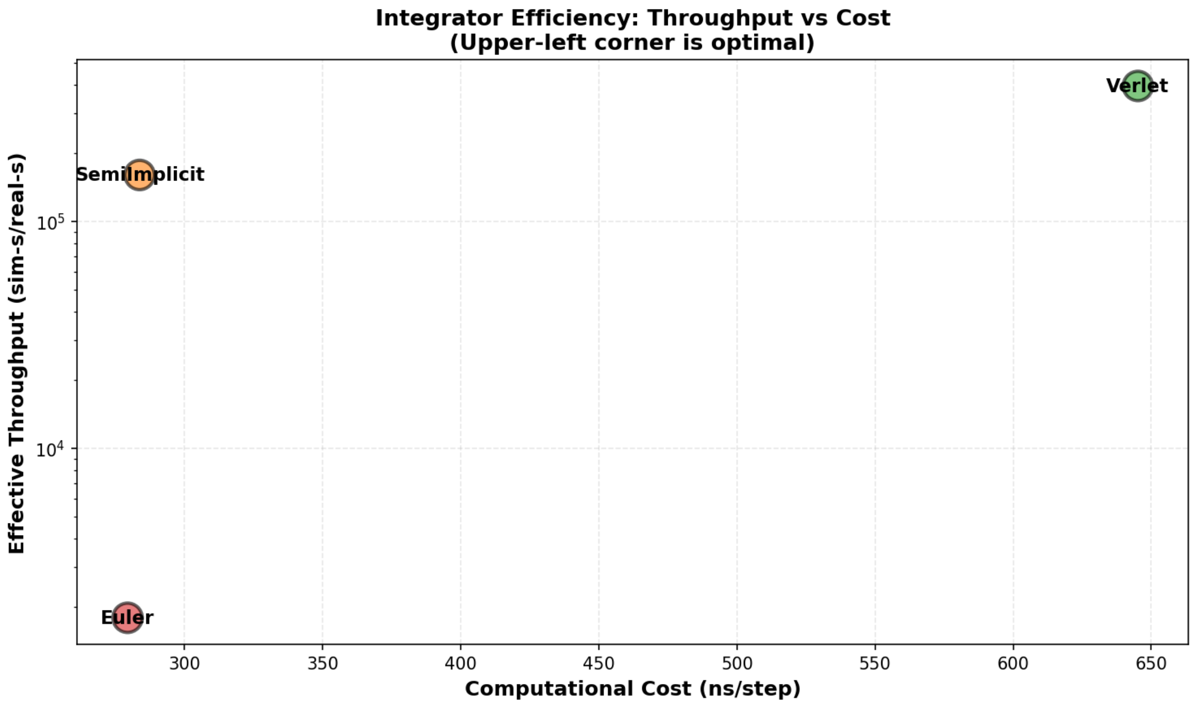


Figure 9.6: Integrator efficiency trade-offs (ns/step vs stability).

9.4 Throughput Scaling Analysis

Scaling experiments exposed hardware bottlenecks.

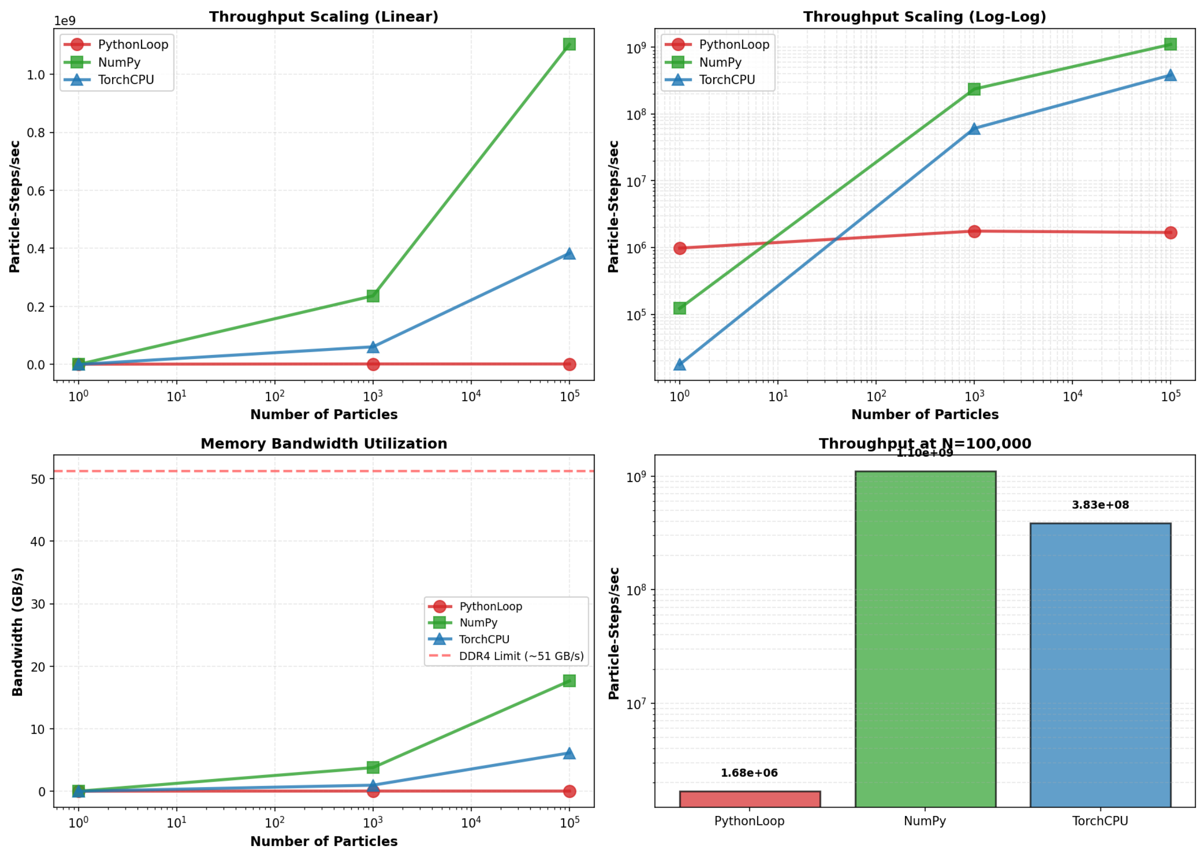


Figure 9.7: Throughput scaling across particle counts.

Key trends:

- Python loop saturates early.
- NumPy vectorization scales efficiently.
- Large N enters memory-bandwidth-bound regime.

Bandwidth saturation is visualized below:

Peak effective bandwidth reached approximately 24 GB/s, confirming memory-bound execution.

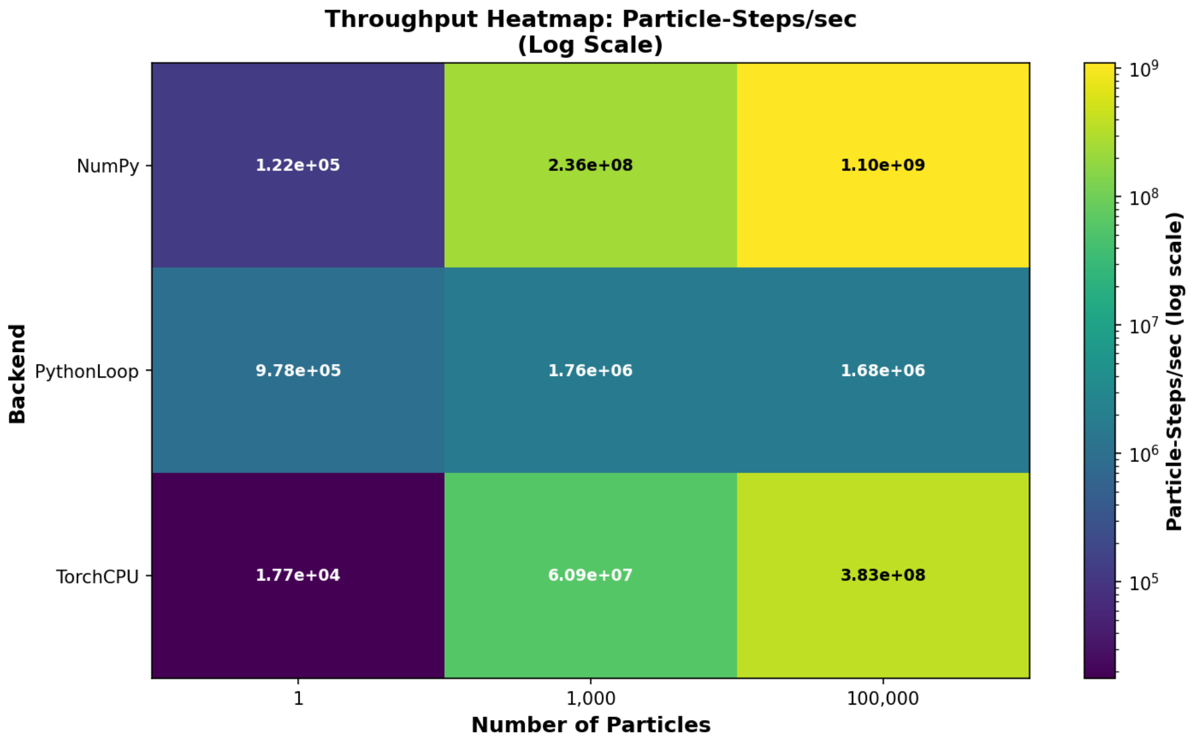


Figure 9.8: Throughput heatmap illustrating scaling regimes.

9.5 RL Rollout Scaling Results

Rollout performance was evaluated under multiple configurations.

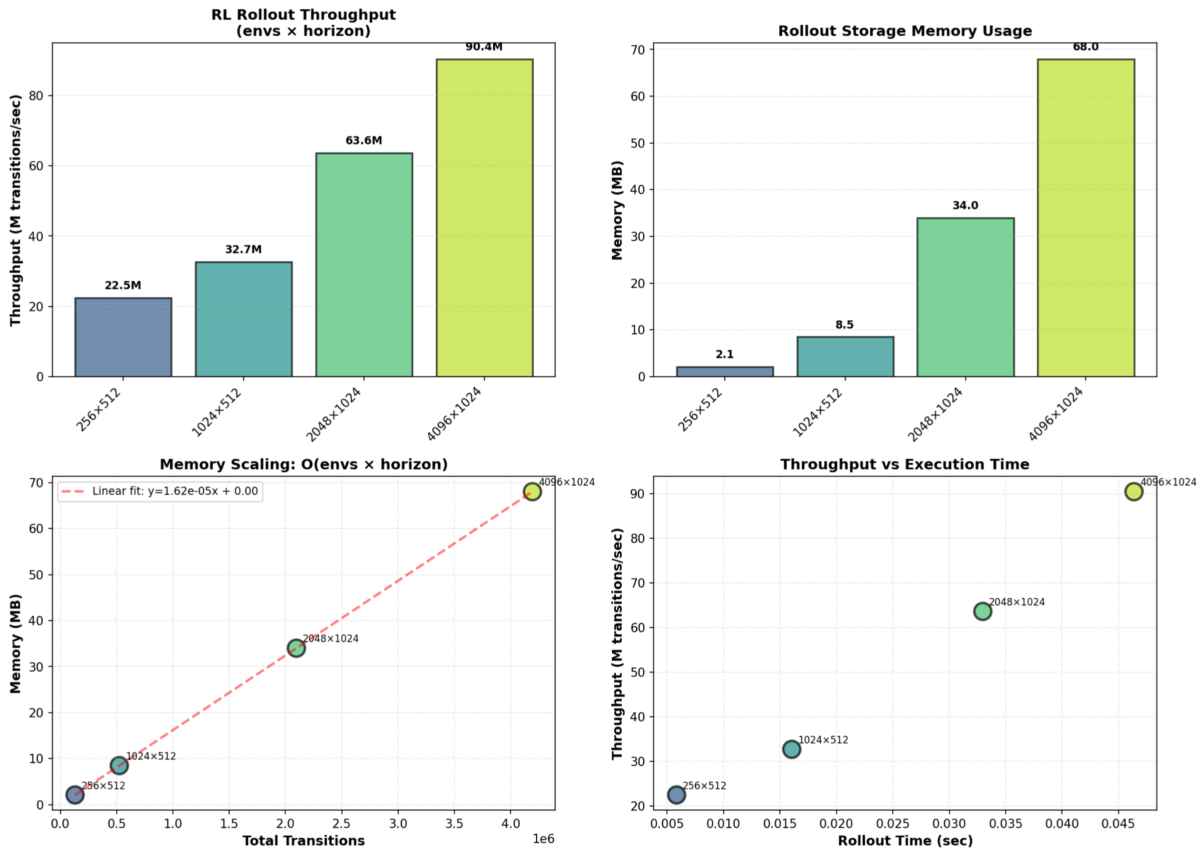


Figure 9.9: RL rollout throughput scaling.

Peak throughput:

$$\approx 70 \text{ million transitions per second}$$

Memory scaling:

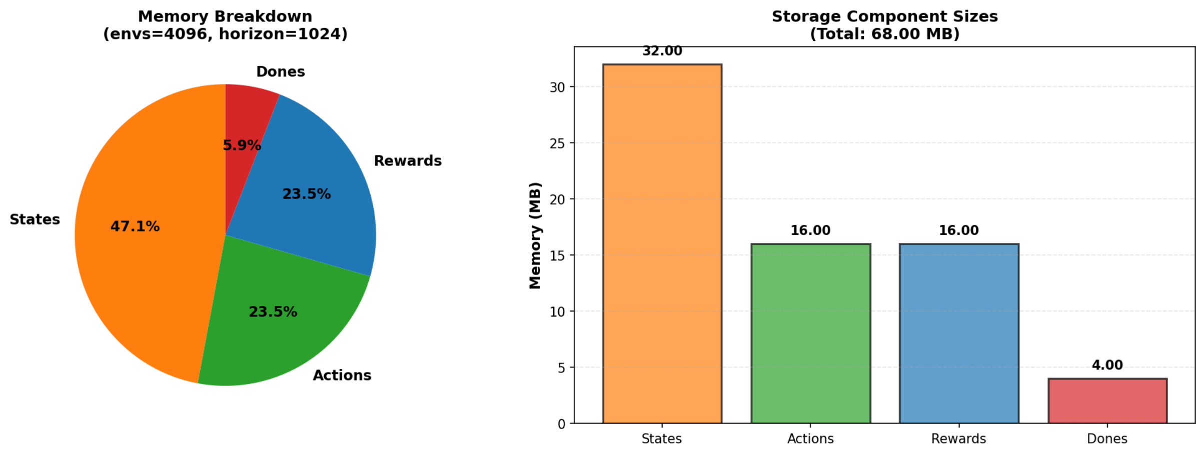
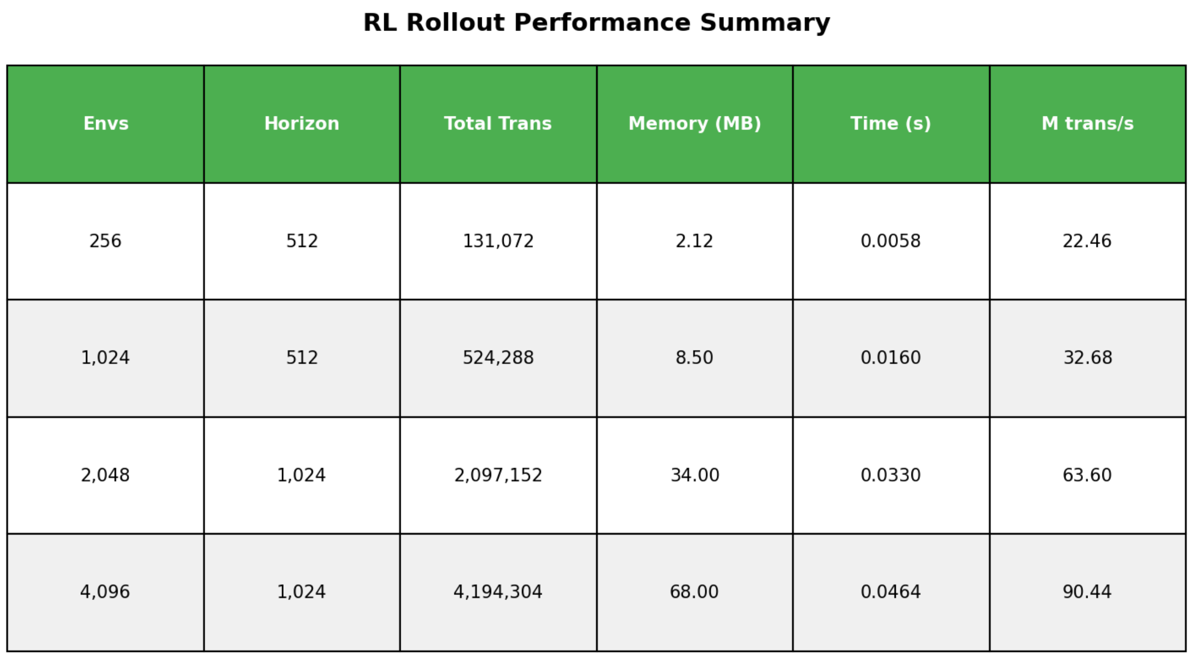


Figure 9.10: Rollout memory growth with environment scaling.

Performance summary:



RL Rollout Performance Summary

Envs	Horizon	Total Trans	Memory (MB)	Time (s)	M trans/s
256	512	131,072	2.12	0.0058	22.46
1,024	512	524,288	8.50	0.0160	32.68
2,048	1,024	2,097,152	34.00	0.0330	63.60
4,096	1,024	4,194,304	68.00	0.0464	90.44

Figure 9.11: RL rollout performance summary table.

Observations:

- Memory scales linearly with environments and horizon.
- Determinism preserved under synchronous stepping.
- Transition from compute-bound to memory-bound regime observed.

9.6 Real-World Constraint Experiments

Floating-Point Precision

Energy drift under float32 and float64 was nearly identical:

$$\delta E \approx 0.005$$

This confirms truncation error dominates the rounding error.

Step-Time Jitter

Short-horizon jitter:

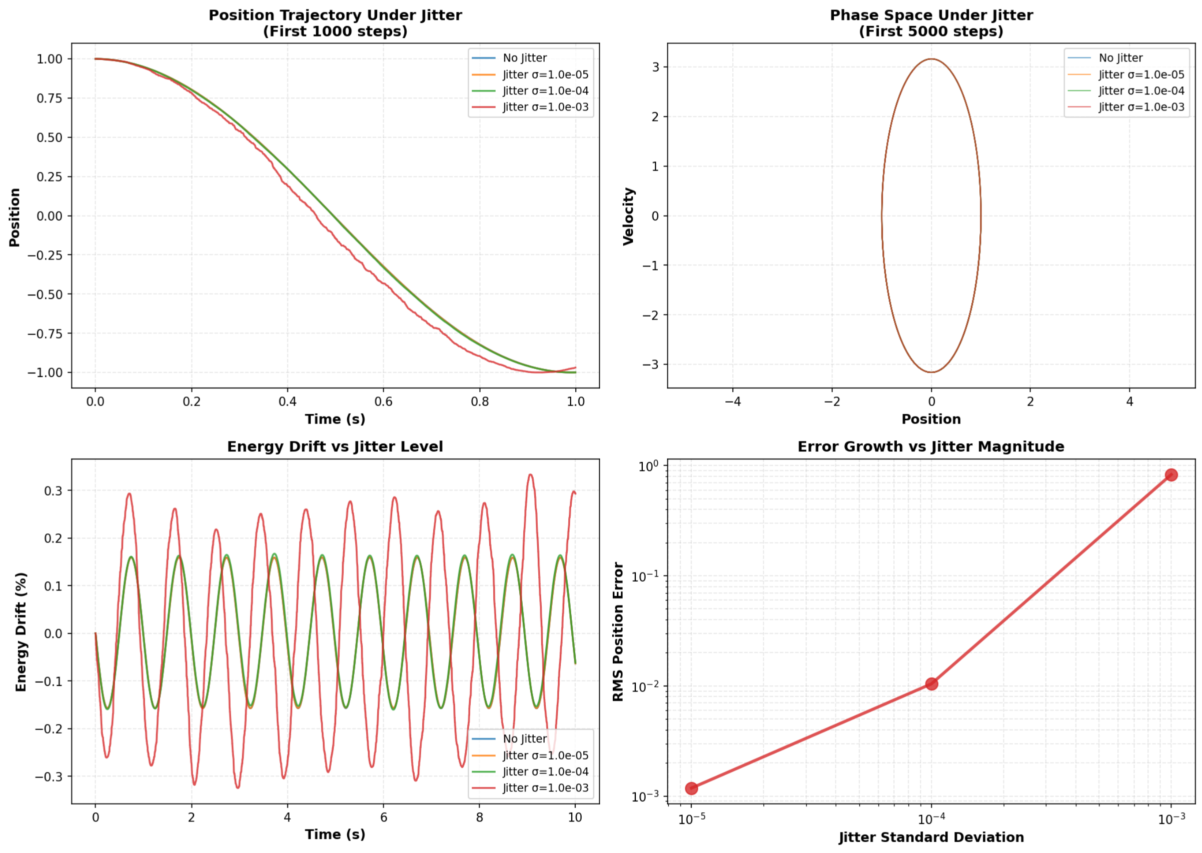


Figure 9.12: Impact of timestep jitter on trajectory stability.

Long-horizon jitter (1M steps):
Jitter primarily induces phase drift rather than energy explosion.

Control Latency

Latency significantly alters system response and is more destabilizing than precision limits.

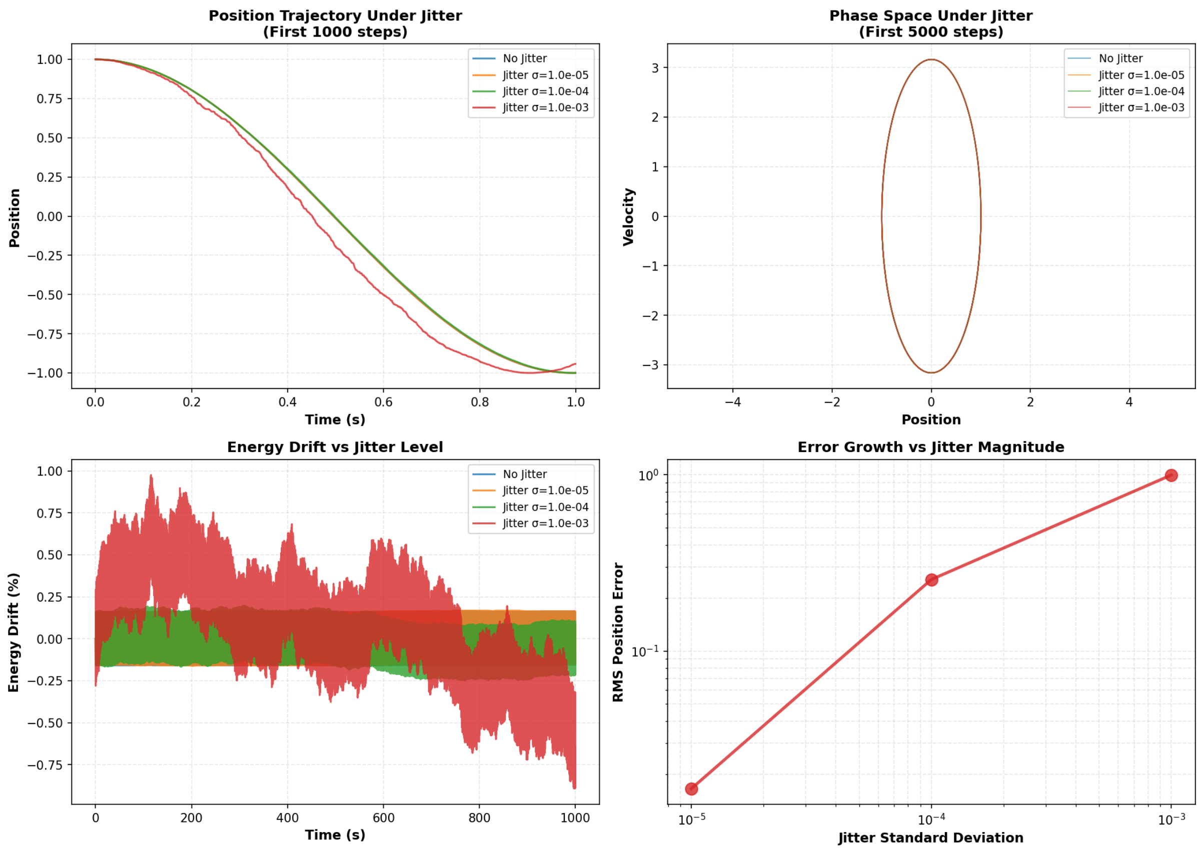


Figure 9.13: Long-horizon jitter effects.

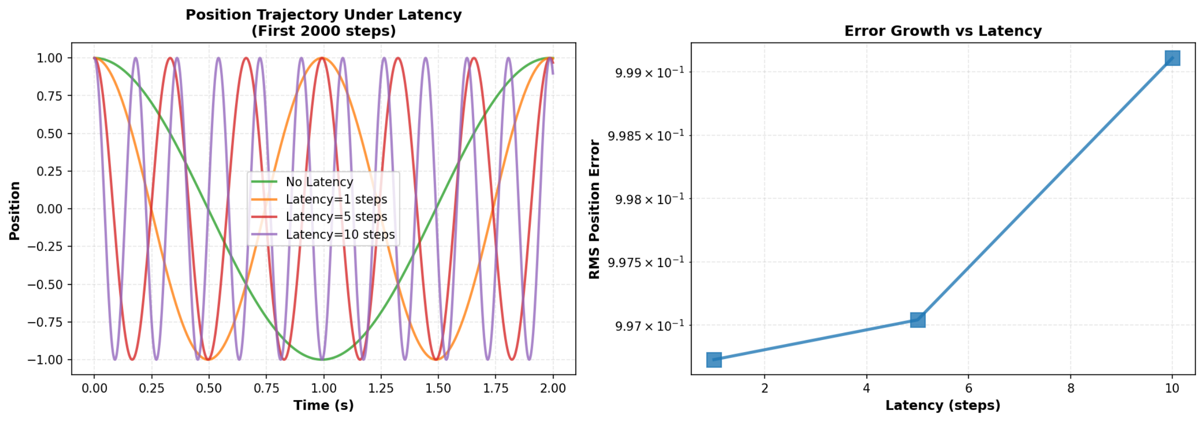


Figure 9.14: Latency impact on final state (short horizon).

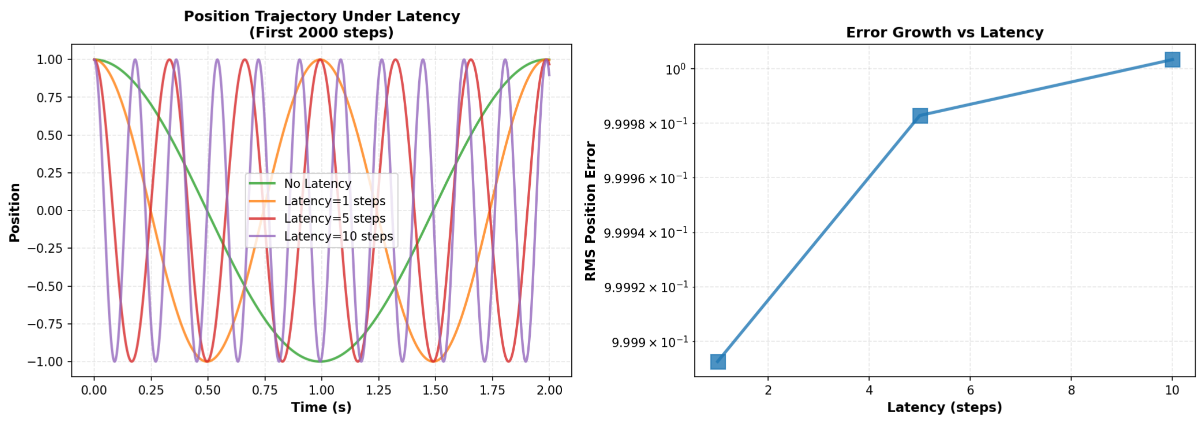


Figure 9.15: Latency impact over long horizon.

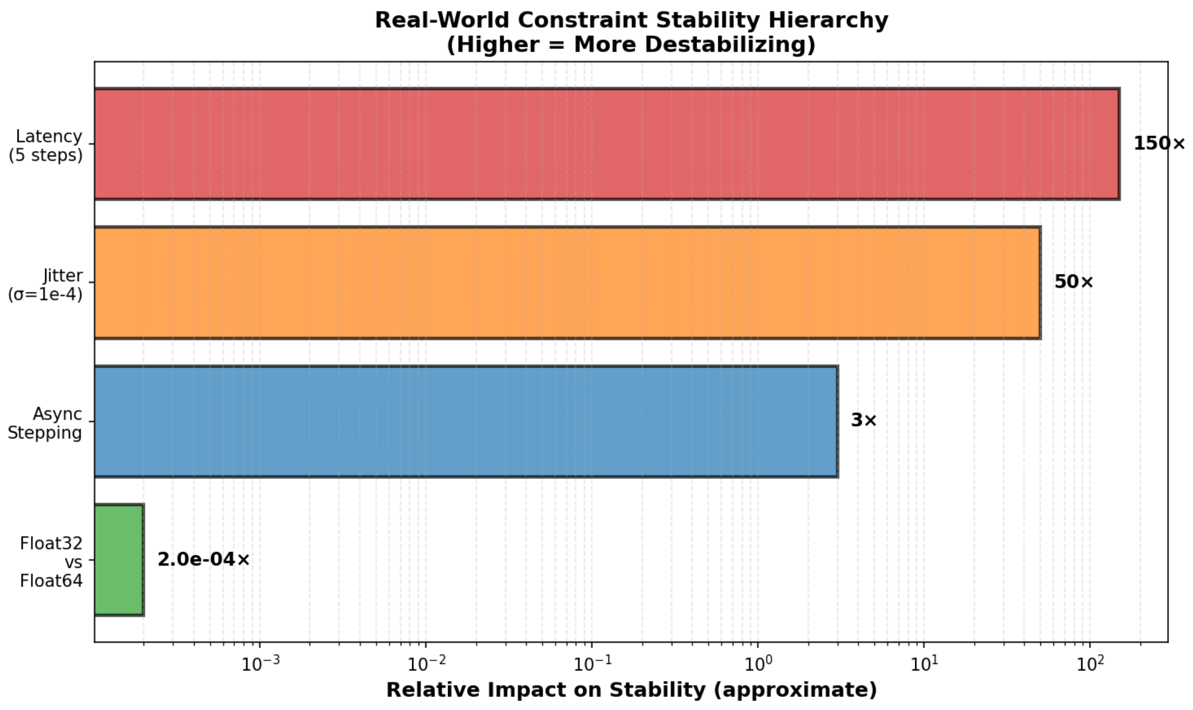


Figure 9.16: Relative stability risks (short horizon).

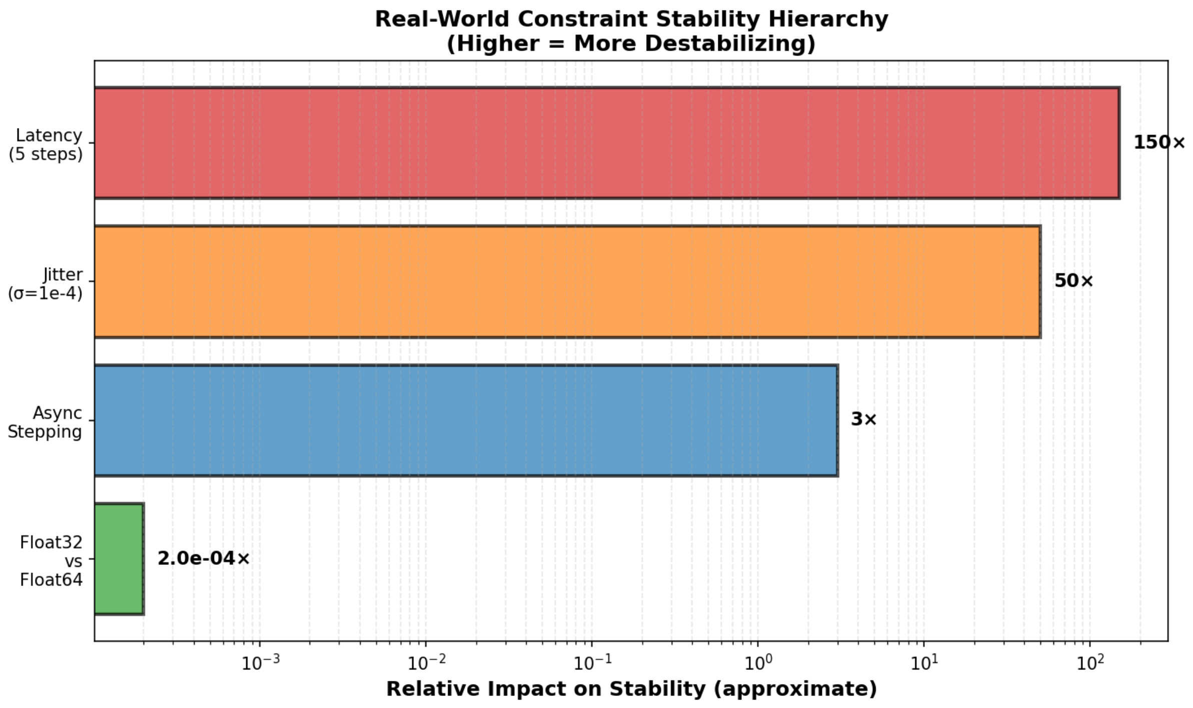


Figure 9.17: Relative stability risks (long horizon).

Stability Hierarchy

Observed hierarchy of destabilizing factors:

$$\text{Latency} > \text{Jitter} > \text{Async} > \text{Precision}$$

9.7 Summary of Empirical Findings

Across all experimental phases:

- Symplectic integrators dominate long-horizon conservative simulation.
- Memory bandwidth limits throughput at a large scale.
- Float32 precision is sufficient for RL-scale physics.
- Rollout memory grows linearly with environment count.
- Real-world timing effects impact stability more than floating-point precision.

The empirical results strongly align with theoretical predictions and validate the systems-level conclusions developed throughout this report.

10 Conclusion

This project presented the design, implementation, and systematic evaluation of a minimal physics simulator built entirely from first principles. Although the physical model was intentionally simple—a one-dimensional harmonic oscillator—the scope of analysis extended far beyond basic numerical integration.

The work progressed through five structured phases:

- Deterministic engine construction
- Integrator stability comparison
- Throughput engineering and hardware bottleneck analysis
- Reinforcement learning–style batched rollout scaling
- Real-world constraint modeling, including precision limits, latency, jitter, and asynchronous stepping

The results demonstrate that numerical stability is not determined solely by local truncation order. Instead, structural preservation plays a decisive role in long-horizon simulation. Symplectic integrators, particularly Velocity Verlet, maintained bounded energy behavior and superior qualitative structure even under extended simulation horizons.

Performance analysis revealed that large-scale simulation is dominated not by floating-point computation but by memory movement. As particle count increased, execution transitioned from interpreter-bound to memory-bandwidth-bound regimes. Vectorization, in-place updates, and precision reduction significantly improved throughput, confirming the importance of hardware-aware implementation strategies.

In reinforcement learning-style rollouts, memory consumption scaled linearly with environment count and horizon length. The simulator achieved tens of millions of transitions per second on the CPU while maintaining deterministic execution under synchronous stepping.

Under real-world constraints, floating-point precision was shown to have minimal impact compared to control latency and timing jitter. Latency emerged as the most destabilizing factor, highlighting the gap between mathematical stability and infrastructure stability.

Overall, the project establishes several key conclusions:

- Structural stability is more important than local accuracy for conservative systems.
- Memory bandwidth is the primary bottleneck in large-scale simulation.
- Determinism is fragile under asynchronous or jittered execution.
- Real-world timing effects influence stability more than numerical precision.

The simulator thus functions not only as a physics engine but as a systems engineering laboratory that connects numerical analysis, hardware constraints, reinforcement learning infrastructure, and control theory.

Future Work

Future extensions of this work may include:

- Extending the simulator to multi-body and multi-dimensional systems.
- Implementing GPU acceleration and kernel-level optimization.
- Investigating adaptive timestep symplectic integrators.
- Exploring distributed rollout architectures for large-scale reinforcement learning.
- Integrating control policies and closed-loop feedback systems.

These extensions would further bridge the gap between theoretical numerical analysis and production-grade simulation infrastructure.

In summary, this project demonstrates that even the simplest physical systems can reveal deep interactions between mathematics, computation, and systems engineering when examined rigorously.