Contents

1 Repo Implementation Guide v2 **3**

1.1 1. Reader Contract And Review Goal . . . 3
1.2 2. Physics-To-Code Invariants . . . 4
1.2.1 2.1 Pauli symbol invariant . . . 5
1.2.2 2.2 Qubit ordering invariant . . . 5
1.2.3 2.3 JW source-of-truth invariant . . . 5
1.2.4 2.4 Canonical PauliTerm source invariant . . . 5
1.2.5 2.5 Drive pass-through invariant . . . 5
1.2.6 2.6 Safe-test invariant . . . 5
1.3 3. Hamiltonian Assembly And JW Contracts . . . 6
1.4 4. Hartree-Fock Construction And Half-Filling Placement . . . 8
1.5 5. VQE Internals: Ansatz, Theta, And Inner Optimization . . . 12
1.5.1 5.1 What theta is in implementation terms . . . 13
1.5.2 5.2 How theta is consumed . . . 13
1.5.3 5.3 Inner optimization semantics . . . 13
1.5.4 5.4 What restarts do concretely . . . 13
1.5.5 5.5 Why this matters for artifact interpretation . . . 13
1.6 6. Trotter Dynamics Internals And Ordering Semantics . . . 15
1.6.1 6.1 What gets reordered . . . 16
1.6.2 6.2 Why native vs sorted exists . . . 16
1.6.3 6.3 Suzuki-2 absolute-time implementation . . . 16
1.6.4 6.4 Expected behavior with finer discretization . . . 16
1.6.5 6.5 Review implications . . . 16
1.7 7. Time-Dependent Drive Implementation . . . 17
1.7.1 7.1 Architecture split . . . 18
1.7.2 7.2 Why pass-through is strict . . . 18
1.7.3 7.3 Drive-enabled reference method . . . 18
1.7.4 7.4 Safe-test contract . . . 18
1.8 8. Exact Reference Semantics And exact_steps_multiplier . . . 20
1.8.1 8.1 Branch semantics in plain implementation terms . . . 21
1.8.2 8.2 Role of exact_steps_multiplier . . . 21
1.8.3 8.3 What to check in outputs . . . 21
1.8.4 8.4 Failure signatures . . . 21
1.9 9. Trajectory Observables: Formulas To Code . . . 22
1.9.1 9.1 Site-resolved occupancy . . . 22
1.9.2 9.2 Doublon . . . 22
1.9.3 9.3 Staggered order . . . 22
1.9.4 9.4 Energy channels . . . 22
1.9.5 9.5 Fidelity . . . 23
1.10 10. Plot Generation Internals And Meaning . . . 25
1.10.1 10.1 Plot data source contract . . . 25
1.10.2 10.2 Page families in hardcoded pipeline PDF . . . 25
1.10.3 10.3 Meaningful interpretation rules . . . 25
1.10.4 10.4 Common misreads to avoid . . . 25
1.10.5 10.5 Why recomputed audit plots were added . . . 26
1.11 11. Canonical Artifact Audits (L=2,3,4) . . . 27
1.11.1 11.1 L=2 static heavy . . . 27
1.11.2 11.2 L=3 static heavy . . . 28
1.11.3 11.3 L=4 drive-enabled dyn . . . 28
1.11.4 11.4 Cross-case summary . . . 28
1.12 12. Minimal Compare/Qiskit Validation Role . . . 32
1.12.1 12.1 What compare runner should do . . . 33
1.12.2 12.2 What qiskit baseline contributes . . . 33
1.12.3 12.3 What this section intentionally does not do . . . 33

1.13 13. Extension Safety: Safe And High-Risk Edits . . . 34
1.13.1 13.1 Practical pre-merge checklist . . . 34
1.13.2 13.2 Key targeted function anchors . . . 34
1.14 14. Ultra-Brief Run Appendix . . . 36
1.15 Appendix A: Figure Atlas (Condensed) . . . 37
1.15.1 A.1 Repository Structure Map . . . 37
1.15.2 A.2 Internal Import DAG . . . 38
1.15.3 A.3 Operator Layer Object Model . . . 38
1.15.4 A.4 Qubit Ordering Convention . . . 39
1.15.5 A.5 JW Mapping Flow . . . 40
1.15.6 A.6 Hardcoded Pipeline Flow . . . 40
1.15.7 A.7 Qiskit Baseline Flow . . . 41
1.15.8 A.8 Shared Drive Backend . . . 42
1.15.9 A.9 Compare Fan-Out/Fan-In . . . 42
1.15.10 A.10 Amplitude 6-Run Workflow . . . 43
1.15.11 A.11 Safe-Test Logic . . . 44
1.15.12 A.12 Artifact Contract Map . . . 44
1.15.13 A.13 Test Contract Coverage . . . 45
1.15.14 A.14 Extension Playbook . . . 46
1.15.15 A.15 HF Occupancy Table L2 blocked . . . 47
1.15.16 A.16 HF Occupancy Table L3 blocked . . . 48
1.15.17 A.17 HF Occupancy Table L4 blocked . . . 49
1.15.18 A.18 HF Occupancy Table L2 interleaved . . . 50
1.15.19 A.19 HF Occupancy Table L3 interleaved . . . 51
1.15.20 A.20 HF Occupancy Table L4 interleaved . . . 52
1.15.21 A.21 Bit Index Examples . . . 53
1.15.22 A.22 Theta Vector Layout . . . 53
1.15.23 A.23 Optimizer Restart Flow . . . 54
1.15.24 A.24 Term Traversal vs Ordering . . . 55
1.15.25 A.25 Suzuki-2 Anatomy . . . 55
1.15.26 A.26 Drive Waveform . . . 56
1.15.27 A.27 Drive Spatial Patterns . . . 57
1.15.28 A.28 Reference Propagator Split . . . 57
1.15.29 A.29 Trajectory Schema Map . . . 58
1.15.30 A.30 Formula Legend Energy . . . 59
1.15.31 A.31 Formula Legend Fidelity . . . 59
1.15.32 A.32 Formula Legend Occupancy . . . 60
1.15.33 A.33 Artifact L2 Energy Audit . . . 61
1.15.34 A.34 Artifact L2 Site0 Audit . . . 61
1.15.35 A.35 Artifact L3 Energy Audit . . . 62
1.15.36 A.36 Artifact L3 Site0 Audit . . . 62
1.15.37 A.37 Artifact L4 Drive Energy Audit . . . 63
1.15.38 A.38 Artifact L4 Site0/Fidelity Audit . . . 63
1.15.39 A.39 Artifact L4 Error Audit . . . 64
1.15.40 A.40 Case Comparison Summary . . . 64
1.15.41 A.41 Function Evidence Map . . . 65
1.15.42 A.42 Function Call Graph . . . 65
1.15.43 A.43 Invariant Evidence . . . 66
1.15.44 A.44 Canonical Metrics Table . . . 66
1.15.45 A.45 Quality Gate Summary . . . 67
1.15.46 A.46 Plot Meaning Map . . . 67
1.16 Appendix B: Target Function Evidence (Condensed) . . . 68
1.17 Appendix C: Canonical Metrics Snapshot (Condensed) . . . 68

1 Repo Implementation Guide v2

Implementation-first guide for `Holstein_test` and its active subrepo `Fermi-Hamil-JW-VQE-TROTTER-PIPELINE`.

Audience assumption: you already know the physics and math, and you want confidence that the implementation is faithful, numerically sane, and interpretable.

1.1 1. Reader Contract And Review Goal

This document is intentionally narrow: it is an implementation review guide, not a physics tutorial and not a CLI handbook. Every major claim is tied back to concrete code paths and emitted artifacts.

What this guide optimizes for:

- What the code *actually computes* at runtime.
- Where assumptions are enforced in code.
- Why each plot channel has meaning (and what it does not mean).
- How to audit correctness without guessing from naming.

What this guide intentionally de-emphasizes:

- Long derivations of Hubbard/JW theory.
- End-user command catalogs.
- Generic software engineering process advice not specific to this repository.

Canonical artifacts used for worked examples:

- `artifacts/json/H_L2_static_t1.0_U4.0_S64_heavy.json`
- `artifacts/json/H_L3_static_t1.0_U4.0_S128_heavy.json`
- `artifacts/json/H_L4_vt_t1.0_U4.0_S256_dyn.json`

These three were selected because they span small and medium lattice size behavior, static and drive-enabled behavior, and very different parameter-vector sizes in VQE.

The review stance throughout is practical:

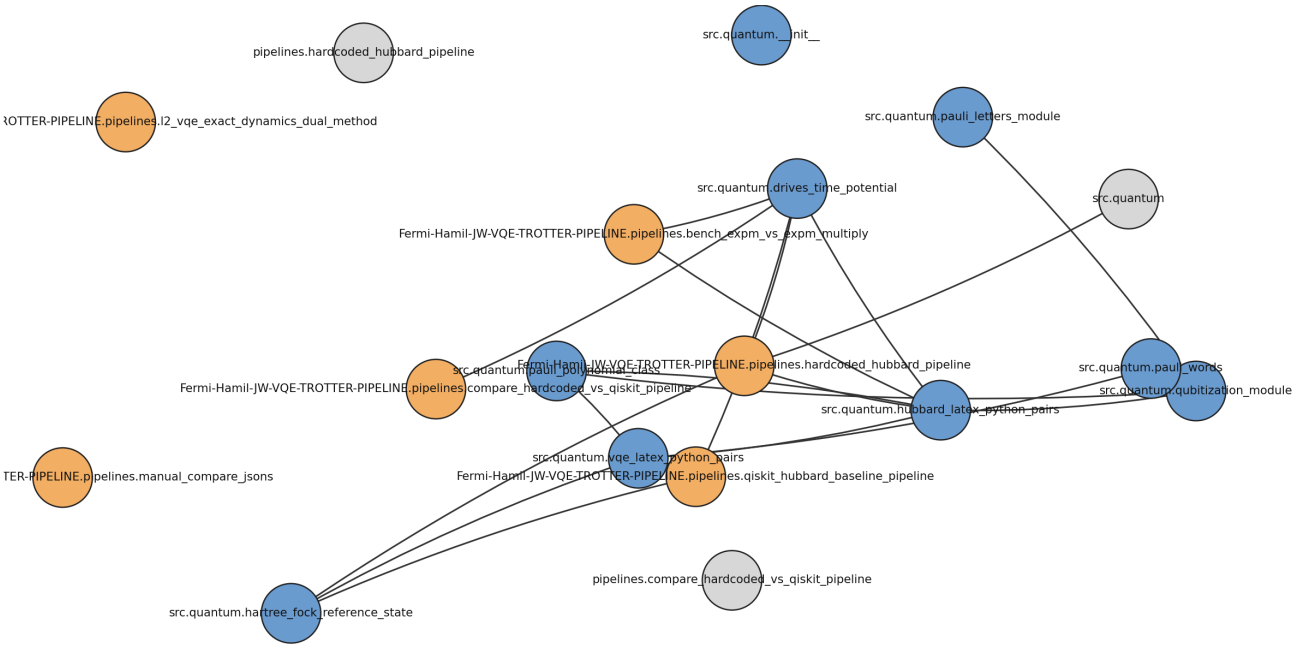
1. State the mathematical quantity.
2. State the exact implementation path.
3. State how it appears in JSON and plots.
4. State one or more failure signatures.

If you only read five sections, read sections 4, 5, 6, 9, and 10 first. Those cover HF placement, VQE optimizer semantics, trotter ordering semantics, observable computation, and how figures are assembled.

Repository Structure Map
Implementation workspace and artifact zones

```
workspace/
|- AGENTS.md
|- src/quantum/
|  |- 9 python modules (operator/core/vqe/drive)
|- test_*.py
|  |- 8 implementation-contract tests
|- Fermi-Hamil-JW-VQE-TROTTER-PIPELINE/
|  |- pipelines/
|  |  |- 6 python runners
|  |  |- 3 shell orchestrators
|  |- artifacts/json/
|  |- artifacts/pdf/
|  |- docs/repo_guide_assets/*.png
|  |- docs/repo_implementation_guide.md
|  |- scripts/generate_repo_guide_assets.py
|  |- scripts/build_repo_implementation_guide.sh
```

Internal Import DAG
src/quantum + pipeline module relationships



1.2 2. Physics-To-Code Invariants

The repository has a few invariants that are effectively non-negotiable. Most implementation mistakes eventually reduce to violating one of these.

1.2.1 2.1 Pauli symbol invariant

Internal algebra uses `e/x/y/z`, not `I/X/Y/Z`. Conversion is boundary-only. This matters because mixed symbol conventions in intermediate code silently break equality checks for labels and dictionary keys.

1.2.2 2.2 Qubit ordering invariant

Pauli word strings are written left-to-right as `q_(n-1)...q_0`; qubit 0 is rightmost in string form. Statevector indexing still uses integer bit positions where qubit 0 is least significant bit.

This dual statement is easy to remember but easy to accidentally violate when translating between string positions and bit operations.

1.2.3 2.3 JW source-of-truth invariant

New code should not hand-roll JW ladders. It should reuse helper functions from the polynomial layer. The practical reason is not style: it avoids drift between "equivalent-looking" definitions when edge cases are hit (identity terms, sign conventions, ordering changes).

1.2.4 2.4 Canonical PauliTerm source invariant

`PauliTerm` is canonical in `qubitization_module.py`. Compatibility aliases exist for interop but should not become new, divergent implementations.

1.2.5 2.5 Drive pass-through invariant

The compare pipeline routes drive arguments through. It does not reinterpret drive physics. This is a correctness property because compare mode should not perturb either HC or QK semantics.

1.2.6 2.6 Safe-test invariant

A drive-enabled run with `A=0` should numerically match the no-drive run within threshold. This is effectively a regression sentinel for the drive code path itself.

Qubit Ordering and Bit Indexing Convention

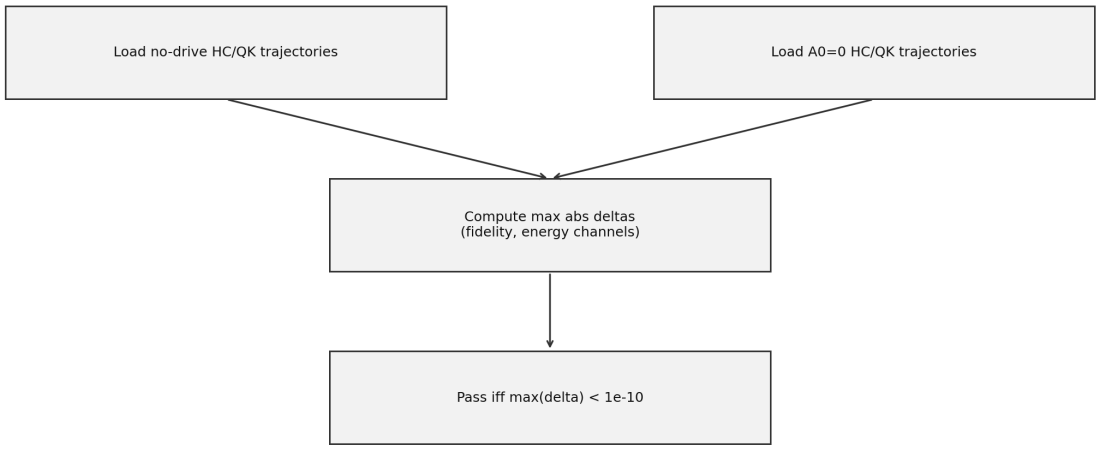
Pauli strings and statevector indexing must agree exactly

```
Example (nq = 6)

Pauli label string:   e z x y e z
                      | | | | | |
Character index:      0 1 2 3 4 5
Physical qubit:       5 4 3 2 1 0

State bitstring formatting in outputs: q_(n-1)...q_0
Basis index in statevector: index = sum_q bit_q * 2^q

Consequence:
- String position p maps to qubit q = nq - 1 - p.
- q0 is least significant bit in basis index arithmetic.
- All occupancy extraction uses (idx >> q) & 1.
```

Invariant Evidence Snippets
Detected invariant contracts with line-level evidence

```
drive_flag_passthrough: found=True
  - Fermi-Hamil-JW-VQE-TROTTER-PIPELINE/pipelines/compare_hardcoded_vs_qiskit_pipeline.py:1169  def _build_drive_args(args: argparse.Namespace) -> list[str]:
  - Fermi-Hamil-JW-VQE-TROTTER-PIPELINE/pipelines/compare_hardcoded_vs_qiskit_pipeline.py:1203  def _build_drive_args_with_amplitude(args: argparse.Namespace, amplitude: float

drive_safe_test_threshold: found=True
  - Fermi-Hamil-JW-VQE-TROTTER-PIPELINE/pipelines/compare_hardcoded_vs_qiskit_pipeline.py:1200  _SAFE_TEST_THRESHOLD: float = 1e-10

jw_mapping_source_of_truth: found=True
  - AGENTS.md:27  - `fermion_plus_operator(repr_mode="JW", nq, j)`
  - AGENTS.md:28  - `fermion_minus_operator(repr_mode="JW", nq, j)`

pauli_string_qubit_ordering: found=True
  - AGENTS.md:21  - **left-to-right = q_(n-1) ... q_0**
  - AGENTS.md:22  - **qubit 0 is rightmost character**

pauli_symbol_convention_exyz: found=True
  - AGENTS.md:16  - Use `e/x/y/z` internally (`e` = identity)

reference_method_metadata: found=True
  - Fermi-Hamil-JW-VQE-TROTTER-PIPELINE/pipelines/hardcoded_hubbard_pipeline.py:51  reference_method_name,
  - Fermi-Hamil-JW-VQE-TROTTER-PIPELINE/pipelines/hardcoded_hubbard_pipeline.py:906  The JSON metadata records ``reference_method`` and

static_vs_driven_reference_split: found=True
  - Fermi-Hamil-JW-VQE-TROTTER-PIPELINE/pipelines/qiskit_hubbard_baseline_pipeline.py:1213  if not has_drive:
  - Fermi-Hamil-JW-VQE-TROTTER-PIPELINE/pipelines/qiskit_hubbard_baseline_pipeline.py:1232  if not has_drive:
```

Safe-Test Invariant Logic
A=0 drive trajectory must match no-drive trajectory within threshold



1.3 3. Hamiltonian Assembly And JW Contracts

Implementation path (hardcoded and qiskit both rely on the same conceptual decomposition):

$$ H = H_t + H_U + H_v = \sum_j c_j P_j $$

where each `P_j` is represented as a Pauli label under the repository convention.

In code, this appears as:

- construction of polynomial terms from hopping/onsite/potential logic,
- conversion into label-coefficient map,
- optional term ordering policy (`native` vs `sorted`) for trotter traversal,
- matrix assembly for exact/reference channels.

The important implementation distinction:

- the *Hamiltonian definition* (set of terms and coefficients) is separate from
- the *trotter traversal order* (sequence used to exponentiate finite-step factors).

When you hear "reordering," it means sequence of exponentials for finite-step propagation, not changing coefficients in the model.

1.3.0.1 Number operator contract

Number operators obey

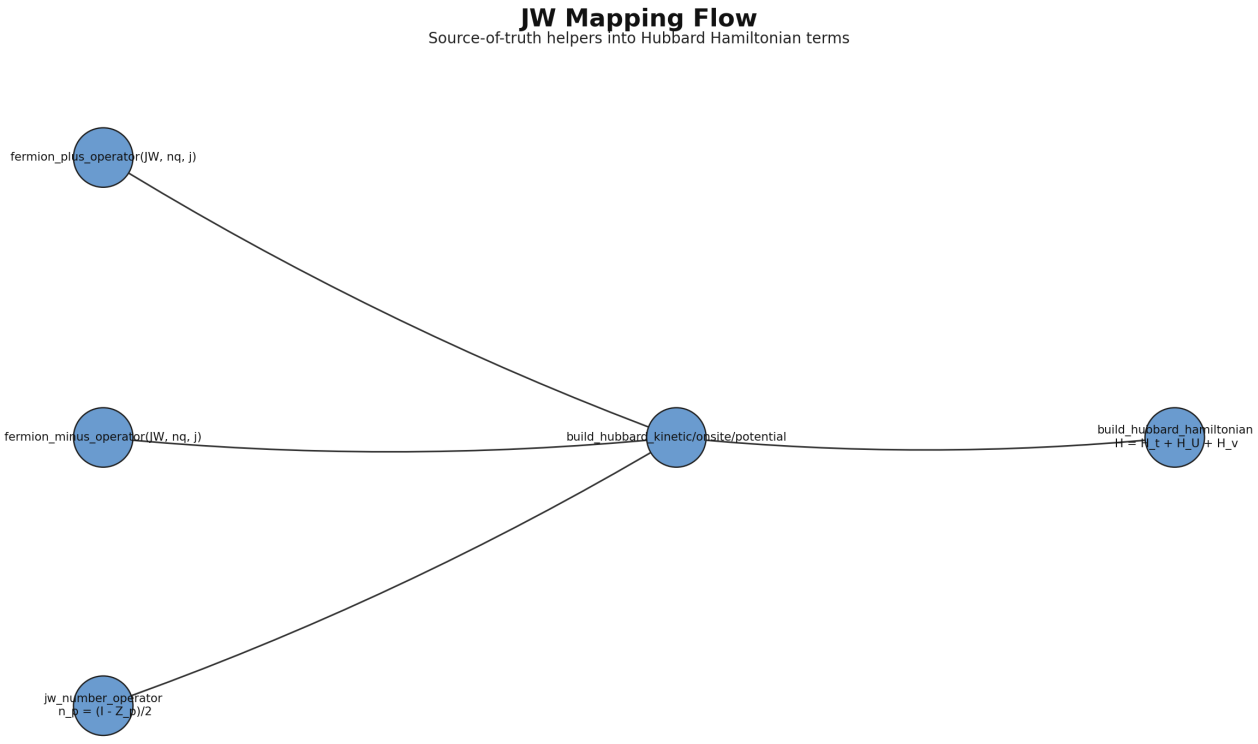
$$ \hat n_p = \frac{I - Z_p}{2} $$

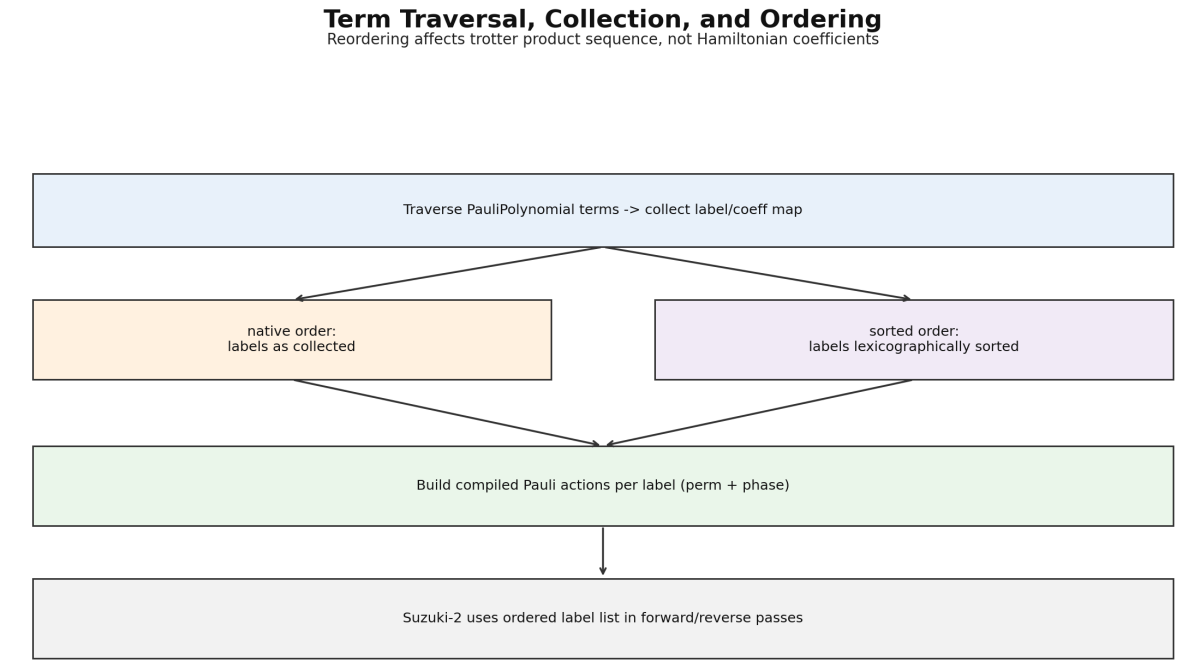
with `Z_p` positioned using the same string/bit convention used everywhere else.

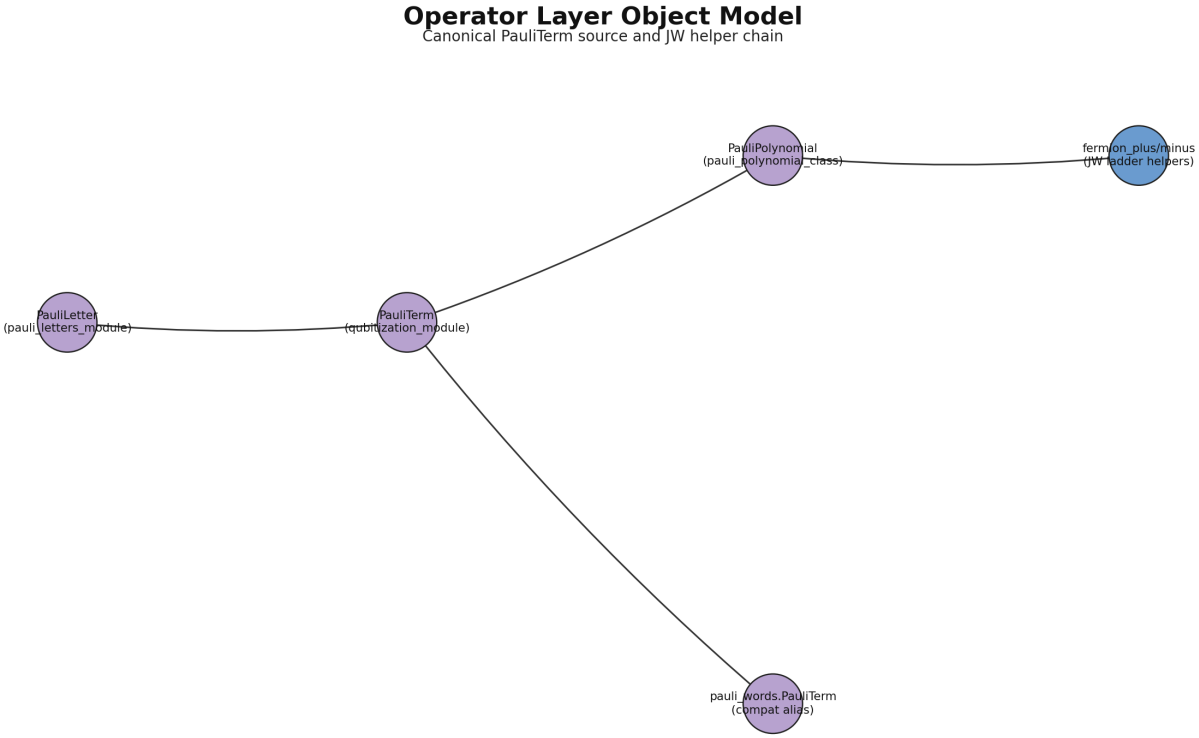
If this convention is broken in one place, downstream signs in site occupation and doublon channels become inconsistent even when energy looks close.

1.3.0.2 Practical audit checklist for Hamiltonian path

- verify term count and coefficient map non-emptiness in payload,
- verify reference sanity block in JSON,
- verify static channels are time-stationary when expected,
- verify no accidental symbol-case drift (`e/x/y/z` vs `I/X/Y/Z`) in intermediate maps.







1.4 4. Hartree-Fock Construction And Half-Filling Placement

This is one of the most misunderstood implementation details, so this section is explicit.

For `L` sites, the repository uses `N_q = 2L` spin-orbital qubits. At half filling,

$$ N_{\uparrow} = \left\lceil \frac{L}{2} \right\rceil, \qquad N_{\downarrow} = \left\lfloor \frac{L}{2} \right\rfloor. $$

The HF state is not discovered by solving a separate mean-field equation in this code path. It is a deterministic occupation pattern used as a reference state in VQE construction and optional initial state source.

The exact qubit positions for up/down occupancy depend on indexing mode:

- `blocked`: up modes first, then down modes,
- `interleaved`: up/down alternate by site.

Bitstring convention in outputs is `q_(N_q-1)...q_0`, but occupancy extraction in simulation uses integer bit position `q` with shifts/masks.

If you want to audit this in code terms:

- check the HF bitstring helper,
- check HF statevector creation from that bitstring,
- check mapping function used by site-resolved observable extraction.

Those three together define "where each electron is placed" for initialization and subsequent diagnostics.

The six HF tables below should be read as implementation truth tables, not conceptual cartoons.

HF Occupancy Map (L=2, ordering=blocked)

Half-filling placement of spin orbitals onto qubit bits

site i	qubit q(i,up)	occ_up	qubit q(i,dn)	occ_dn
0	0	1	2	1
1	1	0	3	0

n_up=1, n_dn=1, N_q=4, bitstring q_(Nq-1)...q_0 = 0101

HF Occupancy Map (L=3, ordering=blocked)

Half-filling placement of spin orbitals onto qubit bits

site i	qubit q(i,up)	occ_up	qubit q(i,dn)	occ_dn
0	0	1	3	1
1	1	1	4	0
2	2	0	5	0

n_up=2, n_dn=1, N_q=6, bitstring q_(Nq-1)...q_0 = 001011

HF Occupancy Map (L=4, ordering=blocked)

Half-filling placement of spin orbitals onto qubit bits

site i	qubit q(i,up)	occ_up	qubit q(i,dn)	occ_dn
0	0	1	4	1
1	1	1	5	1
2	2	0	6	0
3	3	0	7	0

n_up=2, n_dn=2, N_q=8, bitstring q_(Nq-1)...q_0 = 00110011

HF Occupancy Map (L=2, ordering=interleaved)

Half-filling placement of spin orbitals onto qubit bits

site i	qubit q(i,up)	occ_up	qubit q(i,dn)	occ_dn
0	0	1	1	1
1	2	0	3	0

n_up=1, n_dn=1, N_q=4, bitstring q_(Nq-1)...q_0 = 0011

HF Occupancy Map (L=3, ordering=interleaved)

Half-filling placement of spin orbitals onto qubit bits

site i	qubit q(i,up)	occ_up	qubit q(i,dn)	occ_dn
0	0	1	1	1
1	2	1	3	0
2	4	0	5	0

n_up=2, n_dn=1, N_q=6, bitstring q_(Nq-1)...q_0 = 000111

HF Occupancy Map (L=4, ordering=interleaved)

Half-filling placement of spin orbitals onto qubit bits

site i	qubit q(i,up)	occ_up	qubit q(i,dn)	occ_dn
0	0	1	1	1
1	2	1	3	1
2	4	0	5	0
3	6	0	7	0

n_up=2, n_dn=2, N_q=8, bitstring q_(Nq-1)...q_0 = 00001111

Bit Index and Place-Value Examples

How basis index integer maps to qubit occupancies

```
Basis index extraction examples (N_q = 6)

idx = 13  -> binary(q5..q0)=001101
q0 = ((13 >> 0) & 1) = 1
q1 = ((13 >> 1) & 1) = 0
q2 = ((13 >> 2) & 1) = 1
q3 = ((13 >> 3) & 1) = 1
q4 = ((13 >> 4) & 1) = 0
q5 = ((13 >> 5) & 1) = 0

place value: idx = sum_q bit_q * 2^q = 1*2^0 + 0*2^1 + 1*2^2 + 1*2^3

site occupation expectation in code:
    n_up[site] += up_bit * prob(idx)
    n_dn[site] += dn_bit * prob(idx)

where prob(idx) = |psi[idx]|^2 and up_bit/dn_bit are extracted with shifts + masks.
```

1.5 5. VQE Internals: Ansatz, Theta, And Inner Optimization

The hardcoded path minimizes

$$ E(\theta) = \langle \psi(\theta)|H|\psi(\theta)\rangle $$

with

$$ |\psi(\theta)\rangle = U(\theta)|\psi_{\text{ref}}\rangle. $$

1.5.1 5.1 What theta is in implementation terms

`theta` is a real vector, one scalar per ansatz generator instance per repetition layer.

- It is not one global scalar.
- It is not "one or two parameters."
- Dimension is `num_parameters = reps * len(base_terms)`.

1.5.2 5.2 How theta is consumed

The prepare-state loop iterates over reps and base terms, applying one exponential per `theta[k]` and incrementing `k`. In this code path, each generator polynomial is exponentiated via term-wise Pauli rotations.

1.5.3 5.3 Inner optimization semantics

By default in the hardcoded pipeline configuration, optimizer method is `COBYLA` (SciPy path if available). Restarts mean independent initial points are sampled and optimized; final winner is the restart with lowest achieved energy.

This is not gradient-descent on an analytic gradient object. It is derivative-free under COBYLA by default, with objective evaluations routed through state preparation + expectation evaluation.

1.5.4 5.4 What restarts do concretely

For each restart `r`:

1. sample random initial point `x0` with seeded RNG,
2. optimize `E(theta)` from `x0`,
3. compare final energy to incumbent best,
4. keep best `(energy, theta, restart-id)`.

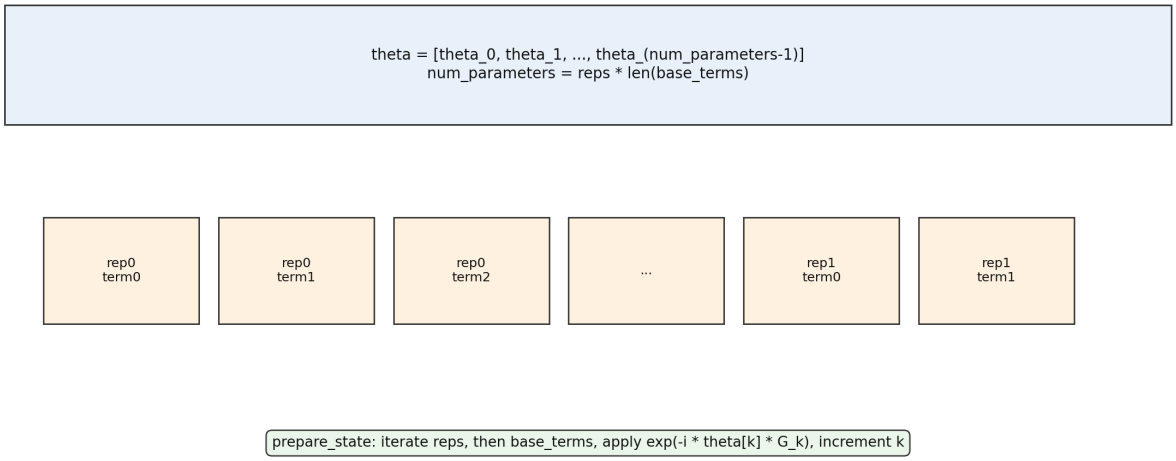
1.5.5 5.5 Why this matters for artifact interpretation

When VQE energies differ slightly between comparable runs, first check:

- same seed,
- same restart count,
- same maxiter and method,
- same ansatz depth (`reps`) and parameter count,
- same initial-state source used for trajectory branch.

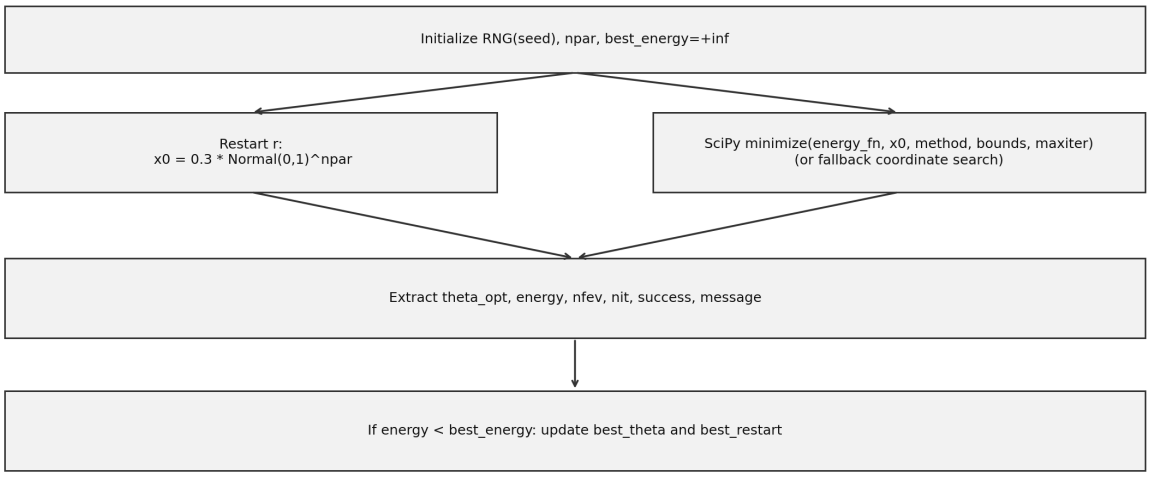
Theta Vector Layout and Generator Assignment

Hardcoded UCCSD ansatz: theta is a real vector indexed by term order across reps



Inner Optimizer and Restart Selection

energy_fn(theta)=<psi(theta)|H|psi(theta)>; keep restart with lowest final energy



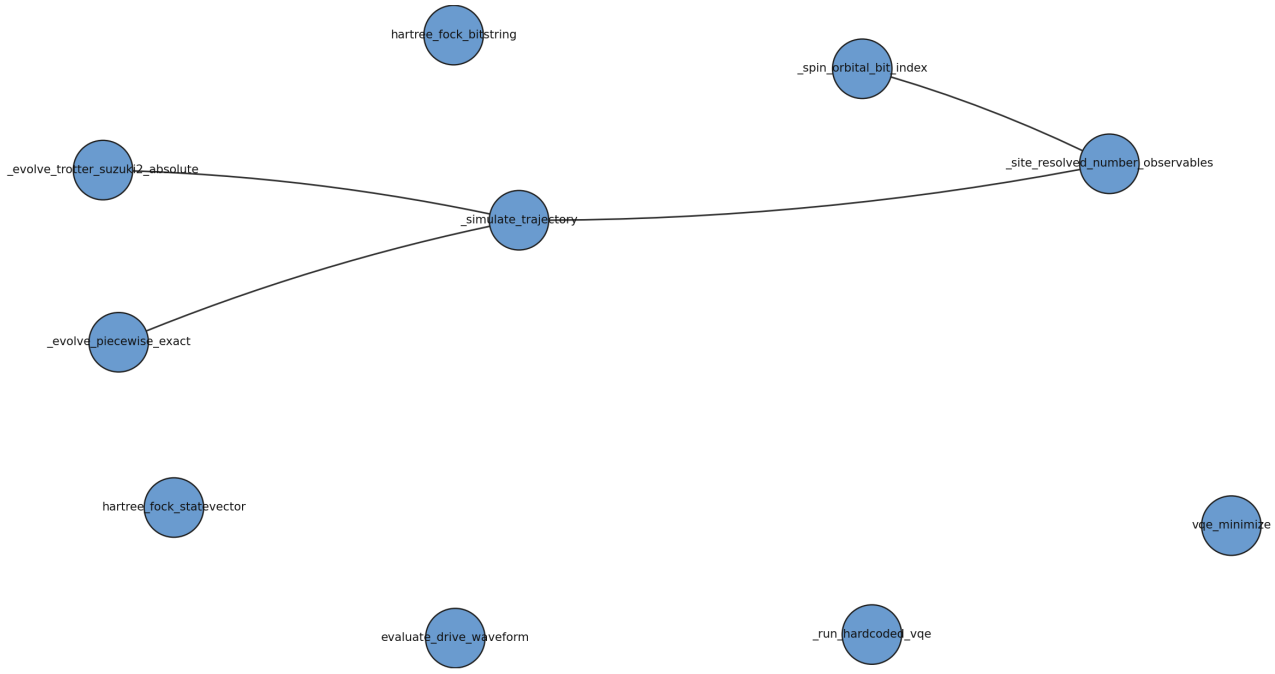
Function Evidence Map

Target implementation functions with file and line spans

_evolve_trotter_suzuki2_absolute	Fermi-Hamil-JW-VQE-TROTTER-PIPELINE/pipelines/hardcoded_hubbard_pipeline.py	L321-391
_spin_orbital_bit_index	Fermi-Hamil-JW-VQE-TROTTER-PIPELINE/pipelines/hardcoded_hubbard_pipeline.py	L430-436
_site_resolved_number_observables	Fermi-Hamil-JW-VQE-TROTTER-PIPELINE/pipelines/hardcoded_hubbard_pipeline.py	L439-461
_run_hardcoded_vqe	Fermi-Hamil-JW-VQE-TROTTER-PIPELINE/pipelines/hardcoded_hubbard_pipeline.py	L500-577
_evolve_piecewise_exact	Fermi-Hamil-JW-VQE-TROTTER-PIPELINE/pipelines/hardcoded_hubbard_pipeline.py	L875-1033
_simulate_trajectory	Fermi-Hamil-JW-VQE-TROTTER-PIPELINE/pipelines/hardcoded_hubbard_pipeline.py	L1036-1289
evaluate_drive_waveform	src/quantum/drives_time_potential.py	L137-170
hartree_fock_bitstring	src/quantum/hartree_fock_reference_state.py	L101-112
hartree_fock_statevector	src/quantum/hartree_fock_reference_state.py	L115-135
vqe_minimize	src/quantum/vqe_latex_python_pairs.py	L689-794

Function Call Graph (Targeted)

Call edges among targeted implementation functions



1.6 6. Trotter Dynamics Internals And Ordering Semantics

Trotterization in this repository is explicit product-form propagation on the statevector. The point that trips people up is the role of order.

For finite step size `dt`, different term sequences generally produce different local errors when terms do not commute.

The implementation allows explicit control via term-order policy.

1.6.1 6.1 What gets reordered

The reordered object is the traversal list of Pauli labels used in

$$ \prod_j e^{-i c_j P_j dt} $$

(or symmetric forward/reverse variant), not the set of Hamiltonian coefficients itself.

1.6.2 6.2 Why native vs sorted exists

- `native` preserves collection order from term traversal.
- `sorted` enforces deterministic lexical order.

The repository often defaults to sorted for reproducibility and easier cross-run comparison.

1.6.3 6.3 Suzuki-2 absolute-time implementation

Drive-enabled evolution evaluates time-dependent coefficients at sampled times and applies the symmetric sequence per sub-step. In static mode this collapses to coefficient-constant behavior.

1.6.4 6.4 Expected behavior with finer discretization

As `dt` decreases (more steps at fixed final time), order sensitivity shrinks. In the infinite-step limit both orderings converge to exact evolution (assuming stable numerical implementation).

1.6.5 6.5 Review implications

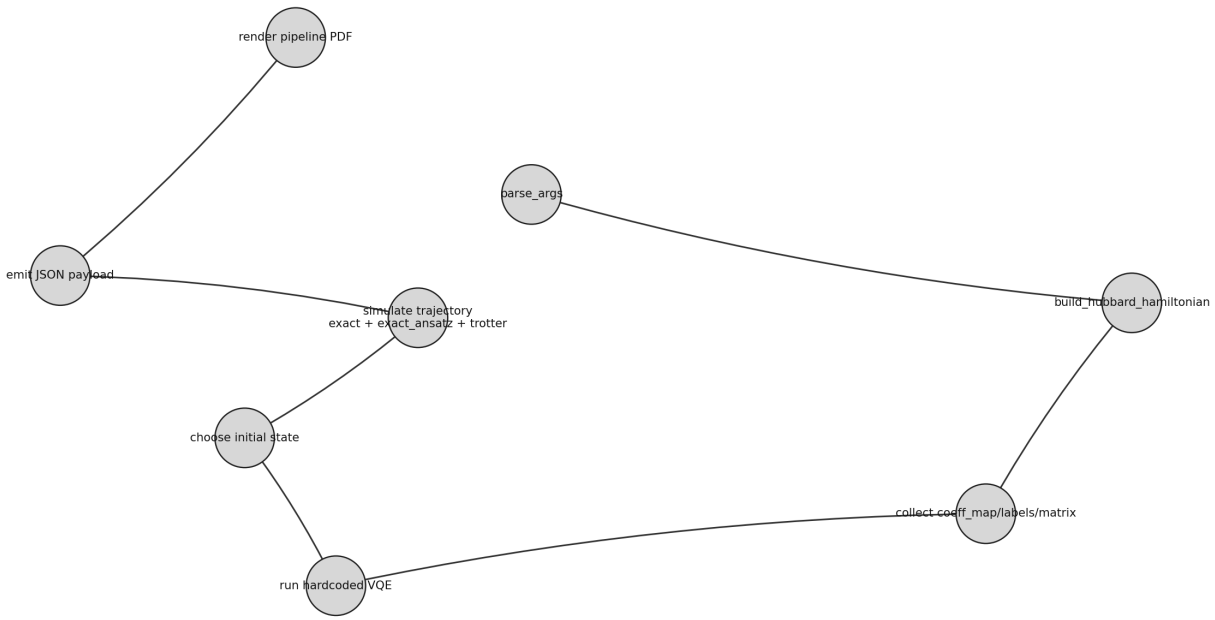
If two runs disagree and everything else matches, check:

- term order policy,
- trotter steps,
- drive time sampling mode,
- reference method branch,
- normalization diagnostics.

Suzuki-2 Step Anatomy
Forward/reverse product structure used by trotter evolution

```
Suzuki-2 step with N trotter slices (dt = t/N)

    U_S2(dt) = [prod_j exp(-i c_j P_j dt/2)] [prod_j^rev exp(-i c_j P_j dt/2)]

Implementation shape:
1) choose ordered_labels (native or sorted)
2) for each slice k:
     a) evaluate drive coefficients at sampling time
     b) forward pass over ordered labels, half-angle
     c) reverse pass over ordered labels, half-angle
3) normalize diagnostics / continue

Ordering impact:
- finite dt: non-commuting term sequence changes local error terms
- dt -> 0: ordering sensitivity shrinks (Trotter limit)
```

Hardcoded Pipeline Flow
Execution sequence in pipelines/hardcoded_hubbard_pipeline.py



1.7 7. Time-Dependent Drive Implementation

Drive model implemented:

$$ v(t)=A\sin(\omega t+\phi)\exp\!\left(-\frac{(t-t_0)^2}{2\bar t^2}\right) $$

with spatial modulation

$$ v_j(t)=s_j\,v(t). $$

1.7.1 7.1 Architecture split

- waveform and spatial pattern logic: quantum drive module,
- CLI plumbing and pass-through: pipeline argument handling,
- exact/trotter integration: shared kernel path in trajectory simulation.

1.7.2 7.2 Why pass-through is strict

Compare pipeline should not reinterpret physics knobs. It should route the same drive settings into both HC and QK sub-runs so discrepancies are attributable to implementation differences, not argument rewriting.

1.7.3 7.3 Drive-enabled reference method

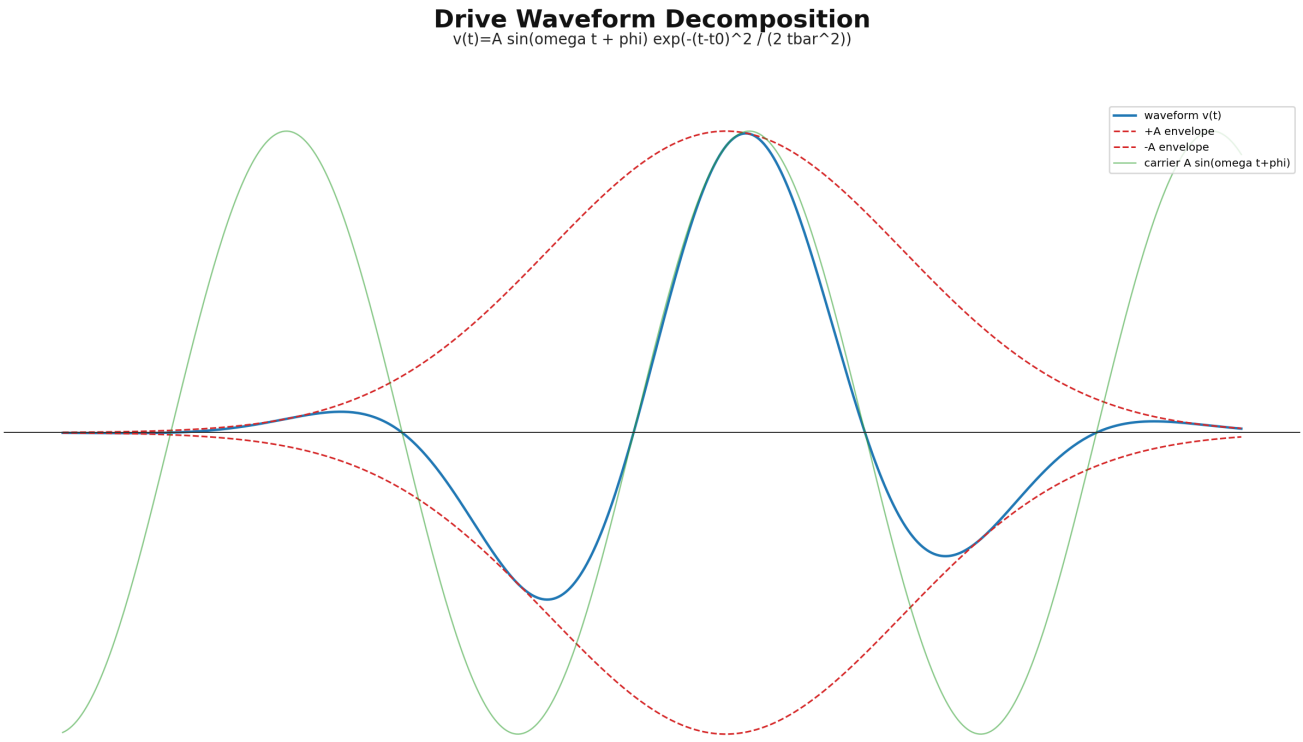
In drive mode, reference evolution uses piecewise exact propagation with refined step count:

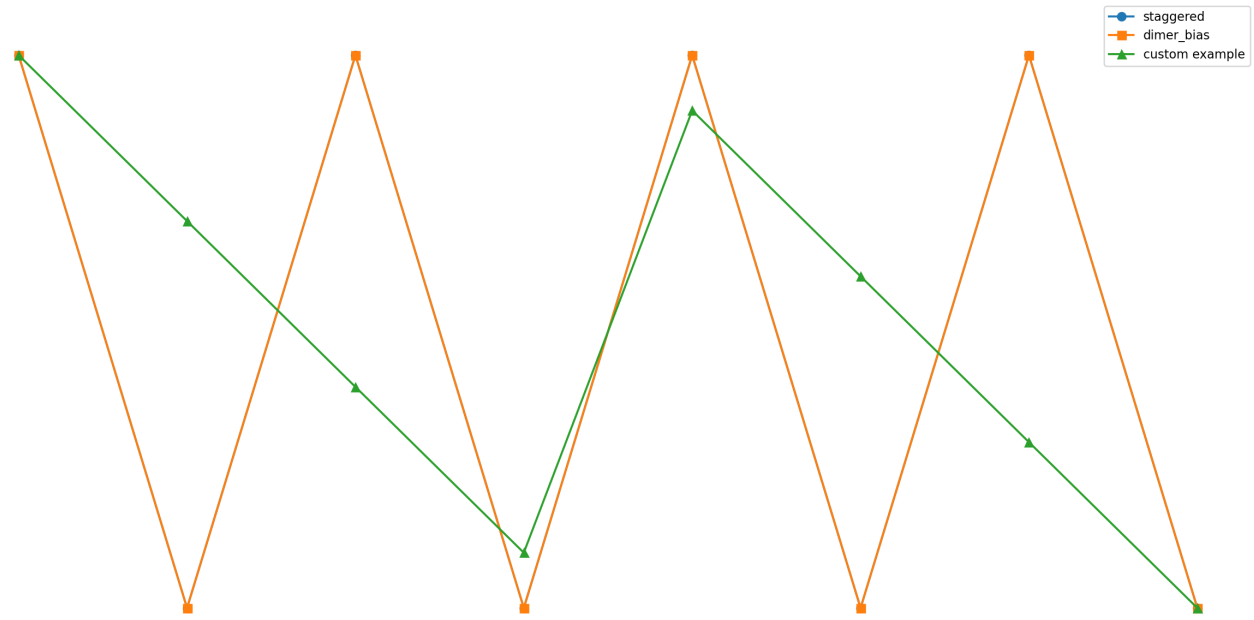
$$ N_{\text{ref}} = \text{exact\_steps*multiplier} *times N{\text{trotter}}. $$

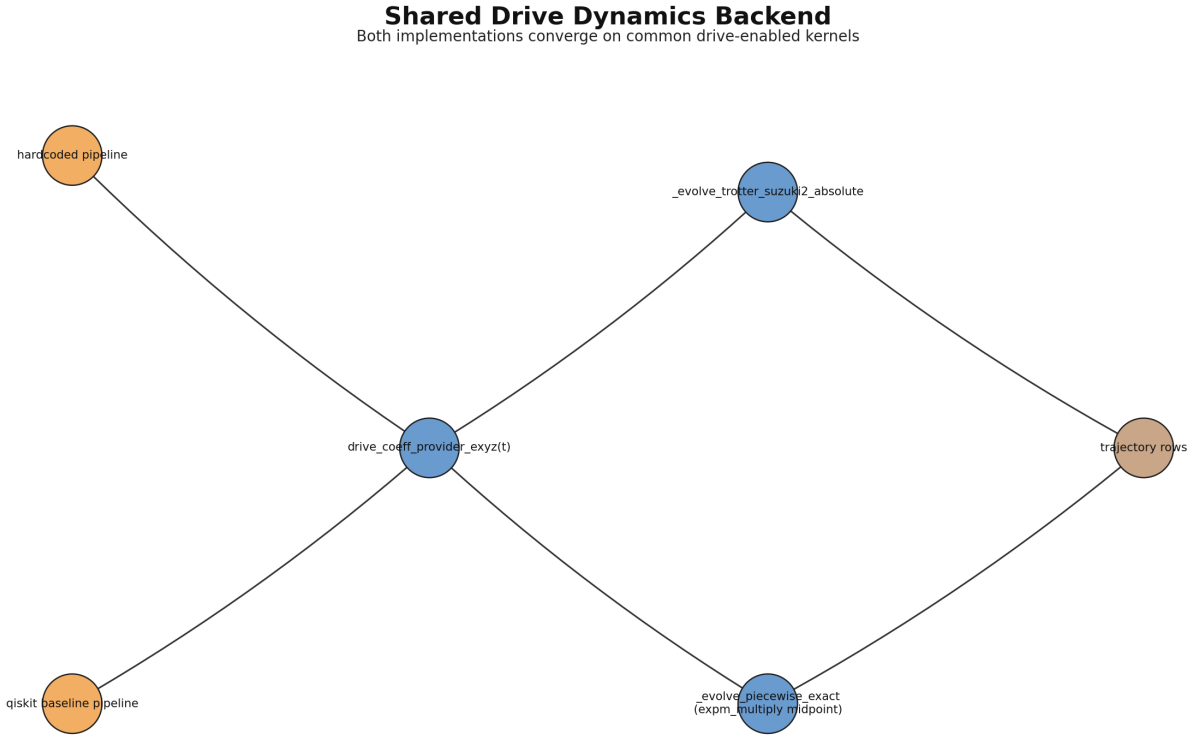
In no-drive mode, static eigendecomposition reference is used.

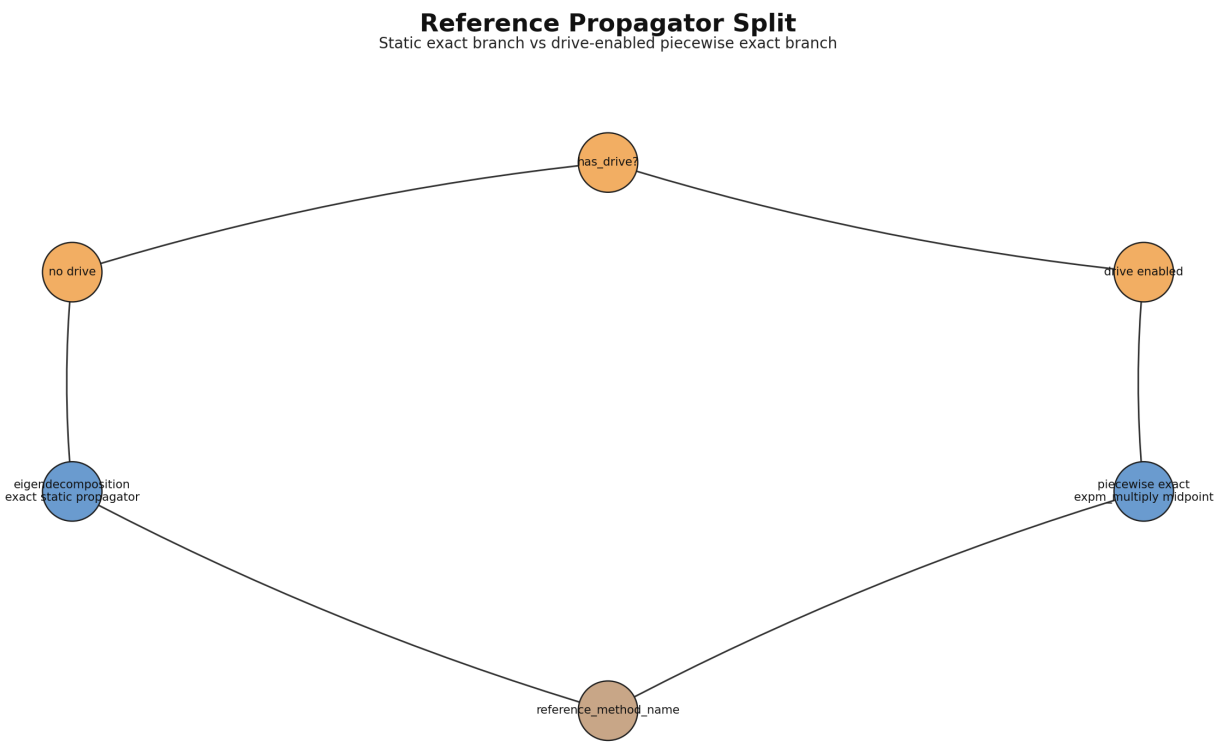
1.7.4 7.4 Safe-test contract

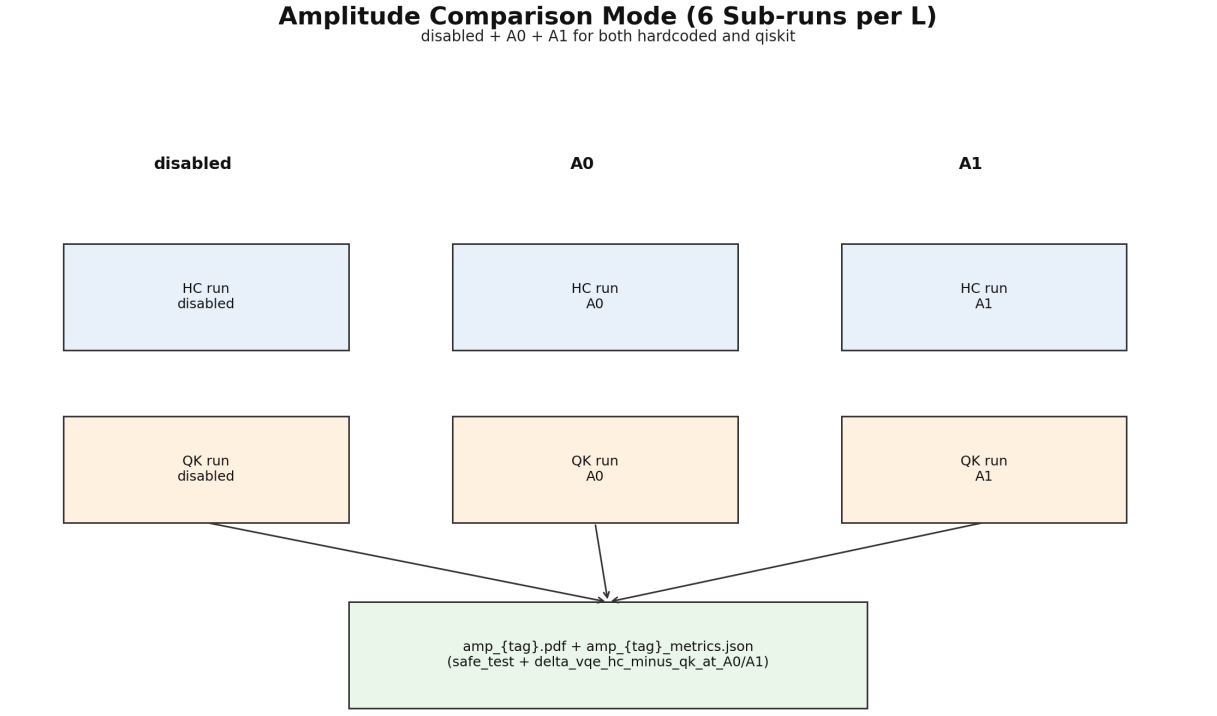
A drive-enabled call with `A=0` should match no-drive channels within numerical threshold. This is a powerful implementation regression check because it validates both path equivalence and parameter plumbing.




Drive Spatial Pattern Weights
Site weights s_j used in v_j(t) = s_j * v(t)
staggered
dimer_bias
custom example


Shared Drive Dynamics Backend
Both implementations converge on common drive-enabled kernels
hardcoded pipeline
_evolve_trotter_suzuki2_absolute
drive_coeff_provider_exyz(t)
trajectory rows
qiskit baseline pipeline
_evolve_piecewise_exact
(expm_multiply midpoint)





1.8 8. Exact Reference Semantics And exact_steps_multiplier

The repository keeps two conceptual families separate:

1. Approximate propagated state (`psi_ansatz_trot`) from Suzuki-Trotter.
2. Reference propagated states (`psi_exact_*`) from exact methods appropriate to mode.

1.8.1 8.1 Branch semantics in plain implementation terms

- `exact_gs` branch: exact/reference propagation from filtered-sector ground-state init.
- `exact_ansatz` branch: exact/reference propagation from ansatz init.
- `trotter` branch: Suzuki-Trotter propagation from ansatz init.

This branch distinction is important for interpretation: not all curves answer the same question.

1.8.2 8.2 Role of exact_steps_multiplier

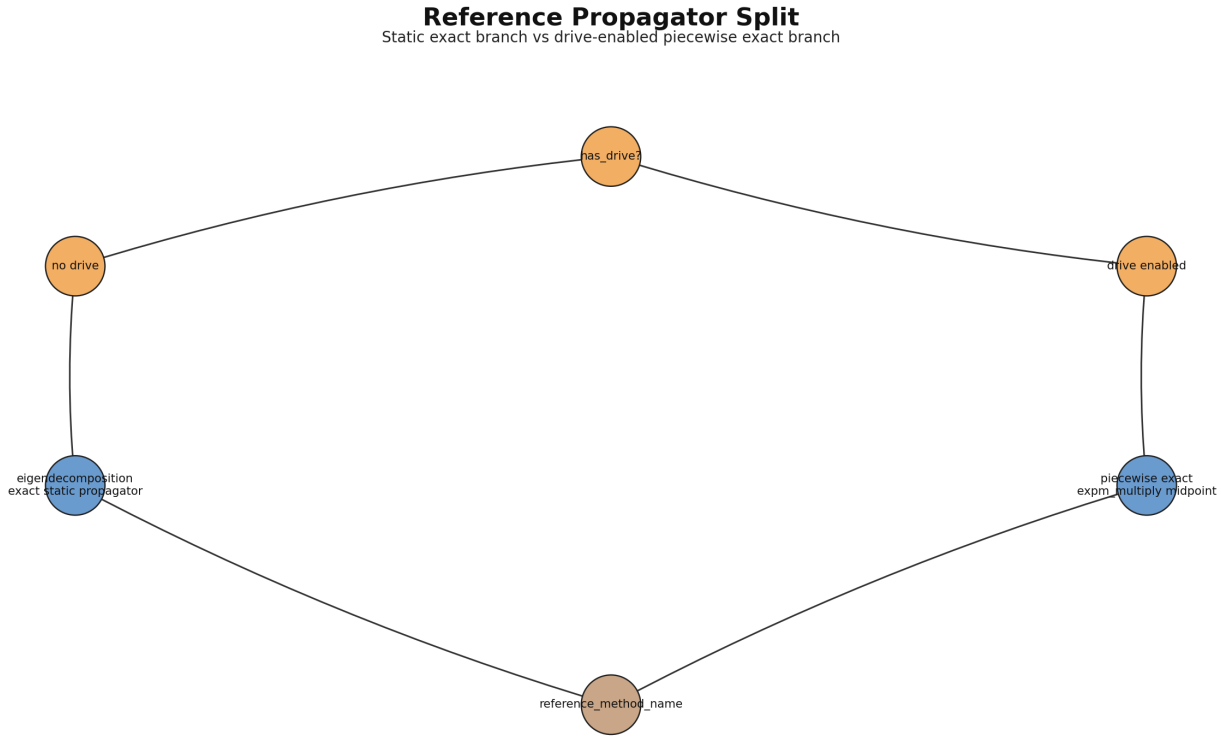
In drive mode only, it decouples reference integration quality from trotter discretization quality. Increasing multiplier tightens reference accuracy without changing trotter branch itself.

1.8.3 8.3 What to check in outputs

- reference method label in settings drive block,
- multiplier value recorded,
- whether total-energy and static-energy channels should be identical (no drive) or not (drive enabled).

1.8.4 8.4 Failure signatures

- static and total channels diverge in no-drive case,
- drive-enabled reference method missing/incorrect,
- fidelity behaving erratically while reference metadata indicates inconsistent branch assumptions.



```
Build quality gates (guide-level)

- operator-core immutability gate
- deterministic diagram regeneration gate
- required section heading gate
- local link/path resolution gate
- figure embedding count gate
- canonical artifact presence gate
- evidence anchor presence gate
- run appendix size cap gate
- pytest snapshot capture gate
- PDF page-count range gate
```

1.9 9. Trajectory Observables: Formulas To Code

This section maps each major trajectory quantity to exact code behavior and interpretation.

1.9.1 9.1 Site-resolved occupancy

For state

$$ |\psi\rangle = \sum_{k=0}^{2^{N_q}-1} \psi_k |k\rangle, \qquad p_k = |\psi_k|^2, $$

site/spin occupancy is accumulated by bit extraction from basis index `k`:

$$ \langle n_{i,\sigma}\rangle = \sum_k p_k\, b_{i,\sigma}(k), \qquad b_{i,\sigma}(k)=((k \gg q_{i,\sigma})\&1). $$

Site-0 channels in JSON are direct projections of site array index 0.

1.9.2 9.2 Doublon

Total doublon channel accumulates

$$ D = \sum_i \langle n_{i,\uparrow} n_{i,\downarrow}\rangle. $$

Average doublon is reported as `D/L`.

1.9.3 9.3 Staggered order

Using total density per site `n_i`, reported staggered channel is

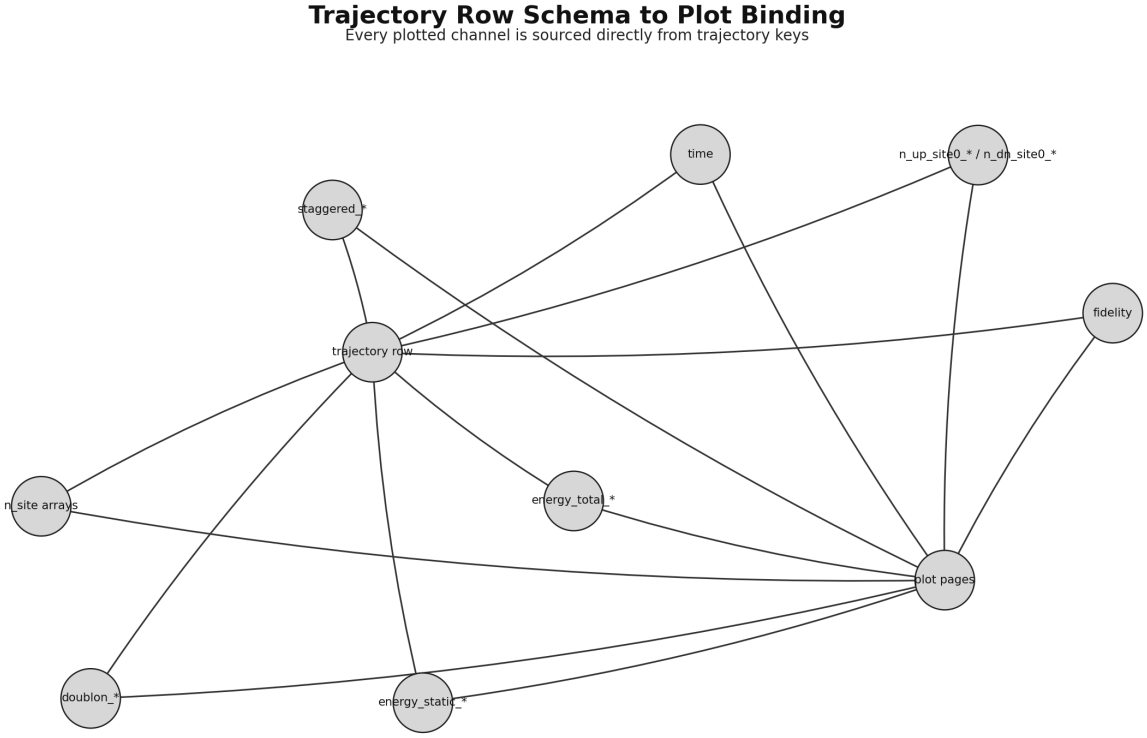
$$ S = \frac{1}{L}\sum_i (-1)^i n_i. $$

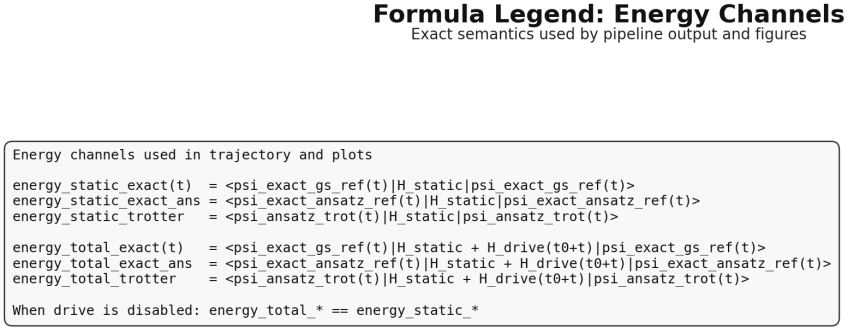
1.9.4 9.4 Energy channels

Static and total channels are separate observables with separate physical meaning in drive mode. In no-drive mode they should match numerically.

1.9.5 9.5 Fidelity

Reported fidelity is projector fidelity against propagated filtered-sector ground manifold, not full-state overlap to a single vector.





Formula Legend: Fidelity

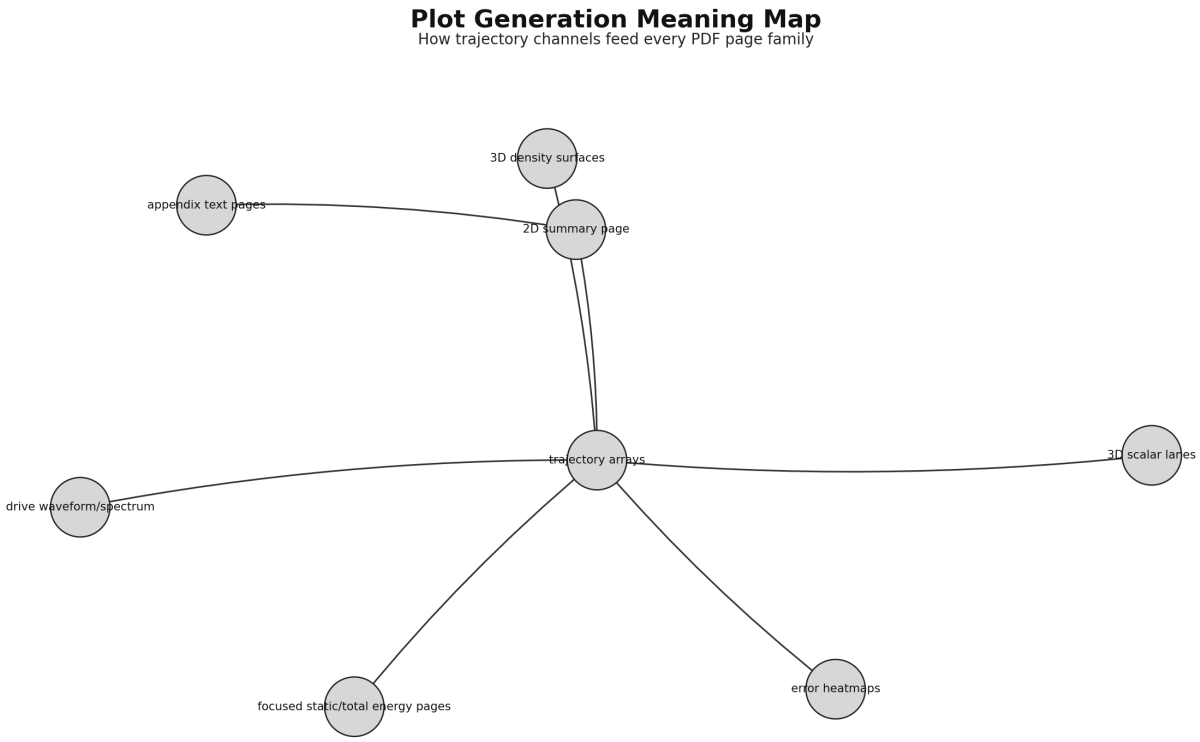
Projector fidelity against filtered-sector propagated ground manifold

```
Fidelity channel

fidelity(t) = <psi_ansatz_trot(t) | P_exact_gs_subspace(t) | psi_ansatz_trot(t)>

where P_exact_gs_subspace(t) projects onto the propagated ground-manifold basis
selected at t=0 from the filtered sector (N_up, N_dn) with energy tolerance:

    E <= E0 + fidelity_subspace_energy_tol

This is not full-state overlap to a single vector; it is projector fidelity
against a (possibly multi-dimensional) reference subspace.
```

Formula Legend: Occupancy, Doublon, Staggered

Observable definitions implemented in trajectory loop

```
Occupation and doublon channels

For basis index idx with probability p(idx)=|psi[idx]|^2:

    n_up[site] += up_bit(site, idx) * p(idx)
    n_dn[site] += dn_bit(site, idx) * p(idx)

Site-0 channels in trajectory:
    n_up_site0_* = n_up[0]
    n_dn_site0_* = n_dn[0]

Doublon:
    doublon_total = sum_site <n_{site,up} n_{site,dn}>
    doublon_avg   = doublon_total / L

Staggered order from total site density n_site:
    staggered = (1/L) * sum_i (-1)^i * n_site[i]
```



1.10 10. Plot Generation Internals And Meaning

This repository writes physically interpretable plots, but only if read with branch/observable semantics in mind.

1.10.1 10.1 Plot data source contract

All plotted channels are read from trajectory rows. There is no hidden post-processor inventing new physics channels. So interpretation starts with row keys.

1.10.2 10.2 Page families in hardcoded pipeline PDF

- 3D total-density surfaces: exact GS, exact ansatz, trotter.
- 3D spin-up and spin-down surfaces.
- 3D lane plots for scalar channels (energy, doublon, staggered).
- Error heatmaps (site-resolved and scalar-channel stacks).
- Drive waveform/spectrum diagnostics (when drive active).
- 2D compact summary page with fidelity, energy overlays, site-0 channels, doublon.
- Focused static-only and total-only energy pages.
- appendix text pages with run metadata and key scalar values.

1.10.3 10.3 Meaningful interpretation rules

- Compare static channels across branches to isolate discretization/initial-state effects.
- Compare total channels in drive mode for actual driven energy response.
- Use site-0 channels as readable local probes, but use site-resolved arrays and heatmaps for full spatial interpretation.
- Use error heatmaps to see where deviations localize in time and space.

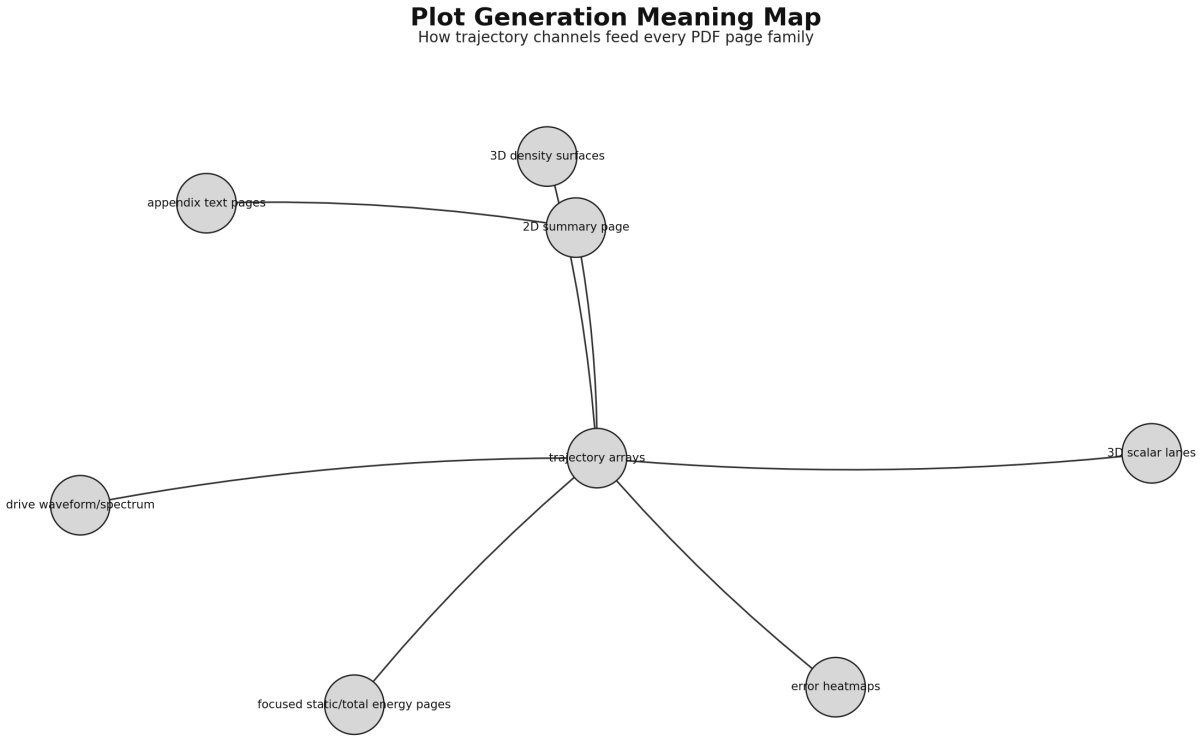
1.10.4 10.4 Common misreads to avoid

- Treating total energy as conserved in driven runs.
- Treating projector fidelity as simple state overlap.

- Interpreting one branch as if it had the same initial state as another branch.
- Ignoring term-order and trotter-step settings when comparing curves.

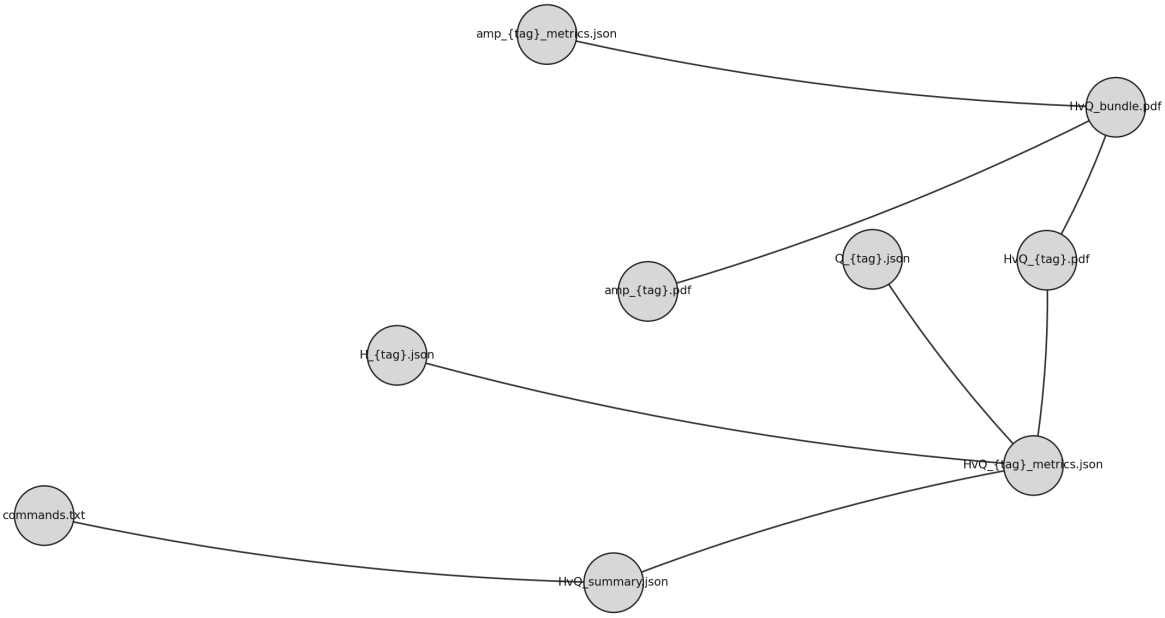
1.10.5 10.5 Why recomputed audit plots were added

The audit figures in this guide are recomputed directly from JSON arrays. They are intended as a second implementation check so that figure interpretation does not rely on internal plotting code alone.



Artifact and JSON Contract Map

How generated outputs compose into comparison/report bundles



Canonical Artifact Metrics Table

Validation and error summary for L2/L3/L4 canonical cases

case	valid	L	drive	|VQE-exact|	max |E_trot-E_exact_ans|
_2_static_t1.0_U4.0_S64_heavy.	True	2	False	3.208e-08	7.799e-02
3_static_t1.0_U4.0_S128_heavy	True	3	False	1.272e-07	8.288e-03
_L4_vt_t1.0_U4.0_S256_dyn.jso	True	4	True	1.035e-04	2.958e-03

1.11 11. Canonical Artifact Audits (L=2,3,4)

This section records direct metrics from the canonical heavy trio and ties them to expected behavior.

1.11.1 11.1 L=2 static heavy

- drive enabled: `False`
- optimizer: `COBYLA`

- parameter count: 6
- |VQE - exact_filtered|: 3.20842046264147e-08
- max |E_static_trot - E_static_exact_ans|: 0.07798533165196853
- final projector fidelity: 0.9869249072500748

Interpretation:

- Static total-minus-static delta is exactly 0 in this case, which is expected in no-drive mode.
- Trotter vs exact-ansatz gap is visible at finite step count and should shrink with refinement.

1.11.2 11.2 L=3 static heavy

- drive enabled: False
- optimizer: COBYLA
- parameter count: 16
- |VQE - exact_filtered|: 1.2720054520798385e-07
- max |E_static_trot - E_static_exact_ans|: 0.008288216471344256
- final projector fidelity: 0.9985135925614043

Interpretation:

- Fidelity is high and stable near the final time.
- Error scale is lower than L=2 in this selected heavy static case due to chosen discretization and branch settings.

1.11.3 11.3 L=4 drive-enabled dyn

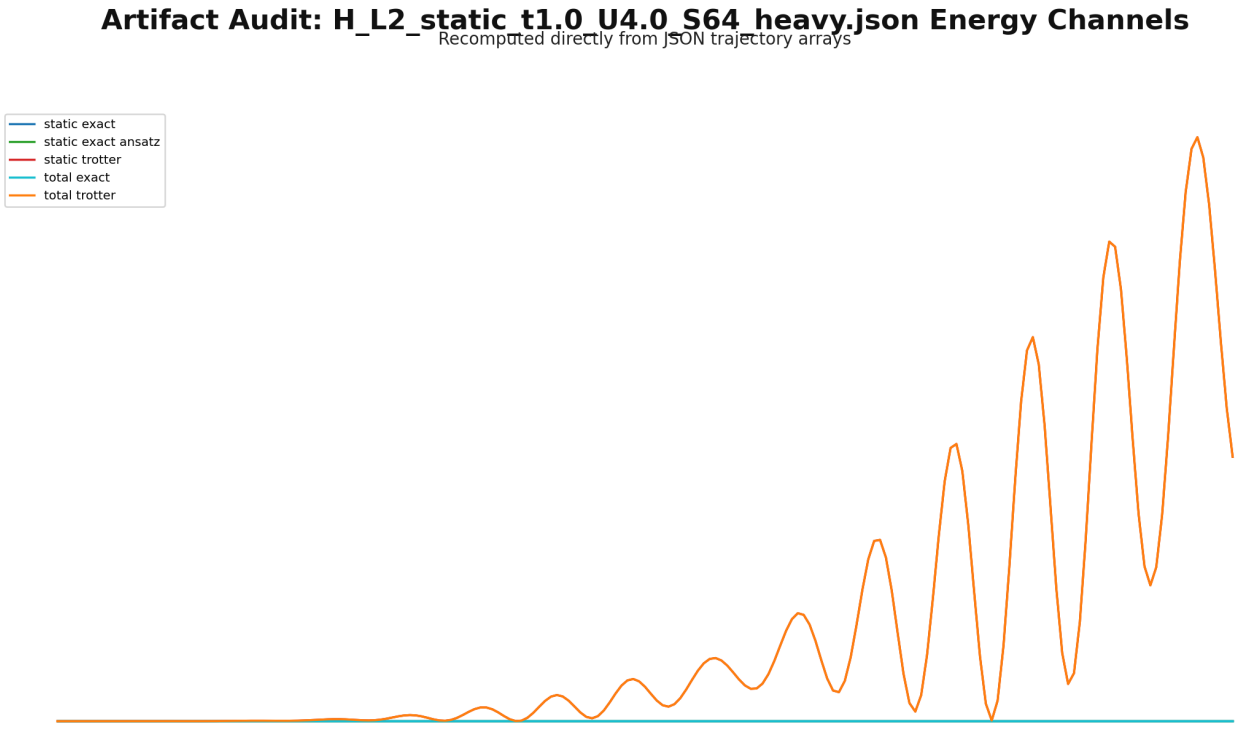
- drive enabled: True
- optimizer: COBYLA
- parameter count: 104
- reference method: exponential_midpoint_magnus2_order2
- |VQE - exact_filtered|: 0.00010348314457075958
- max |E_static_trot - E_static_exact_ans|: 0.0029578796116696005
- max |E_total_trot - E_static_trot|: 0.12363082821378724
- final projector fidelity: 0.9993357304628818

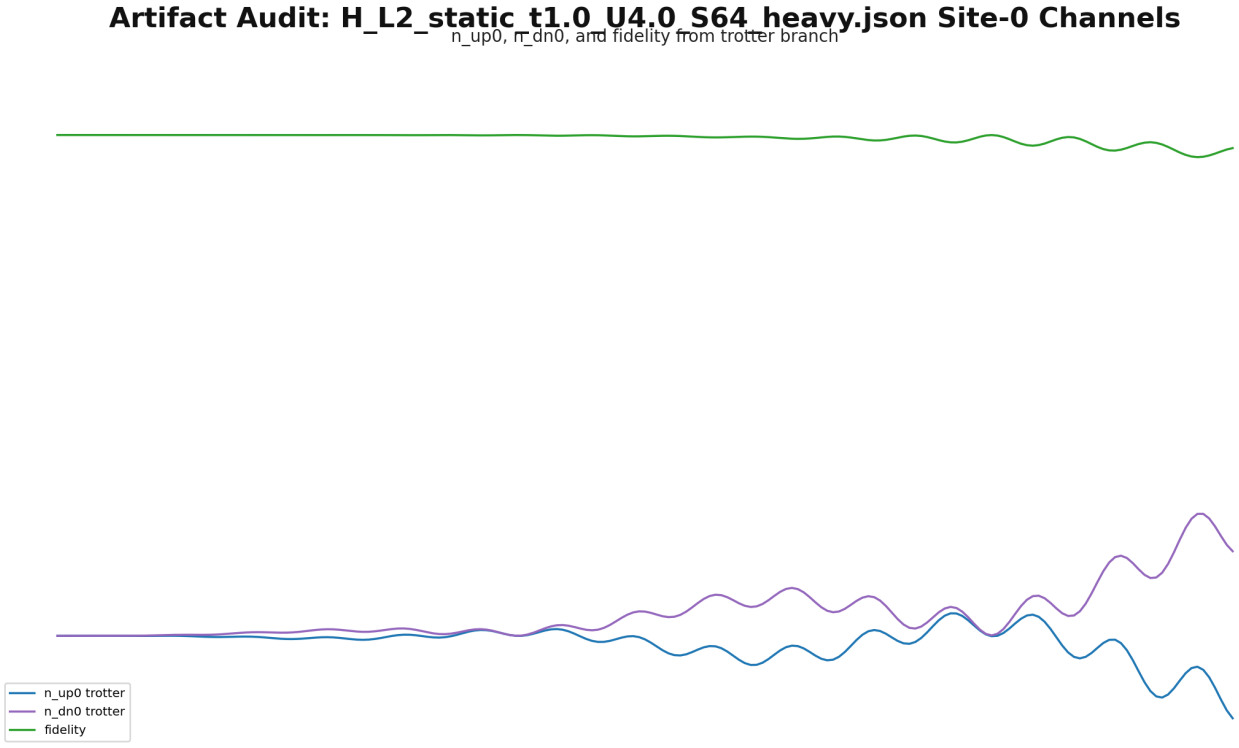
Interpretation:

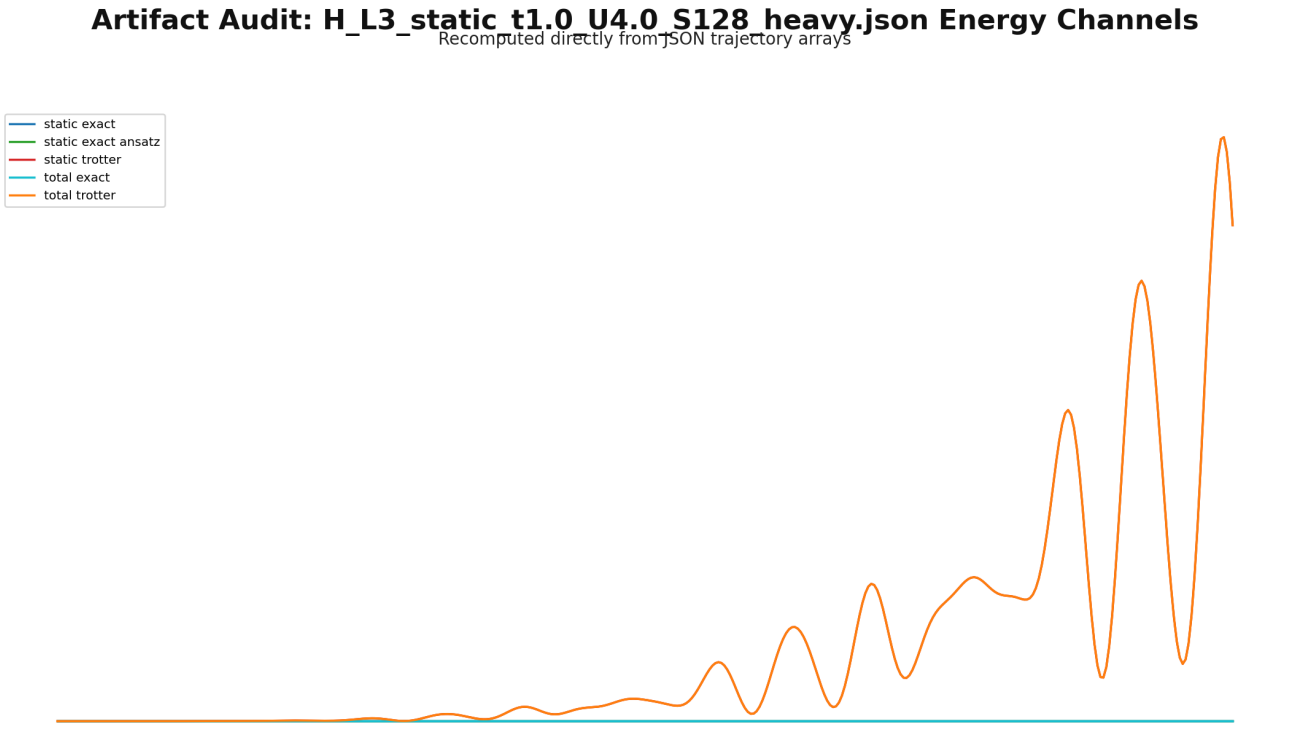
- Nonzero total-minus-static delta is expected and confirms driven observable separation.
- The reported reference method indicates the drive-aware branch was used.

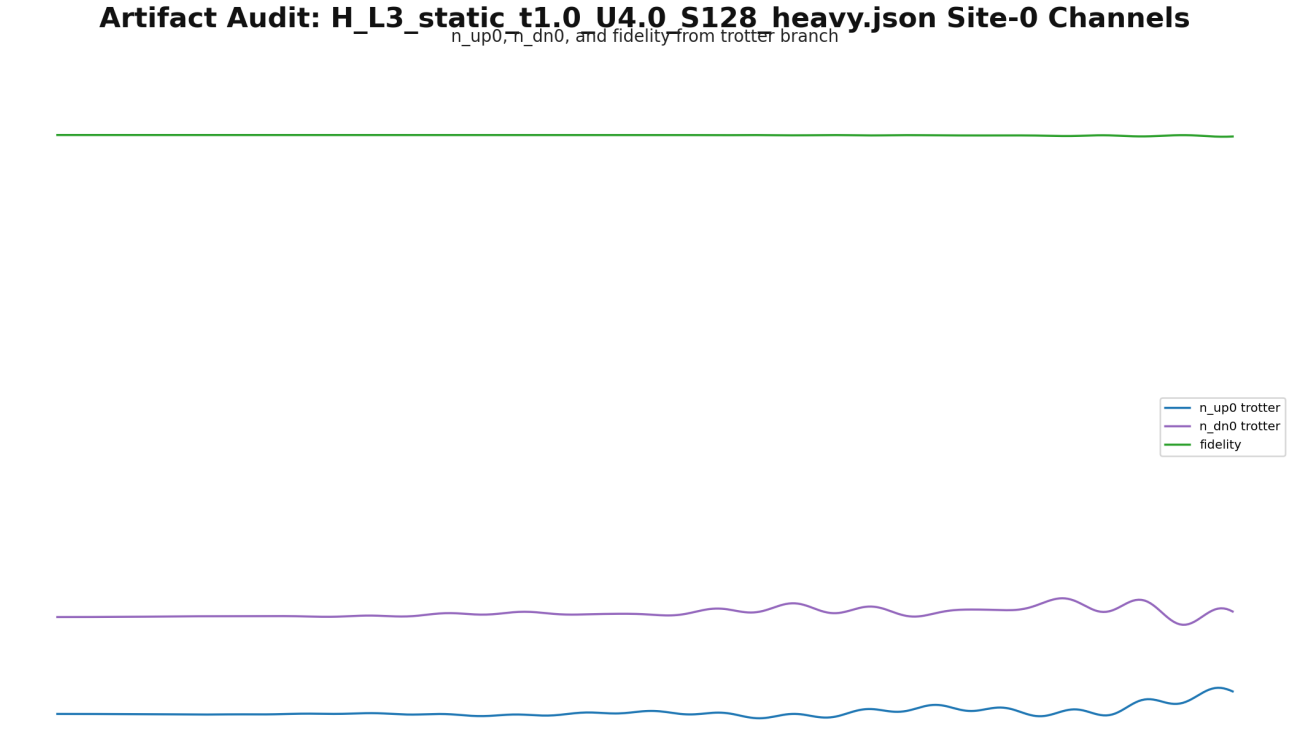
1.11.4 11.4 Cross-case summary

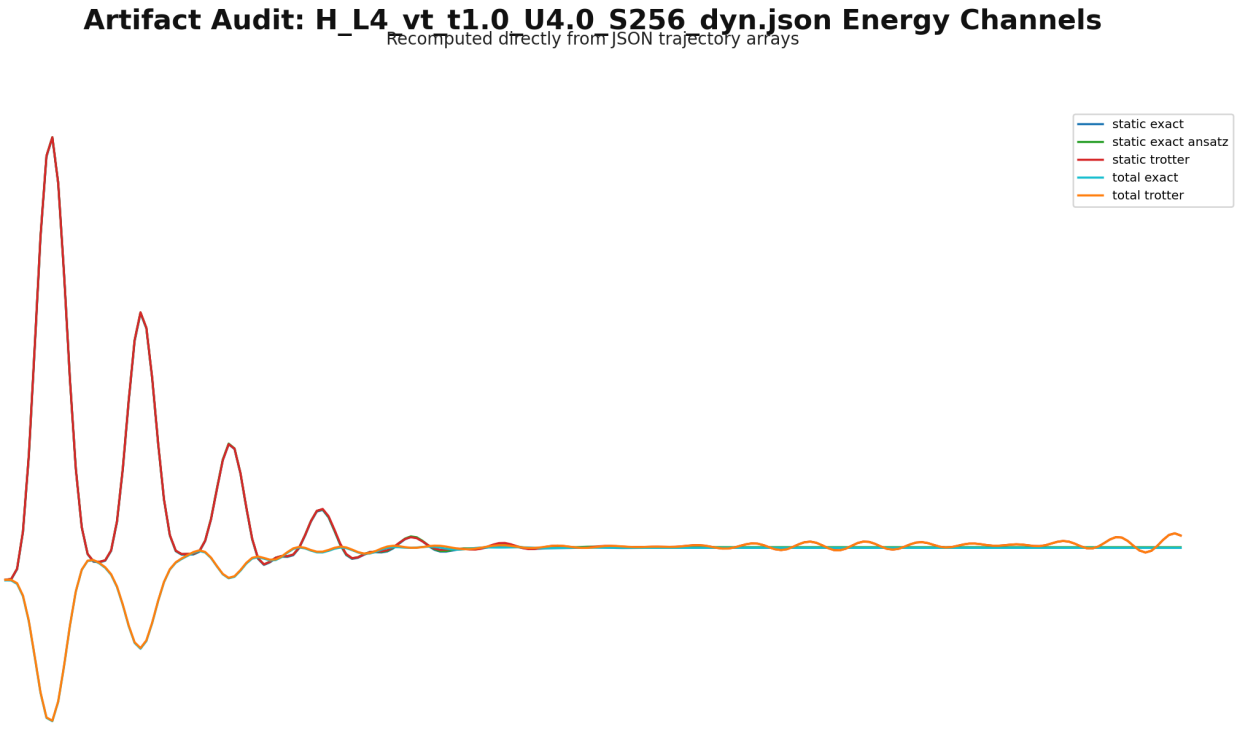
- all canonical cases valid: True
- occupancy bounds checks: pass in canonical metrics payload
- doublon bounds checks: pass in canonical metrics payload

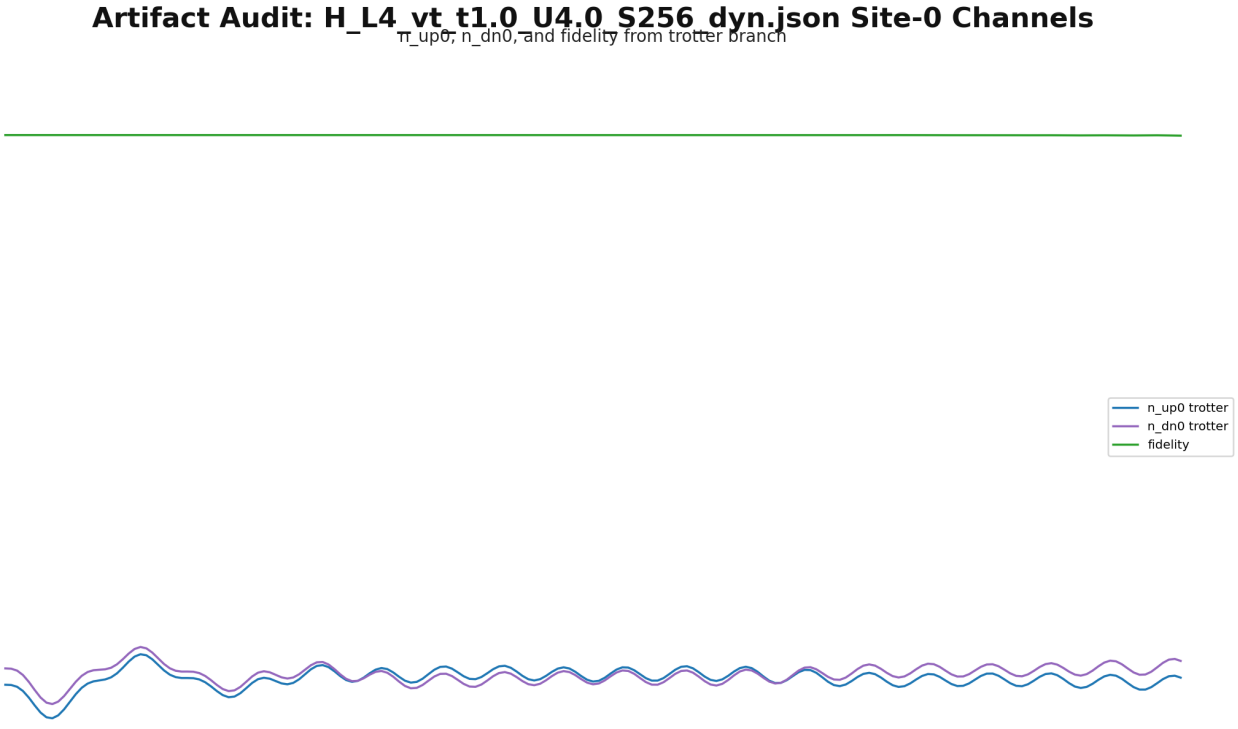

Artifact Audit: H_L2_static_t1.0_U4.0_S64_heavy.json Energy Channels
Recomputed directly from JSON trajectory arrays
static exact
static exact ansatz
static trotter
total exact
total trotter

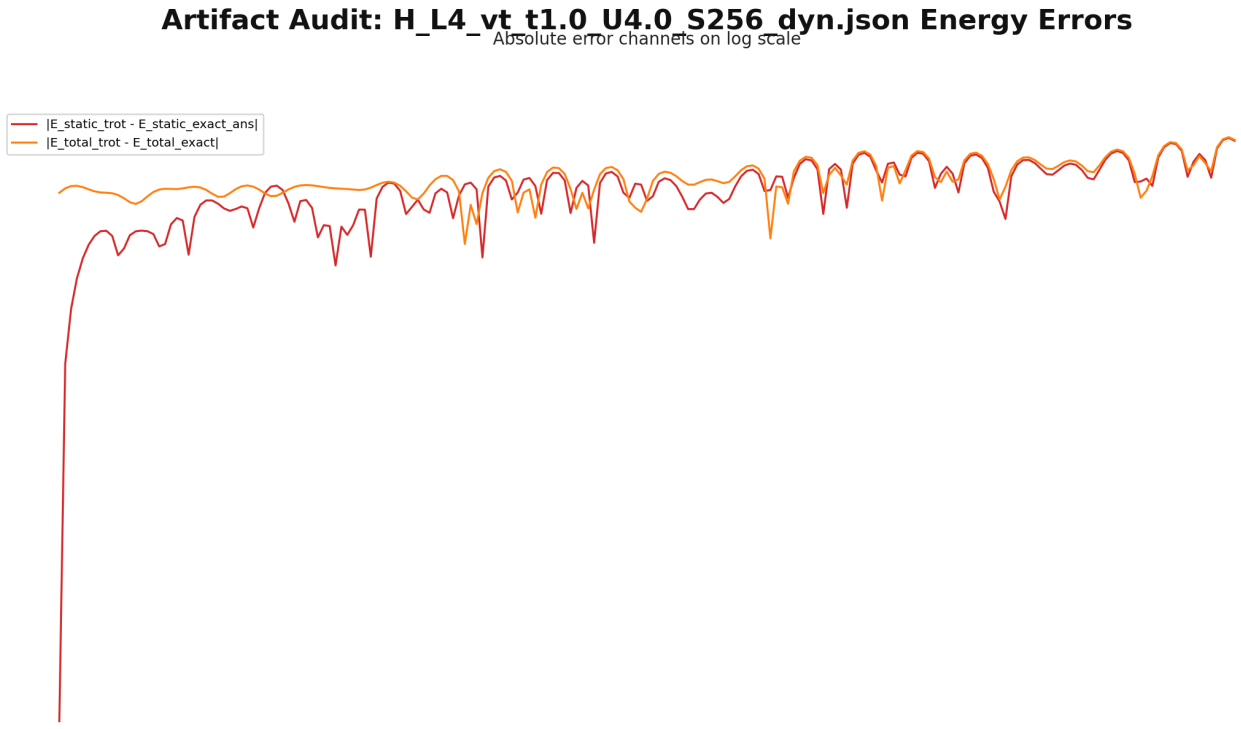

Artifact Audit: H_L2_static_t1.0_U4.0_S64_heavy.json Site-0 Channels
n_up0, n_dn0, and fidelity from trotter branch
n_up0 trotter
n_dn0 trotter
fidelity

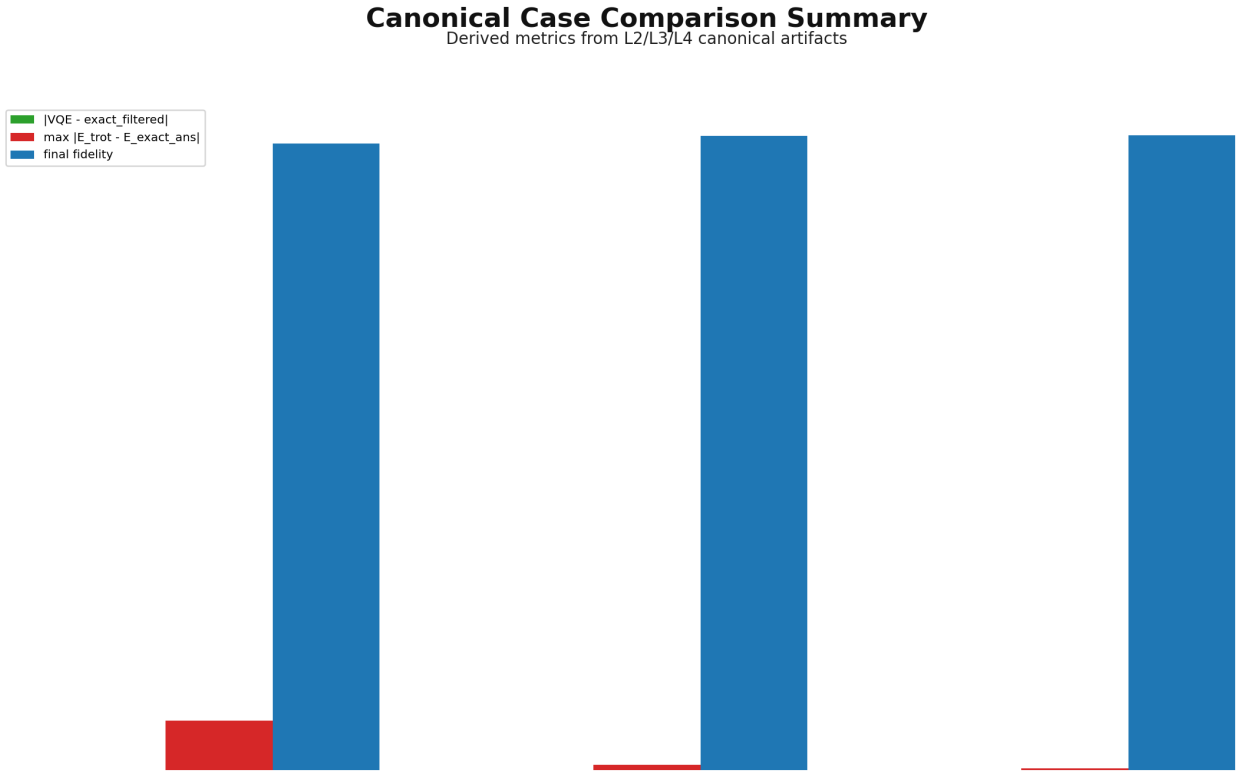

Artifact Audit: H_L3_static_t1.0_U4.0_S128_heavy.json Energy Channels
Recomputed directly from JSON trajectory arrays
static exact
static exact ansatz
static trotter
total exact
total trotter


Artifact Audit: H_L3_static_t1.0_U4.0_S128_heavy.json Site-0 Channels
n_up0, n_dn0, and fidelity from trotter branch
n_up0 trotter
n_dn0 trotter
fidelity


Artifact Audit: H_L4_vt_t1.0_U4.0_S256_dyn.json Energy Channels
Recomputed directly from JSON trajectory arrays
static exact
static exact ansatz
static trotter
total exact
total trotter


Artifact Audit: H_L4_vt_t1.0_U4.0_S256_dyn.json Site-0 Channels
n_up0, n_dn0, and fidelity from trotter branch
n_up0 trotter
n_dn0 trotter
fidelity





1.12 12. Minimal Compare/Qiskit Validation Role

This guide focuses on implementation meaning in the hardcoded path, but compare/qiskit still matter as validation infrastructure.

1.12.1 12.1 What compare runner should do

- build two commands,
- run both (or load existing JSONs),
- compute reconciled metrics,
- emit combined reports.

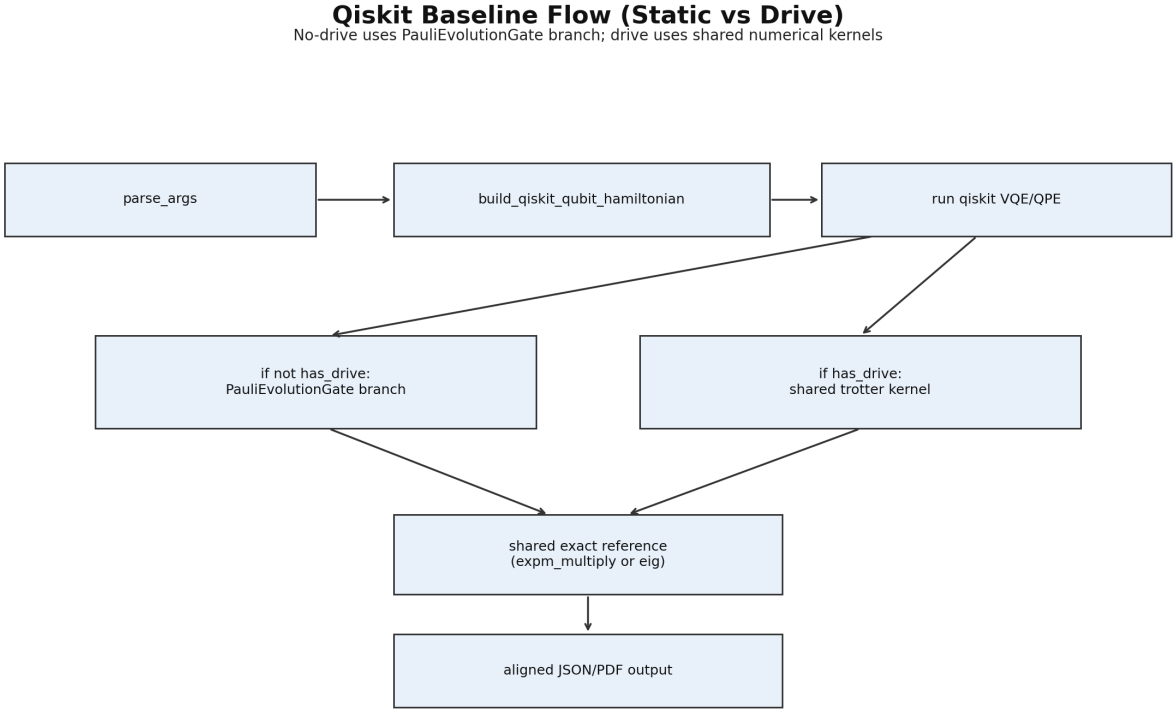
It should not alter the physical model settings while routing arguments.

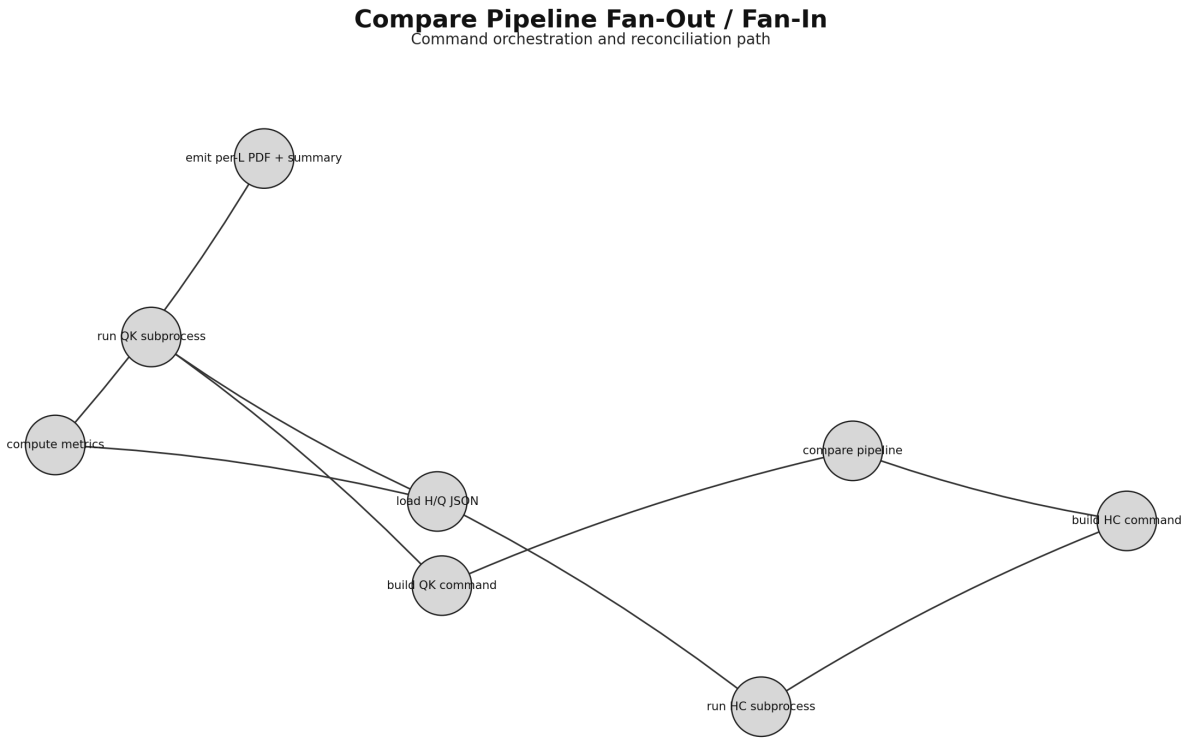
1.12.2 12.2 What qiskit baseline contributes

- independent implementation pathway for cross-checking,
- alternative library stack for VQE internals,
- aligned output schema to enable direct comparison.

1.12.3 12.3 What this section intentionally does not do

It does not attempt to prove one backend is universally superior. It treats qiskit path as a useful comparator and schema-aligned reference baseline.





1.13 13. Extension Safety: Safe And High-Risk Edits

Safe edits tend to preserve one of these properties:

- no change to operator algebra semantics,
- no change to label/bit convention,
- no change to drive pass-through behavior,
- no change to trajectory key contracts.

High-risk edits include:

- touching indexing logic without exhaustive checks,
- touching JW/number-operator semantics,
- adding drive knobs without updating all pipeline surfaces,
- changing meaning of existing JSON keys.

1.13.1 13.1 Practical pre-merge checklist

1. Re-run canonical L2/L3/L4 artifact generation.
2. Recompute guide asset metrics and inspect deltas.
3. Verify safe-test logic still passes under A=0 check scenarios.
4. Confirm section-10 plot meaning rules still match code.
5. Confirm no operator-core base file edits slipped in.

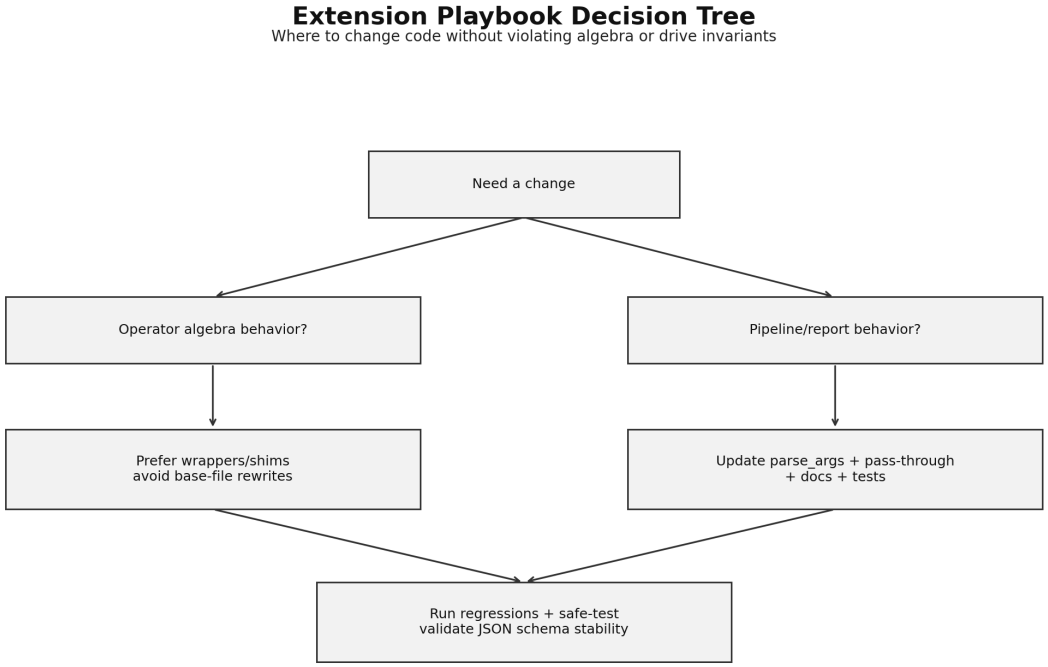
1.13.2 13.2 Key targeted function anchors

- `_evolve_trotter_suzuki2_absolute` in `Fermi-Hamil-JW-VQE-TROTTER-PIPELINE/pipelines/hardcoded_hubbard_pip` lines 321-391: def _evolve_trotter_suzuki2_absolute(psi0: np.ndarray, ordered_labels_exyz: list[str], coeff_map_exyz: dict[str, complex], compiled_actions: dict[str, CompiledPauliAction], time_value: float, trotter_steps: int, *, drive_coeff_provider_exyz: Any | None = None, t0: float = 0.0, time_sampling: str = "midpoint", coeff_tol: float = 1e-12,) -> np.ndarray: """Suzuki-Trotter order-2 evolution, with optional time-dependent drive. When *drive_coeff_provider_exyz* is `None` the original bit-for-bit time-independent path is taken (no behavioural change). When provided, drive coefficients are sampled once per Trotter slice and

additively merged with the static coefficients. """ # --- time-independent fast path (bit-for-bit identical to original) --- if drive_coeff_provider_exyz is None: psi = np.array(psi0, copy=True) if abs(time_value) <= 1e-15: return psi dt = float(time_value) / float(trotter_steps) half = 0.5 * dt for _ in range(trotter_steps):

- `_spin_orbital_bit_index` in `Fermi-Hamil-JW-VQE-TROTTER-PIPELINE/pipelines/hardcoded_hubbard_pipeline.py` lines 430-436: def _spin_orbital_bit_index(...)
- `_site_resolved_number_observables` in `Fermi-Hamil-JW-VQE-TROTTER-PIPELINE/pipelines/hardcoded_hubbard_pi` lines 439-461: def _site_resolved_number_observables(psi: np.ndarray, num_sites: int, ordering: str,) -> tuple[np.ndarray, np.ndarray, float]: probs = np.abs(psi) ** 2 n_up = np.zeros(int(num_sites), dtype=float) n_dn = np.zeros(int(num_sites), dtype=float) doublon_total = 0.0 up_bits = [_spin_orbital_bit_index(site, 0, num_sites, ordering) for site in range(int(num_sites))] dn_bits = [_spin_orbital_bit_index(site, 1, num_sites, ordering) for site in range(int(num_sites))] for idx, prob in enumerate(probs):
- `_run_hardcoded_vqe` in `Fermi-Hamil-JW-VQE-TROTTER-PIPELINE/pipelines/hardcoded_hubbard_pipeline.py` lines 500-577: def _run_hardcoded_vqe(...)
- `_evolve_piecewise_exact` in `Fermi-Hamil-JW-VQE-TROTTER-PIPELINE/pipelines/hardcoded_hubbard_pipeline.py` lines 875-1033: def *evolve_piecewise_exact(*, psi0: np.ndarray, hmat_static: np.ndarray, drive_coeff_provider_exyz: Any, time_value: float, trotter_steps: int, t0: float = 0.0, time_sampling: str = "midpoint",) -> np.ndarray: """Piecewise-constant matrix-exponential reference propagator. Approximation order ---------- This function is* ***not*** *a true time-ordered exponential. It is a piecewise-constant approximation: each sub-interval [t_k, t{k+1}] of width Deltat = time_value / trotter_steps is replaced by the exact exponential of H evaluated at a single representative time t_k. The order depends on how t_k is chosen (*`time_sampling`*):
- `_simulate_trajectory` in `Fermi-Hamil-JW-VQE-TROTTER-PIPELINE/pipelines/hardcoded_hubbard_pipeline.py` lines 1036-1289: def _simulate_trajectory(*, num_sites: int, ordering: str, psi0_ansatz_trot: np.ndarray, psi0_exact_ref: np.ndarray, fidelity_subspace_basis_v0: np.ndarray, fidelity_subspace_energy_tol: float, hmat: np.ndarray, ordered_labels_exyz: list[str], coeff_map_exyz: dict[str, complex], trotter_steps: int, t_final: float, num_times: int, suzuki_order: int, drive_coeff_provider_exyz: Any | None = None, drive_t0: float = 0.0, drive_time_sampling: str = "midpoint", exact_steps_multiplier: int = 1,) -> tuple[list[dict[str, float]], list[np.ndarray]]: if int(suzuki_order) != 2: raise ValueError("This script currently supports suzuki_order=2 only.") nq = 2 * int(num_sites) evals, evecs = np.linalg.eigh(hmat) evecs_dag = np.conjugate(evecs).T compiled = {lbl: _compile_pauli_action(lbl, nq) for lbl in ordered_labels_exyz} times = np.linspace(0.0, float(t_final), int(num_times)) n_times = int(times.size) stride = max(1, n_times // 20) t0 = time.perf_counter() basis_v0 = np.asarray(fidelity_subspace_basis_v0, dtype=complex) if basis_v0.ndim != 2 or basis_v0.shape[0] != psi0_ansatz_trot.size: raise ValueError("fidelity_subspace_basis_v0 must have shape (dim, k) with matching dim.") if basis_v0.shape[1] <= 0: raise ValueError("fidelity_subspace_basis_v0 must contain at least one basis vector.") has_drive = drive_coeff_provider_exyz is not None # When drive is enabled the reference propagator may use a finer step count # to improve its quality independently of the Trotter discretization. reference_steps = int(trotter_steps) * max(1, int(exact_steps_multiplier)) static_basis_eig = evecs_dag @ basis_v0 _ai_log("hardcoded_trajectory_start", L=int(num_sites), num_times=n_times, t_final=float(t_final), trotter_steps=int(trotter_steps), reference_steps=reference_steps, exact_steps_multiplier=int(exact_steps_multiplier), suzuki_order=int(suzuki_order), drive_enabled=has_drive, ground_subspace_dimension=int(basis_v0.shape[1]), fidelity_subspace_energy_tol=float(fidelity_subspace_energy_tol), fidelity_selection_rule="E <= E0 + tol",) rows: list[dict[str, float]] = [] exact_states: list[np.ndarray] = [] for idx, time_val in enumerate(times):
- `evaluate_drive_waveform` in `src/quantum/drives_time_potential.py` lines 137-170: def evaluate_drive_waveform(...)
- `hartree_fock_bitstring` in `src/quantum/hartree_fock_reference_state.py` lines 101-112: def hartree_fock_bitstring(...)
- `hartree_fock_statevector` in `src/quantum/hartree_fock_reference_state.py` lines 115-135: def hartree_fock_statevector(...)
- `vqe_minimize` in `src/quantum/vqe_latex_python_pairs.py` lines 689-794: def vqe_minimize(H: PauliPolynomial, ansatz: Any, psi_ref: np.ndarray, *, restarts: int = 3, seed: int = 7, initial_point_stddev: float = 0.3, method: str = "SLSQP", maxiter: int = 1800, bounds: Optional[Tuple[float, float]] = (-math.pi, math.pi),) -> VQEResult: """ Hardcoded VQE: minimize <psi(theta)|H|psi(theta)> with a statevector backend. Uses SciPy if available; otherwise falls back to a tiny coordinate search. """ minimize = _try_import_scipy_minimize() rng = np.random.default_rng(int(seed)) npar = int(ansatz.num_parameters) if npar <= 0: log.error("ansatz has no parameters") def energy_fn(x: np.ndarray) -> float: theta = np.asarray(x, dtype=float) psi = ansatz.prepare_state(theta, psi_ref) return expval_pauli_polynomial(psi, H) best_energy = float("inf")

best_theta = None best_restart = -1 best_nfev = 0 best_nit = 0 best_success = False best_message = "no run" for r in range(int(restarts)):




```
Build quality gates (guide-level)

- operator-core immutability gate
- deterministic diagram regeneration gate
- required section heading gate
- local link/path resolution gate
- figure embedding count gate
- canonical artifact presence gate
- evidence anchor presence gate
- run appendix size cap gate
- pytest snapshot capture gate
- PDF page-count range gate
```

1.14 14. Ultra-Brief Run Appendix

This appendix is intentionally short.

Use the build wrapper from subrepo root:

`scripts/build_repo_implementation_guide.sh`

It regenerates assets, enforces deterministic figures, runs `pytest -q` snapshot, and builds the PDF.

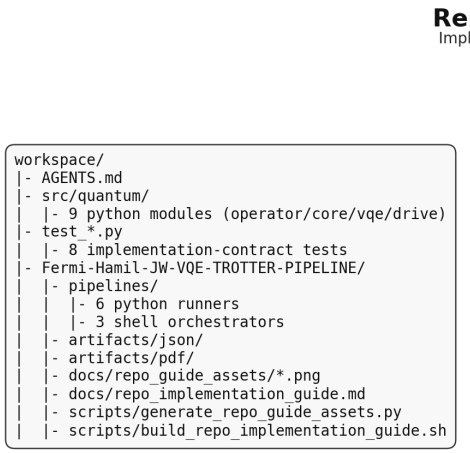
Primary outputs:

- `docs/Repo implementation guide.PDF`
- `docs/repo_guide_assets/repo_guide_summary.json`
- `docs/repo_guide_assets/repo_guide_artifact_metrics.json`

For implementation review, read sections 4-11 first.

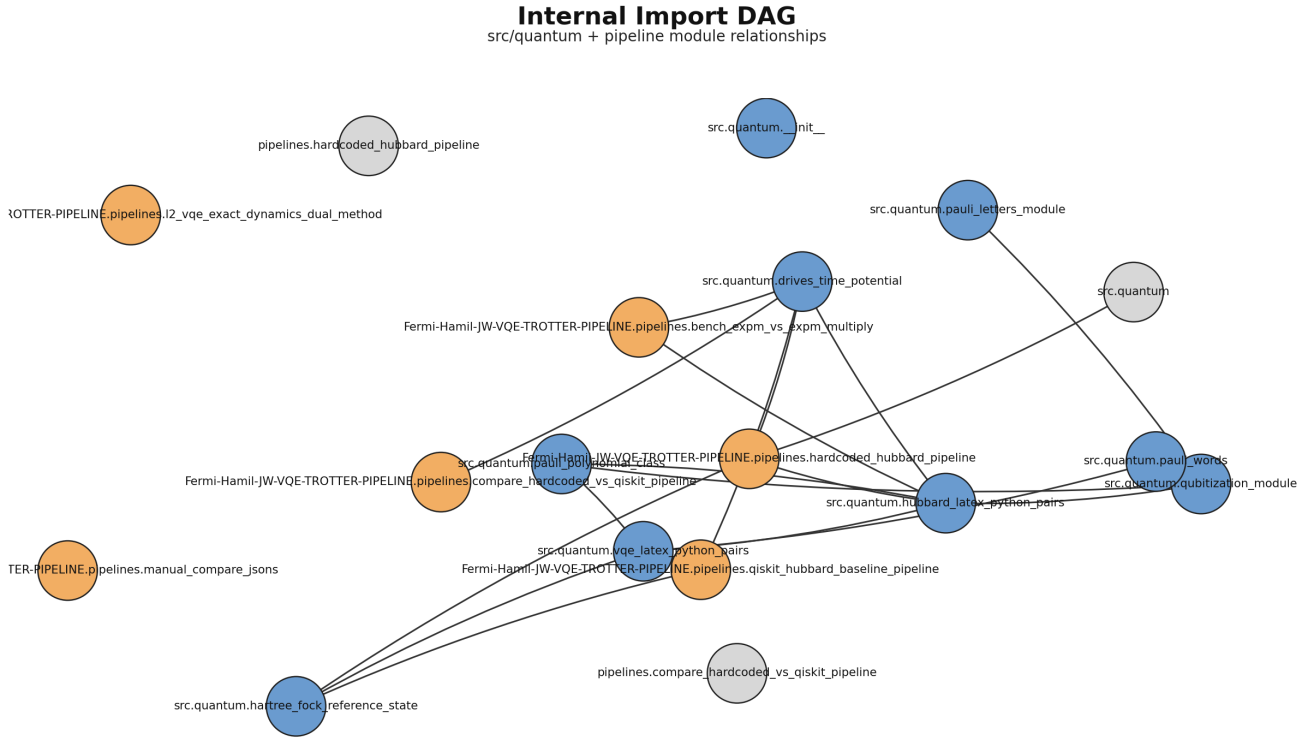
1.15 Appendix A: Figure Atlas (Condensed)

1.15.1 A.1 Repository Structure Map



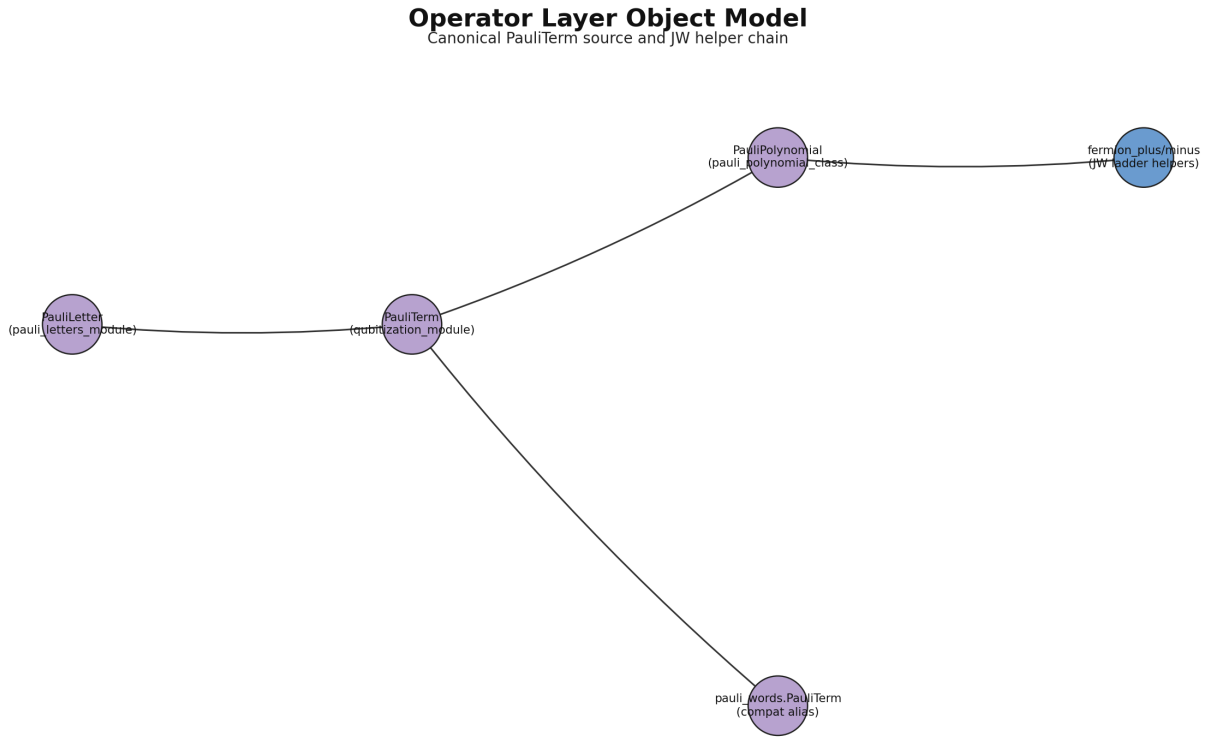
Use: Workspace and artifact topology. Cross-check against section formulas before drawing conclusions.

1.15.2 A.2 Internal Import DAG



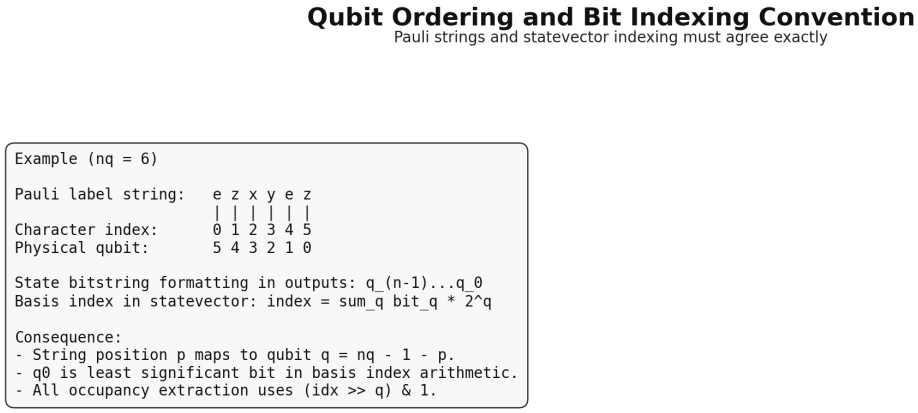
Use: src/quantum and pipeline imports. Cross-check against section formulas before drawing conclusions.

1.15.3 A.3 Operator Layer Object Model



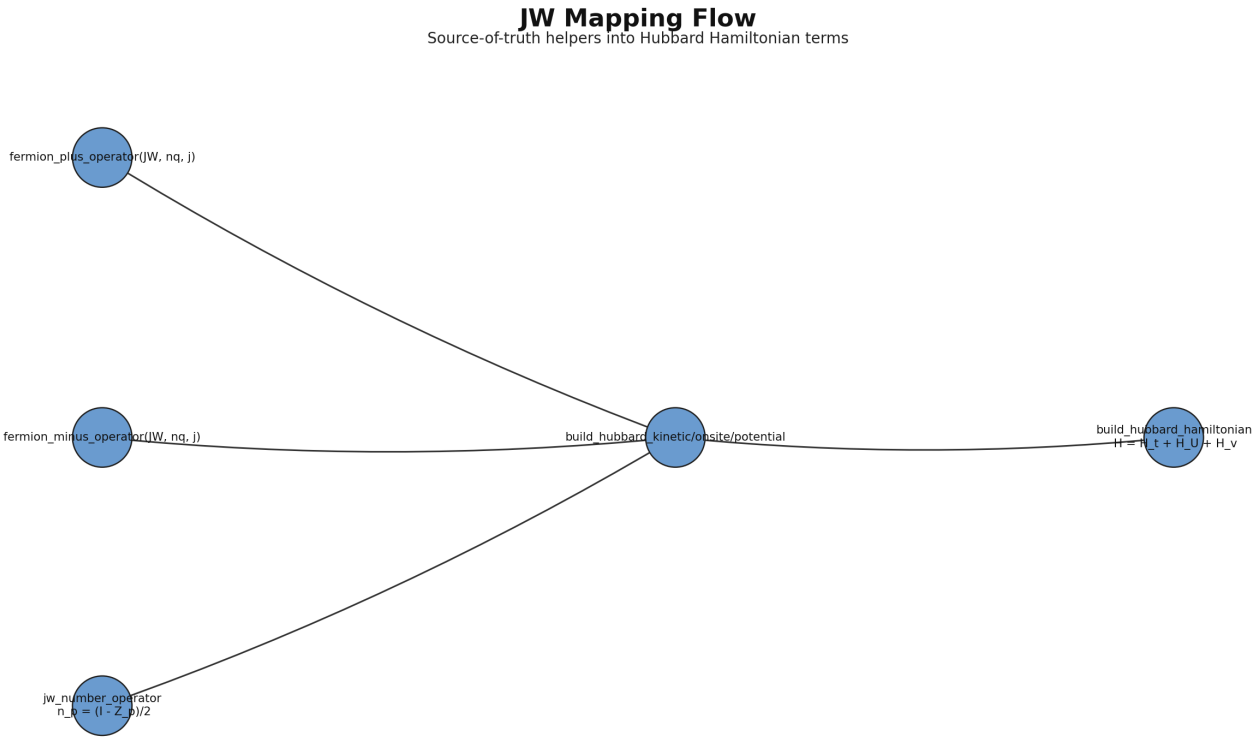
Use: Pauli abstraction chain. Cross-check against section formulas before drawing conclusions.

1.15.4 A.4 Qubit Ordering Convention



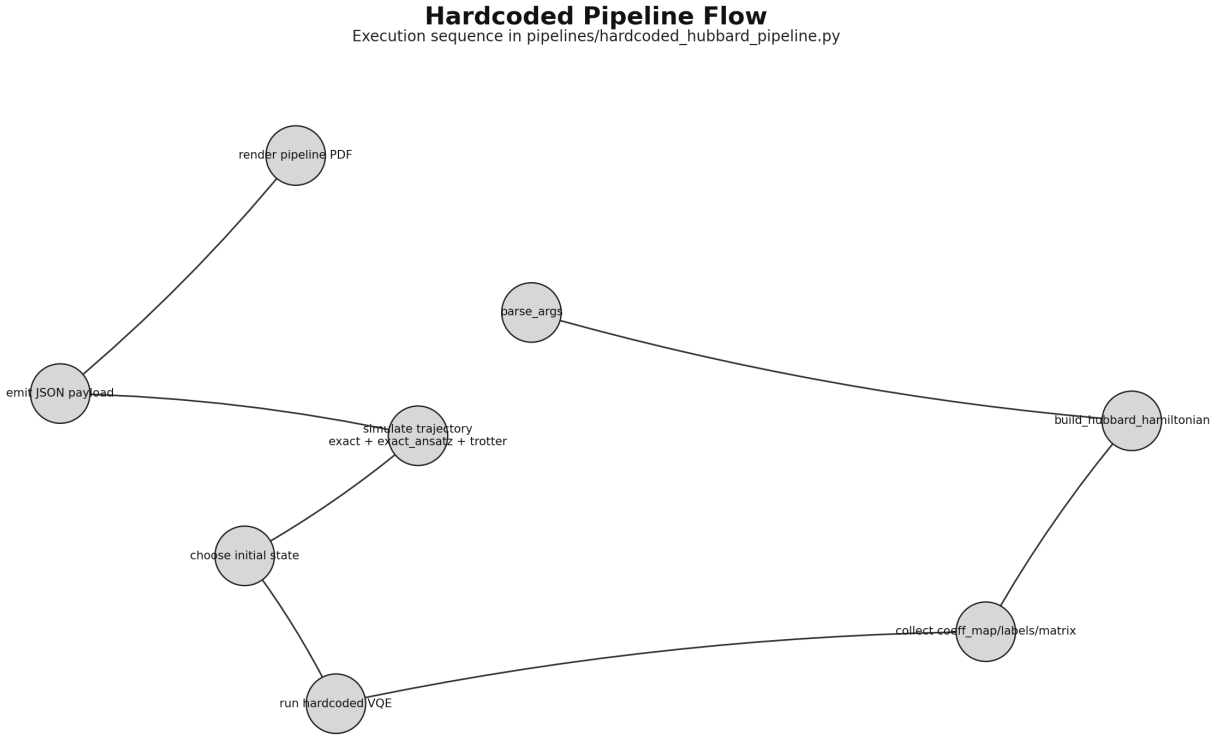
Use: String to qubit index mapping. Cross-check against section formulas before drawing conclusions.

1.15.5 A.5 JW Mapping Flow



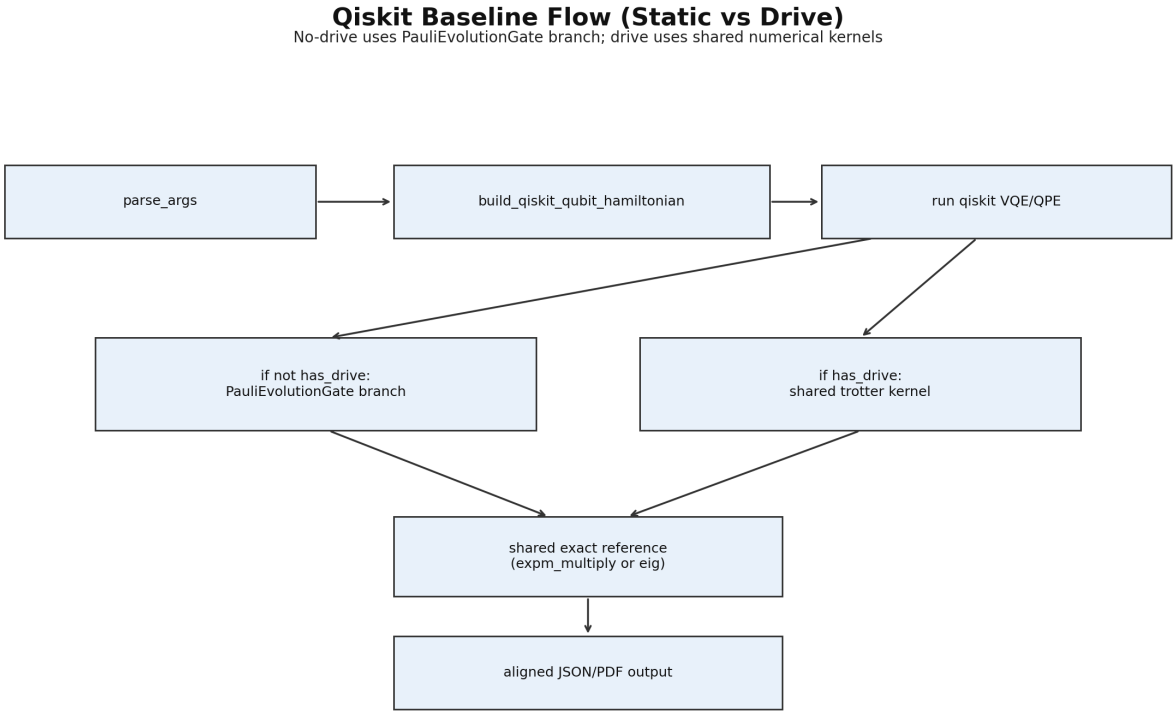
Use: Ladder helper flow into Hamiltonian. Cross-check against section formulas before drawing conclusions.

1.15.6 A.6 Hardcoded Pipeline Flow



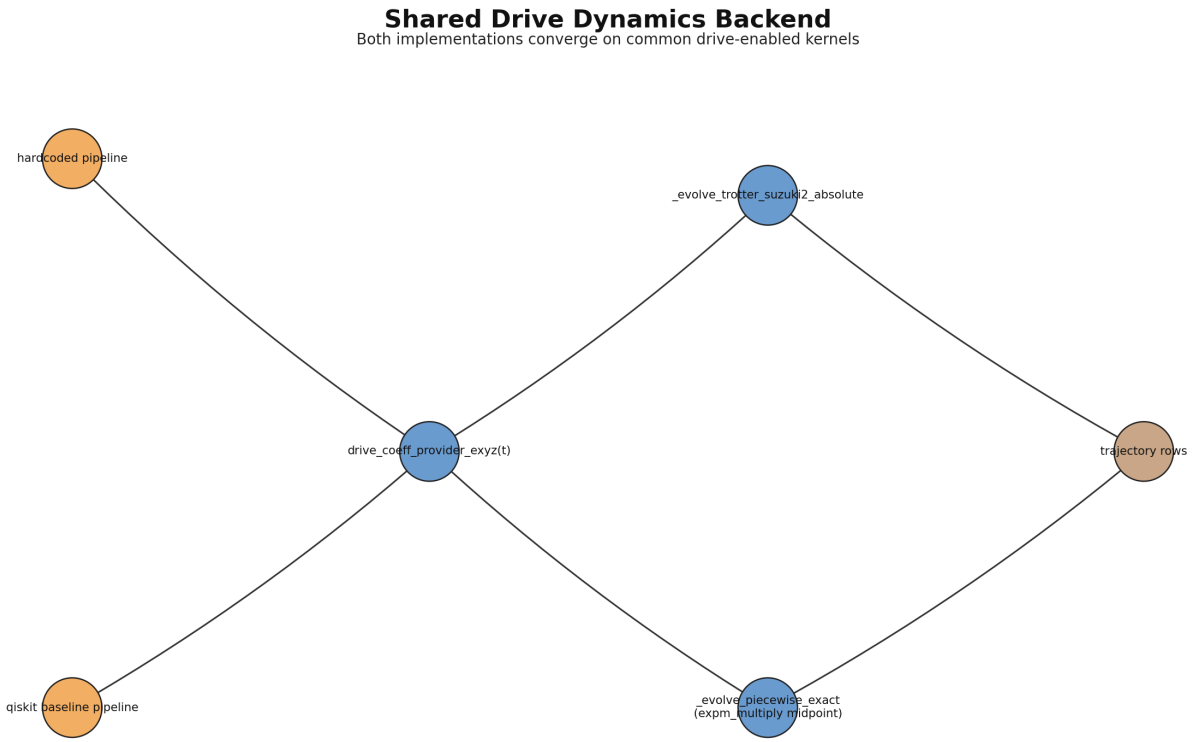
Use: Execution sequence. Cross-check against section formulas before drawing conclusions.

1.15.7 A.7 Qiskit Baseline Flow



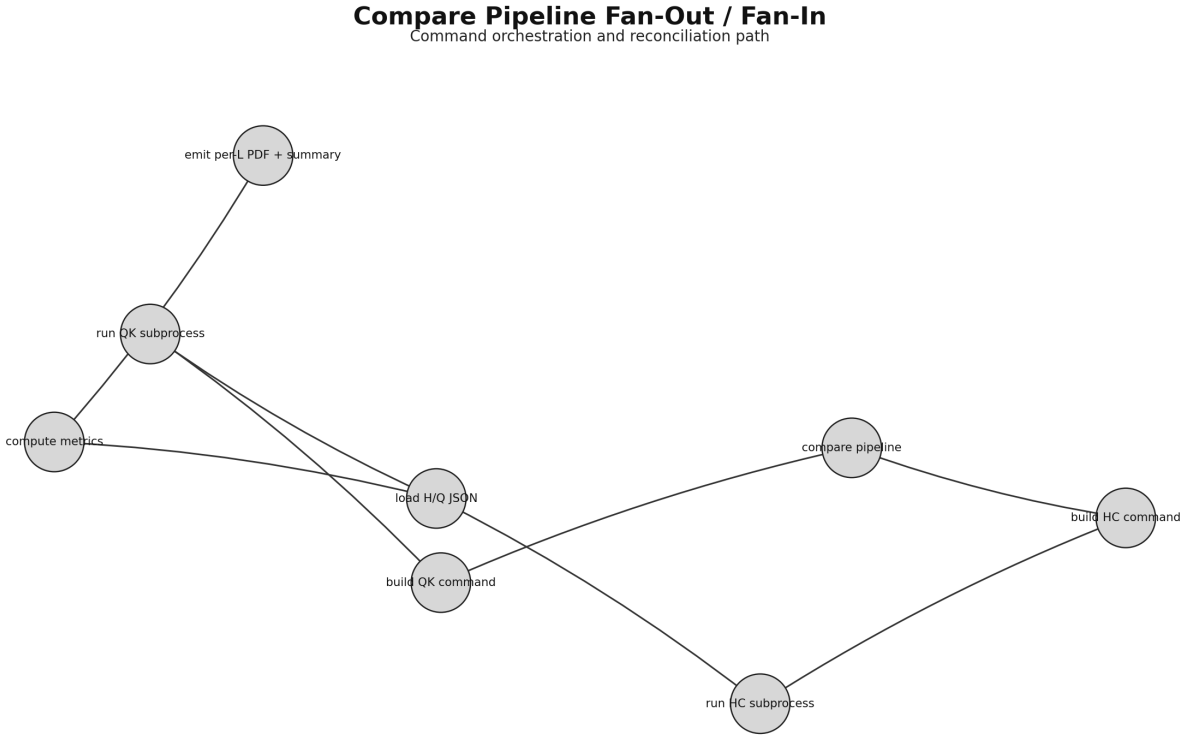
Use: Static vs drive branch. Cross-check against section formulas before drawing conclusions.

1.15.8 A.8 Shared Drive Backend



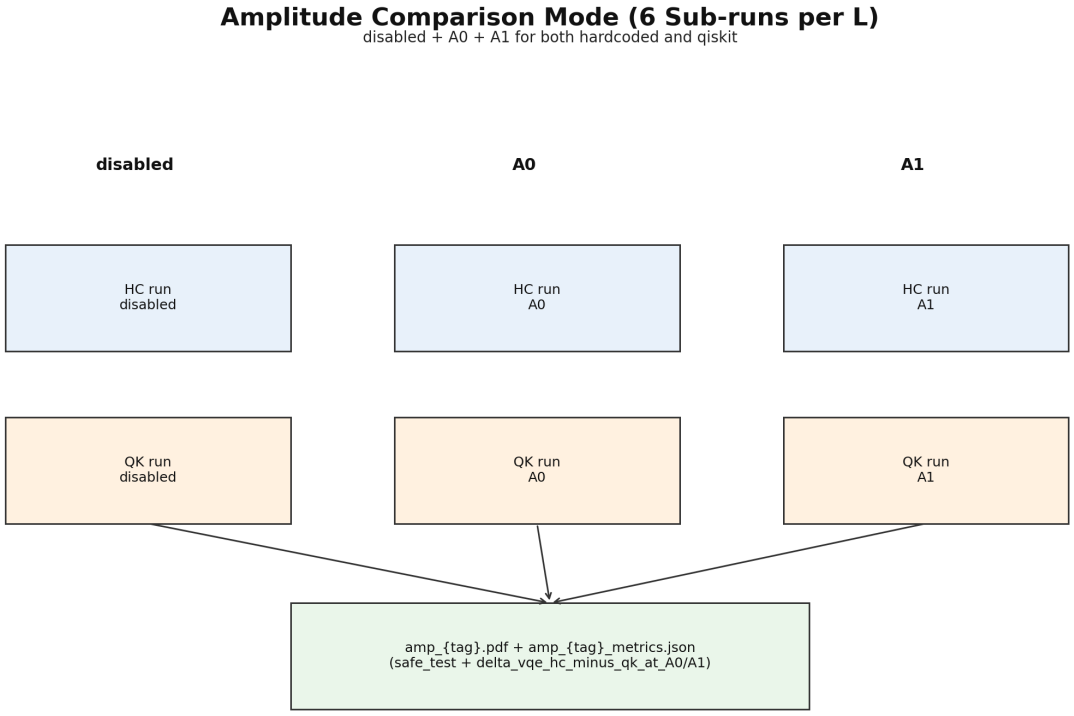
Use: Common drive kernels. Cross-check against section formulas before drawing conclusions.

1.15.9 A.9 Compare Fan-Out/Fan-In



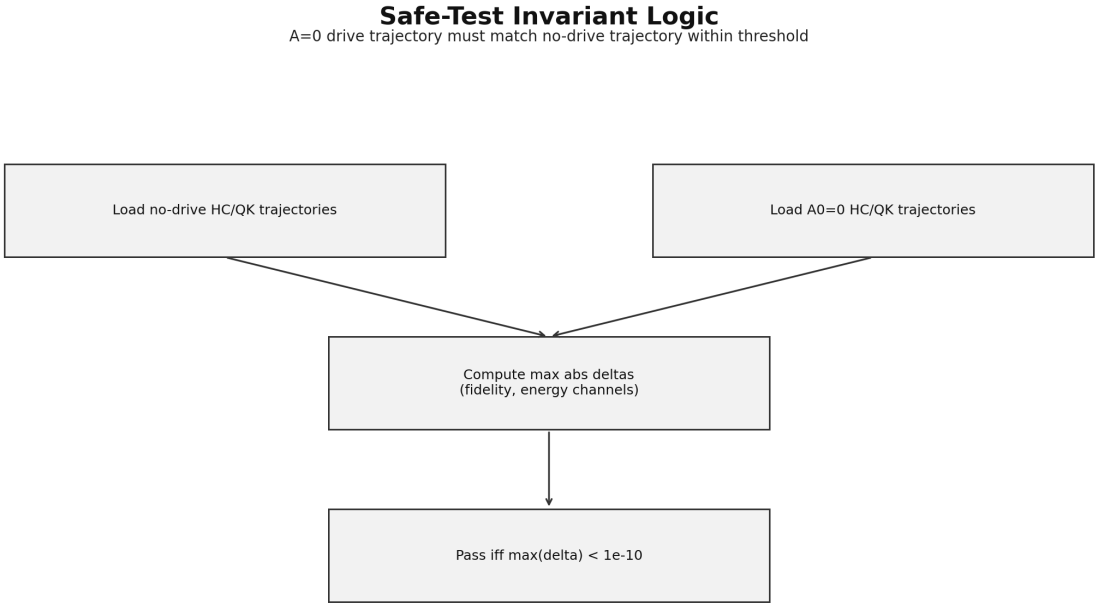
Use: Orchestration path. Cross-check against section formulas before drawing conclusions.

1.15.10 A.10 Amplitude 6-Run Workflow



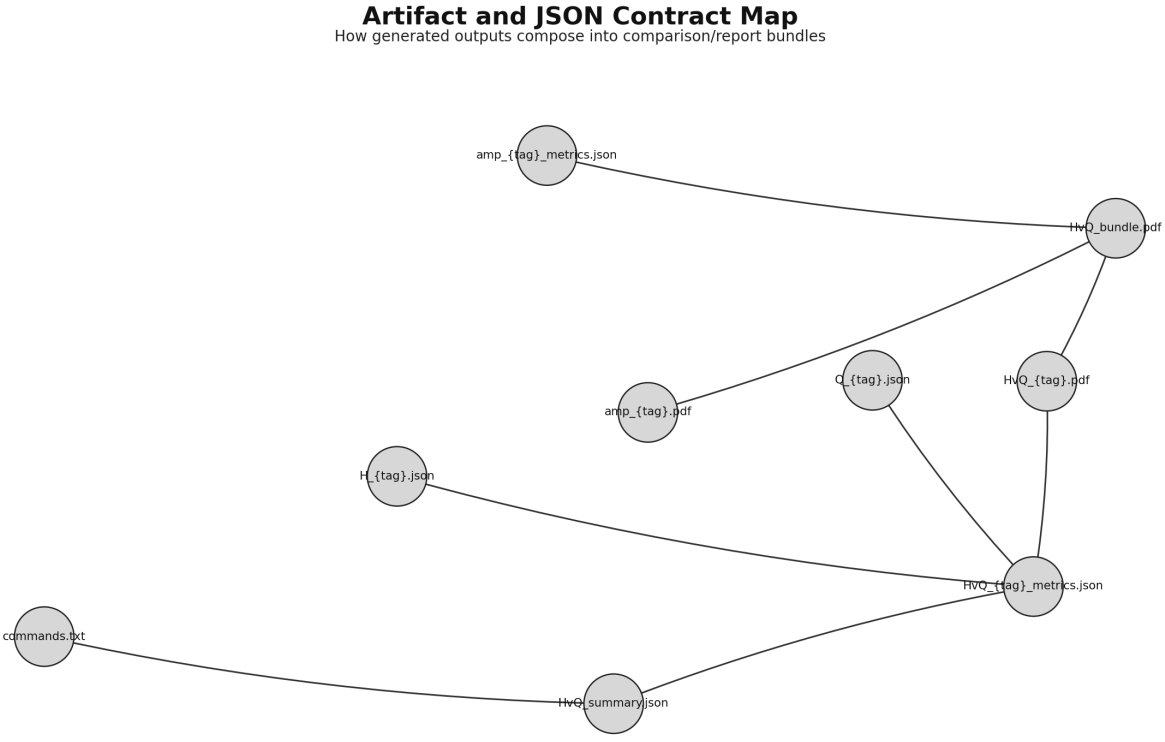
Use: disabled/A0/A1 x HC/QK. Cross-check against section formulas before drawing conclusions.

1.15.11 A.11 Safe-Test Logic



Use: A=0 no-drive equivalence gate. Cross-check against section formulas before drawing conclusions.

1.15.12 A.12 Artifact Contract Map



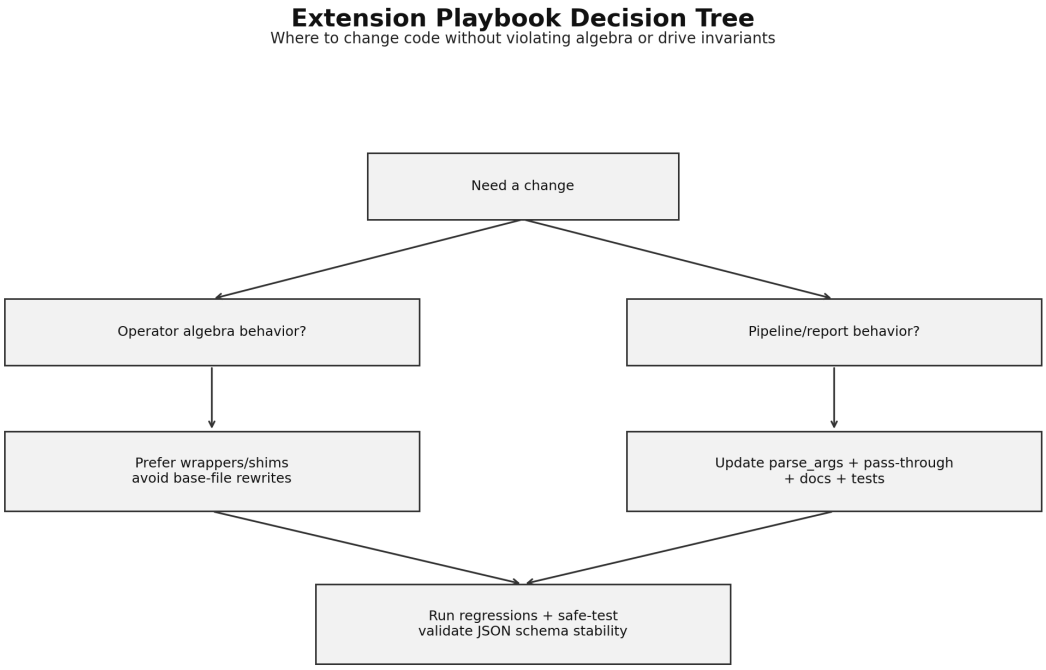
Use: JSON/PDF dependency graph. Cross-check against section formulas before drawing conclusions.

1.15.13 A.13 Test Contract Coverage


```
Detected tests:
- test_compare_drive_passthrough.py
- test_energy_total_observables.py
- test_exact_steps_multiplier.py
- test_fidelity_subspace_projector.py
- test_integration.py
- test_pauli_polynomial_ops.py
- test_tex2text.py
- test_time_potential_drive.py

Contract domains:
- drive pass-through and default routing
- exact-steps multiplier behavior and metadata
- energy_total observable semantics and A=0 checks
- time-dependent drive waveform behavior
- fidelity subspace projector semantics
- import-level integration and aliases
```

Use: Tests as executable spec. Cross-check against section formulas before drawing conclusions.

1.15.14 A.14 Extension Playbook



Use: Safe change decision tree. Cross-check against section formulas before drawing conclusions.

1.15.15 A.15 HF Occupancy Table L2 blocked

HF Occupancy Map (L=2, ordering=blocked)

Half-filling placement of spin orbitals onto qubit bits

site i	qubit q(i,up)	occ_up	qubit q(i,dn)	occ_dn
0	0	1	2	1
1	1	0	3	0

n_up=1, n_dn=1, N_q=4, bitstring q_(Nq-1)...q_0 = 0101

Use: Half-filling map. Cross-check against section formulas before drawing conclusions.

1.15.16 A.16 HF Occupancy Table L3 blocked

HF Occupancy Map (L=3, ordering=blocked)

Half-filling placement of spin orbitals onto qubit bits

site i	qubit q(i,up)	occ_up	qubit q(i,dn)	occ_dn
0	0	1	3	1
1	1	1	4	0
2	2	0	5	0

n_up=2, n_dn=1, N_q=6, bitstring q_(Nq-1)...q_0 = 001011

Use: Half-filling map. Cross-check against section formulas before drawing conclusions.

1.15.17 A.17 HF Occupancy Table L4 blocked

HF Occupancy Map (L=4, ordering=blocked)

Half-filling placement of spin orbitals onto qubit bits

site i	qubit q(i,up)	occ_up	qubit q(i,dn)	occ_dn
0	0	1	4	1
1	1	1	5	1
2	2	0	6	0
3	3	0	7	0

n_up=2, n_dn=2, N_q=8, bitstring q_(Nq-1)...q_0 = 00110011

Use: Half-filling map. Cross-check against section formulas before drawing conclusions.

1.15.18 A.18 HF Occupancy Table L2 interleaved

HF Occupancy Map (L=2, ordering=interleaved)

Half-filling placement of spin orbitals onto qubit bits

site i	qubit q(i,up)	occ_up	qubit q(i,dn)	occ_dn
0	0	1	1	1
1	2	0	3	0

n_up=1, n_dn=1, N_q=4, bitstring q_(Nq-1)...q_0 = 0011

Use: Half-filling map. Cross-check against section formulas before drawing conclusions.

1.15.19 A.19 HF Occupancy Table L3 interleaved

HF Occupancy Map (L=3, ordering=interleaved)

Half-filling placement of spin orbitals onto qubit bits

site i	qubit q(i,up)	occ_up	qubit q(i,dn)	occ_dn
0	0	1	1	1
1	2	1	3	0
2	4	0	5	0

n_up=2, n_dn=1, N_q=6, bitstring q_(Nq-1)...q_0 = 000111

Use: Half-filling map. Cross-check against section formulas before drawing conclusions.

1.15.20 A.20 HF Occupancy Table L4 interleaved

HF Occupancy Map (L=4, ordering=interleaved)

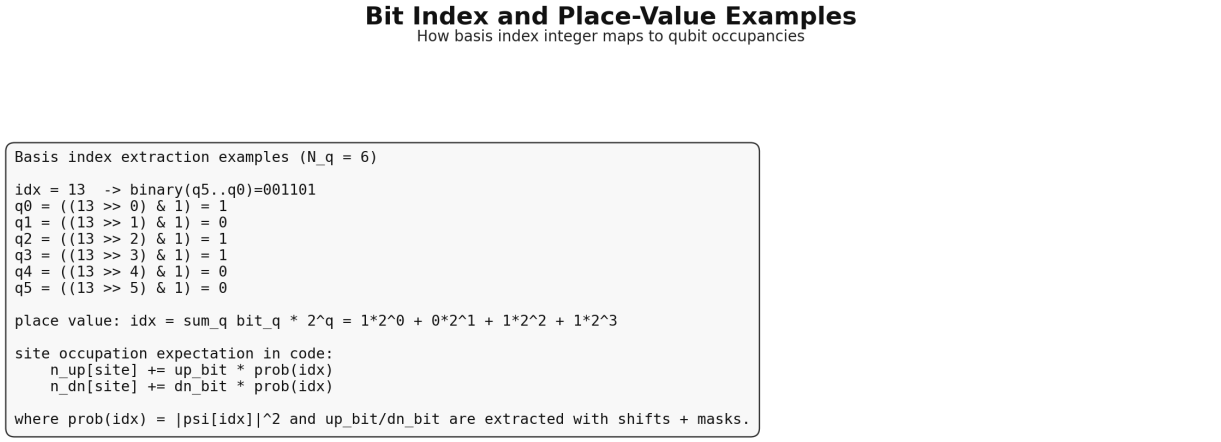
Half-filling placement of spin orbitals onto qubit bits

site i	qubit q(i,up)	occ_up	qubit q(i,dn)	occ_dn
0	0	1	1	1
1	2	1	3	1
2	4	0	5	0
3	6	0	7	0

n_up=2, n_dn=2, N_q=8, bitstring q_(Nq-1)...q_0 = 00001111

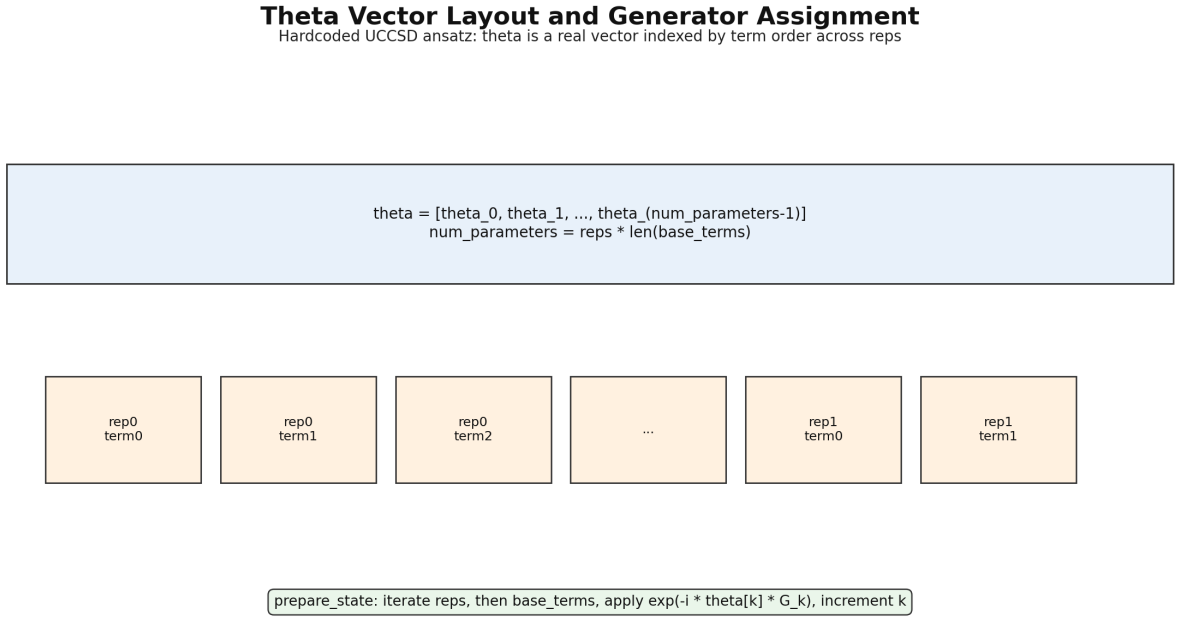
Use: Half-filling map. Cross-check against section formulas before drawing conclusions.

1.15.21 A.21 Bit Index Examples



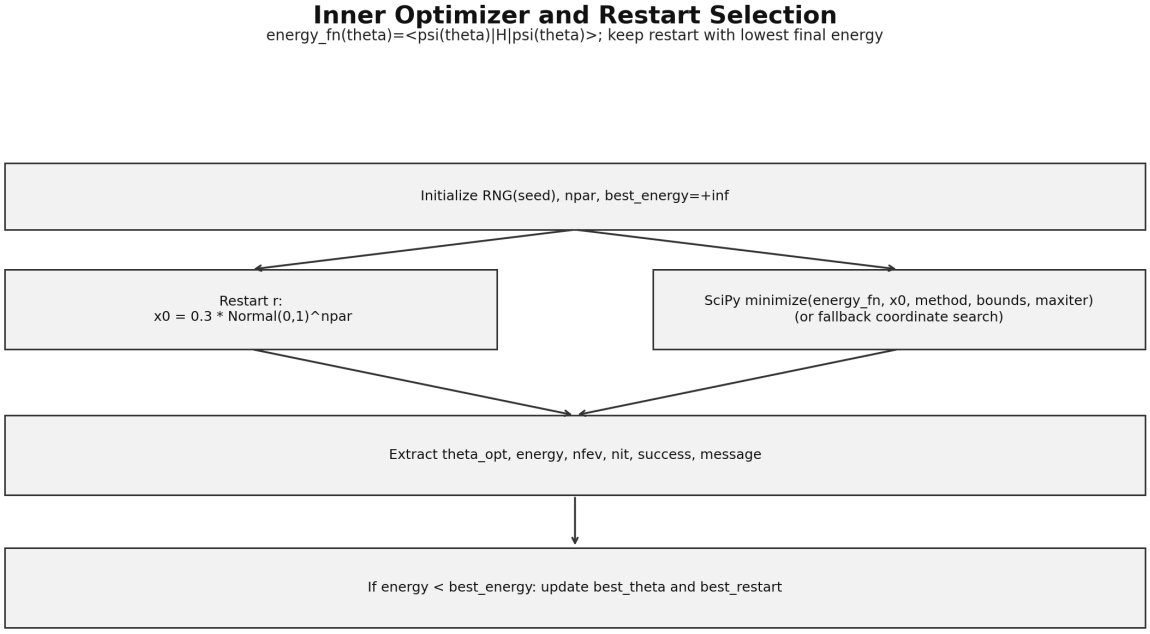
Use: Basis index extraction. Cross-check against section formulas before drawing conclusions.

1.15.22 A.22 Theta Vector Layout



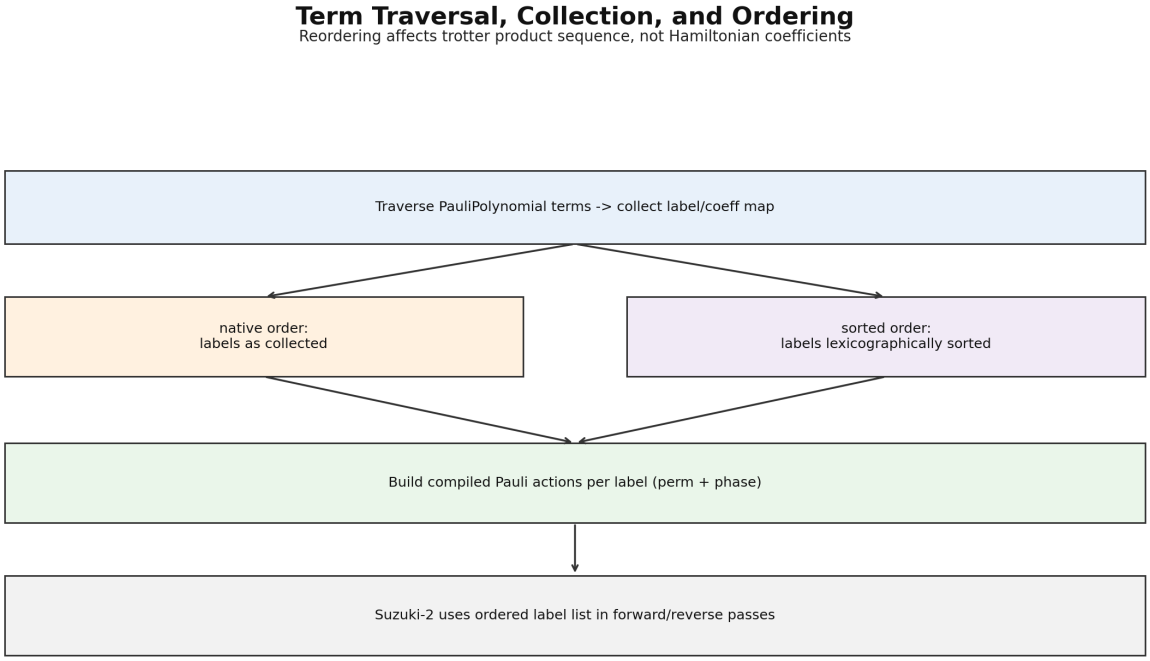
Use: Parameter indexing in ansatz. Cross-check against section formulas before drawing conclusions.

1.15.23 A.23 Optimizer Restart Flow



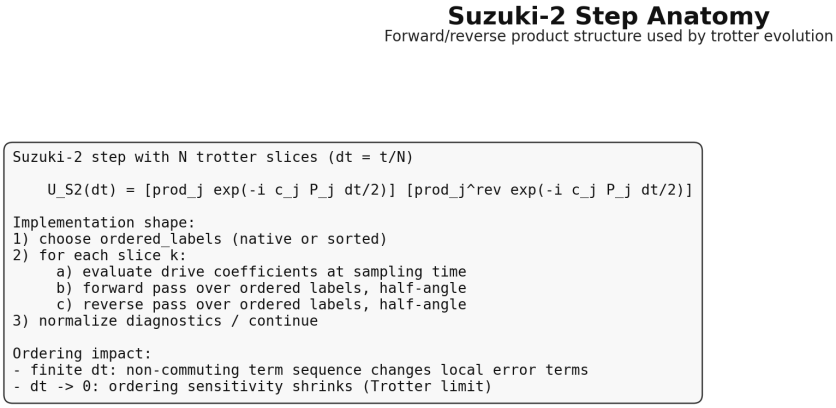
Use: Inner VQE optimization logic. Cross-check against section formulas before drawing conclusions.

1.15.24 A.24 Term Traversal vs Ordering



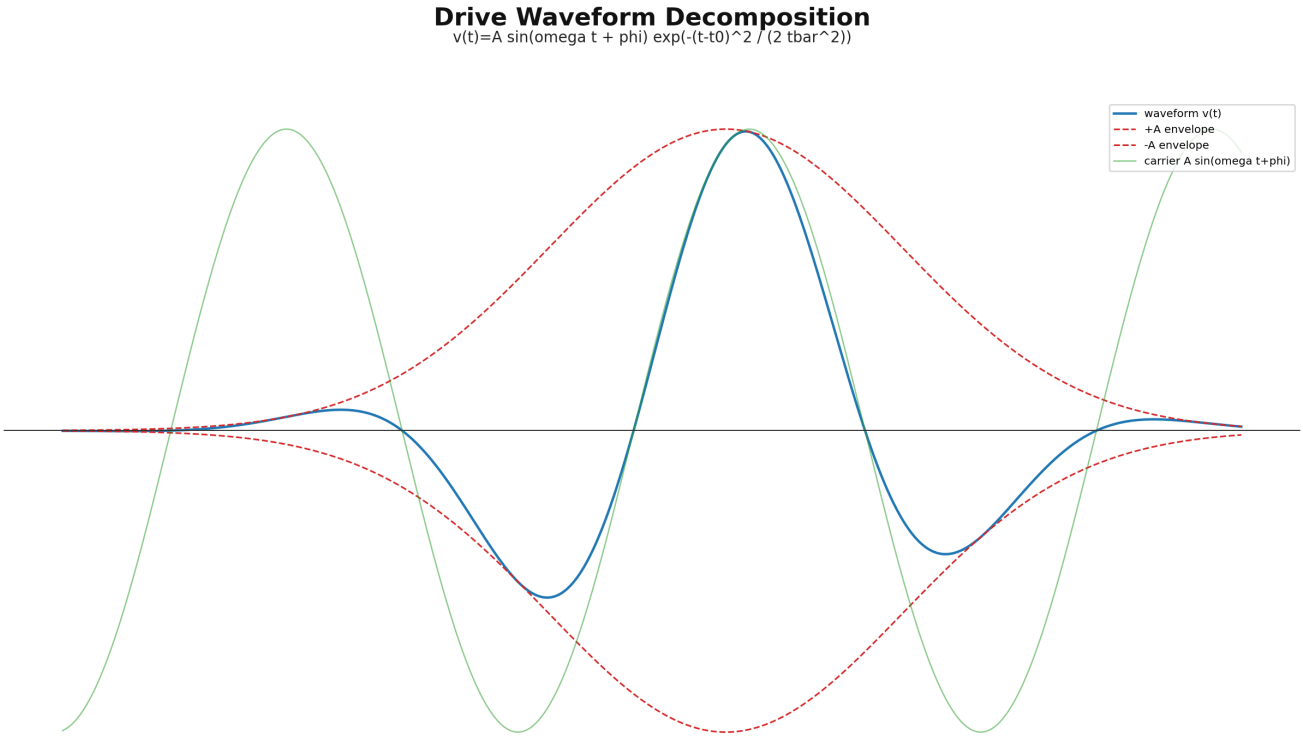
Use: collection/order/compiled actions. Cross-check against section formulas before drawing conclusions.

1.15.25 A.25 Suzuki-2 Anatomy



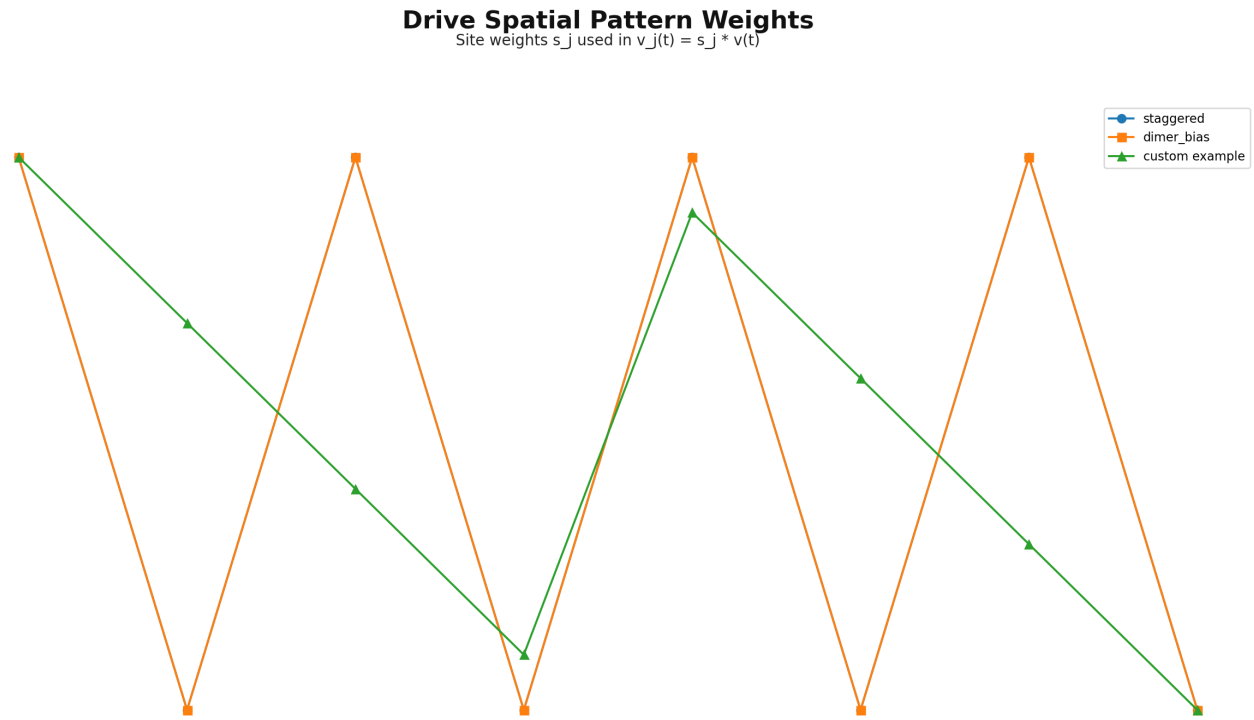
Use: Forward/reverse pass implementation. Cross-check against section formulas before drawing conclusions.

1.15.26 A.26 Drive Waveform



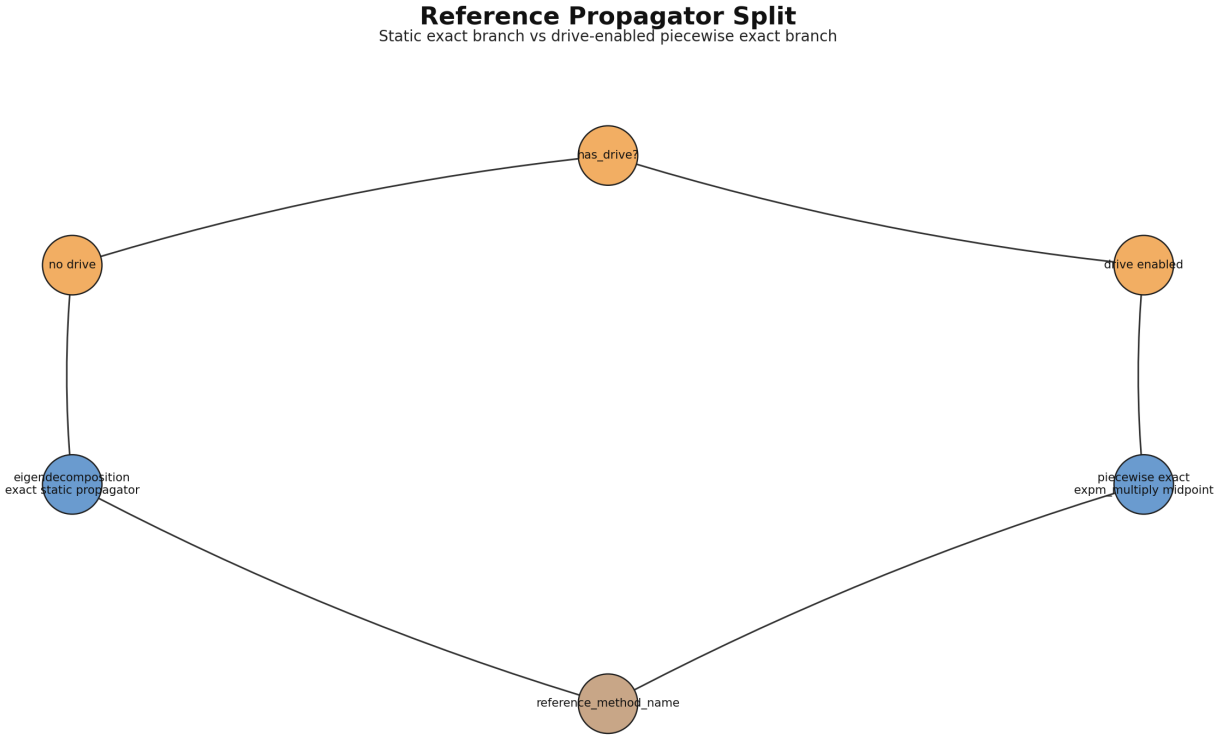
Use: Carrier-envelope decomposition. Cross-check against section formulas before drawing conclusions.

1.15.27 A.27 Drive Spatial Patterns



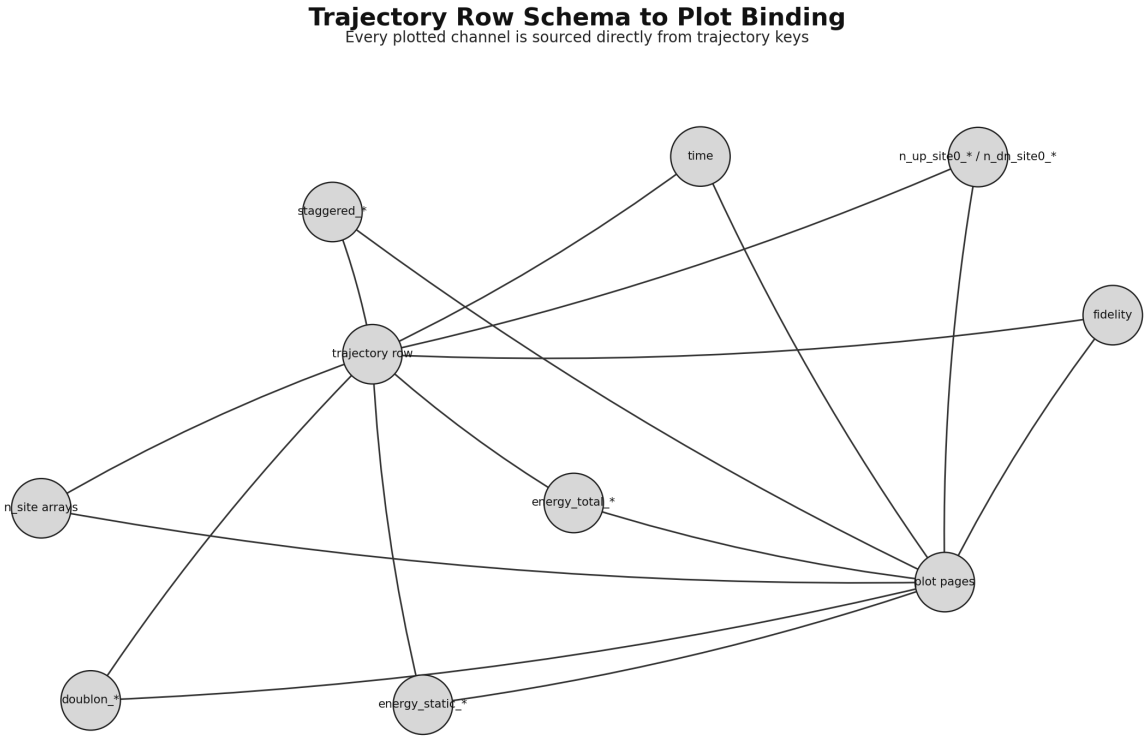
Use: staggered/dimer/custom weights. Cross-check against section formulas before drawing conclusions.

1.15.28 A.28 Reference Propagator Split



Use: static eig vs drive piecewise exact. Cross-check against section formulas before drawing conclusions.

1.15.29 A.29 Trajectory Schema Map



Use: row keys to plot channels. Cross-check against section formulas before drawing conclusions.

1.15.30 A.30 Formula Legend Energy

Formula Legend: Energy Channels
Exact semantics used by pipeline output and figures

```
Energy channels used in trajectory and plots

energy_static_exact(t)  = <psi_exact_gs_ref(t)|H_static|psi_exact_gs_ref(t)>
energy_static_exact_ans = <psi_exact_ansatz_ref(t)|H_static|psi_exact_ansatz_ref(t)>
energy_static_trotter   = <psi_ansatz_trot(t)|H_static|psi_ansatz_trot(t)>

energy_total_exact(t)   = <psi_exact_gs_ref(t)|H_static + H_drive(t0+t)|psi_exact_gs_ref(t)>
energy_total_exact_ans  = <psi_exact_ansatz_ref(t)|H_static + H_drive(t0+t)|psi_exact_ansatz_ref(t)>
energy_total_trotter    = <psi_ansatz_trot(t)|H_static + H_drive(t0+t)|psi_ansatz_trot(t)>

When drive is disabled: energy_total_* == energy_static_*
```

Use: energy observable definitions. Cross-check against section formulas before drawing conclusions.

1.15.31 A.31 Formula Legend Fidelity

Formula Legend: Fidelity
Projector fidelity against filtered-sector propagated ground manifold

```
Fidelity channel

fidelity(t) = <psi_ansatz_trot(t) | P_exact_gs_subspace(t) | psi_ansatz_trot(t)>

where P_exact_gs_subspace(t) projects onto the propagated ground-manifold basis
selected at t=0 from the filtered sector (N_up, N_dn) with energy tolerance:

    E <= E0 + fidelity_subspace_energy_tol

This is not full-state overlap to a single vector; it is projector fidelity
against a (possibly multi-dimensional) reference subspace.
```

Use: projector fidelity definition. Cross-check against section formulas before drawing conclusions.

1.15.32 A.32 Formula Legend Occupancy

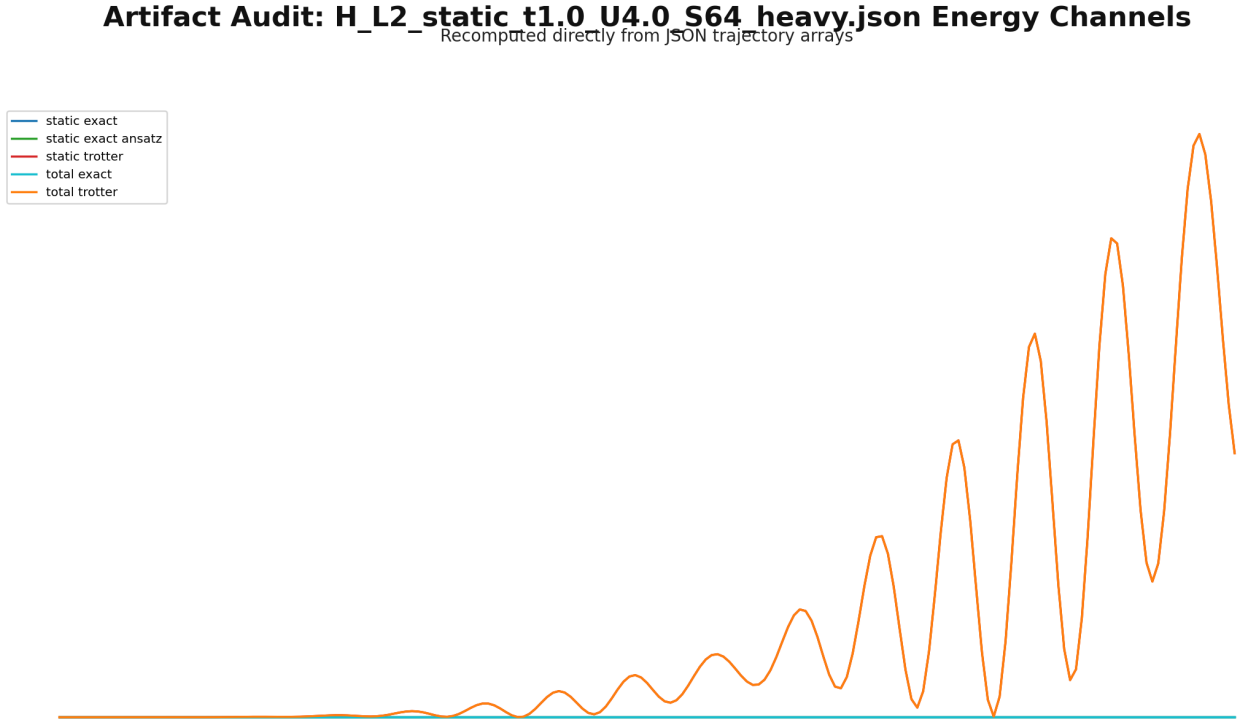
Formula Legend: Occupancy, Doublon, Staggered

Observable definitions implemented in trajectory loop

```
Occupation and doublon channels

For basis index idx with probability p(idx)=|psi[idx]|^2:

    n_up[site] += up_bit(site, idx) * p(idx)
    n_dn[site] += dn_bit(site, idx) * p(idx)

Site-0 channels in trajectory:
    n_up_site0_* = n_up[0]
    n_dn_site0_* = n_dn[0]

Doublon:
    doublon_total = sum_site <n_{site,up} n_{site,dn}>
    doublon_avg   = doublon_total / L

Staggered order from total site density n_site:
    staggered = (1/L) * sum_i (-1)^i * n_site[i]
```

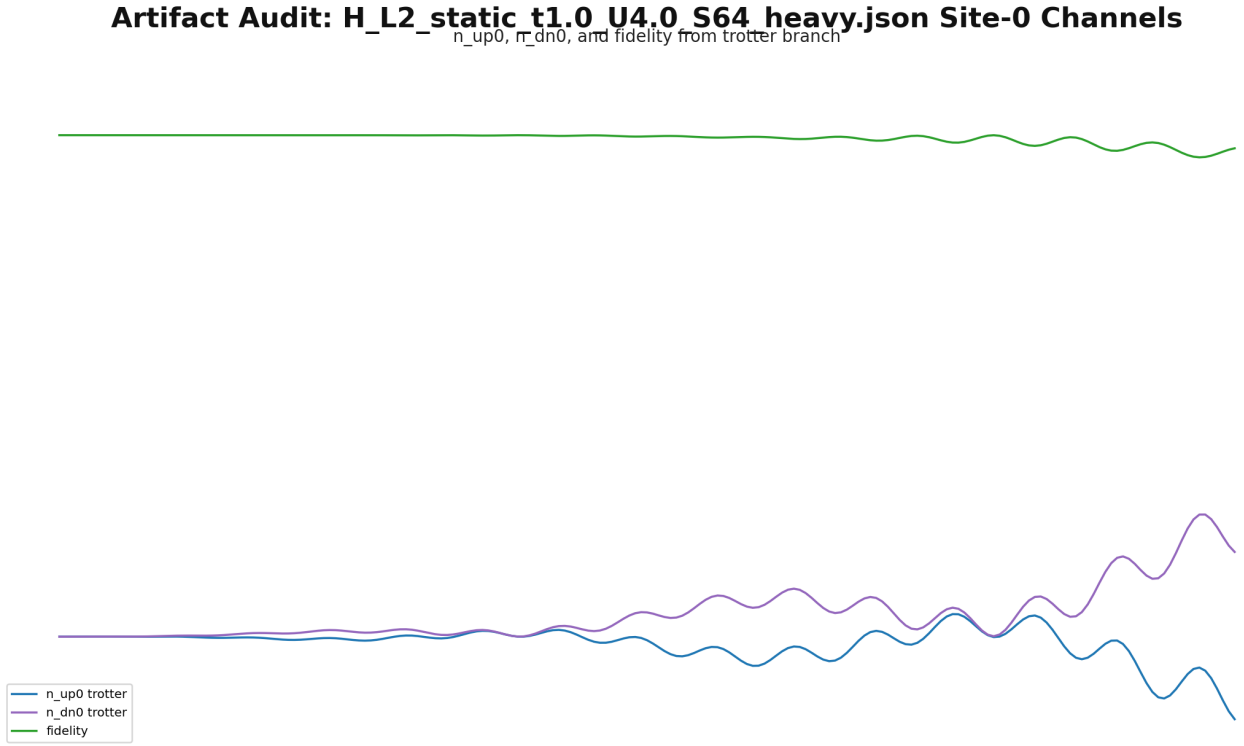
Use: n_up/n_dn/doublon/staggered. Cross-check against section formulas before drawing conclusions.

1.15.33 A.33 Artifact L2 Energy Audit



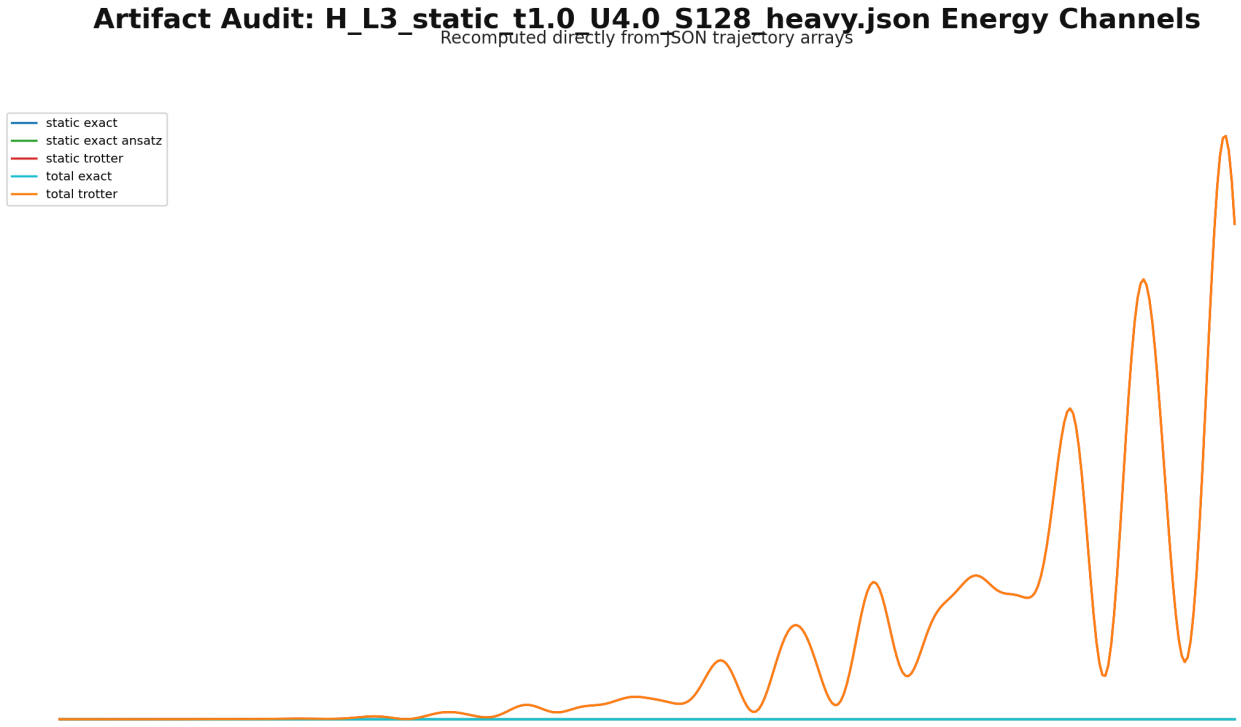
Use: L2 heavy static energy channels. Cross-check against section formulas before drawing conclusions.

1.15.34 A.34 Artifact L2 Site0 Audit



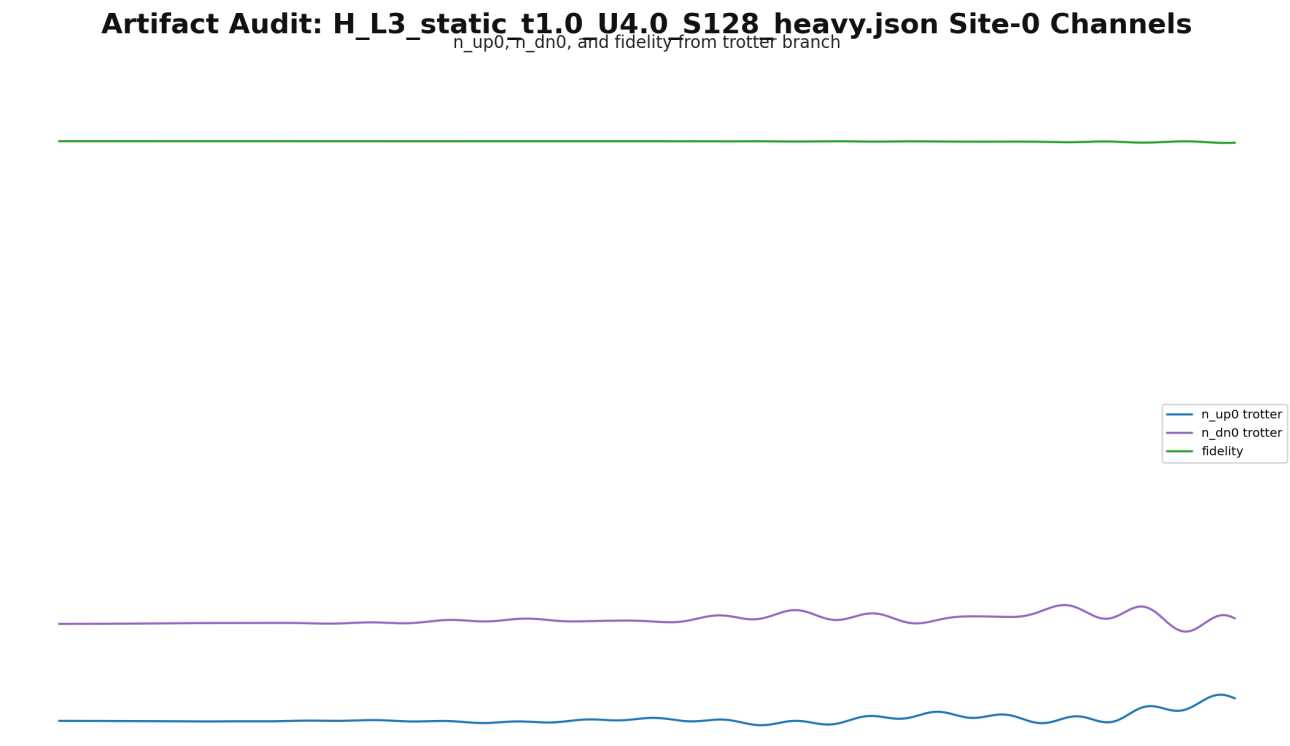
Use: L2 heavy static site-0/fidelity channels. Cross-check against section formulas before drawing conclusions.

1.15.35 A.35 Artifact L3 Energy Audit



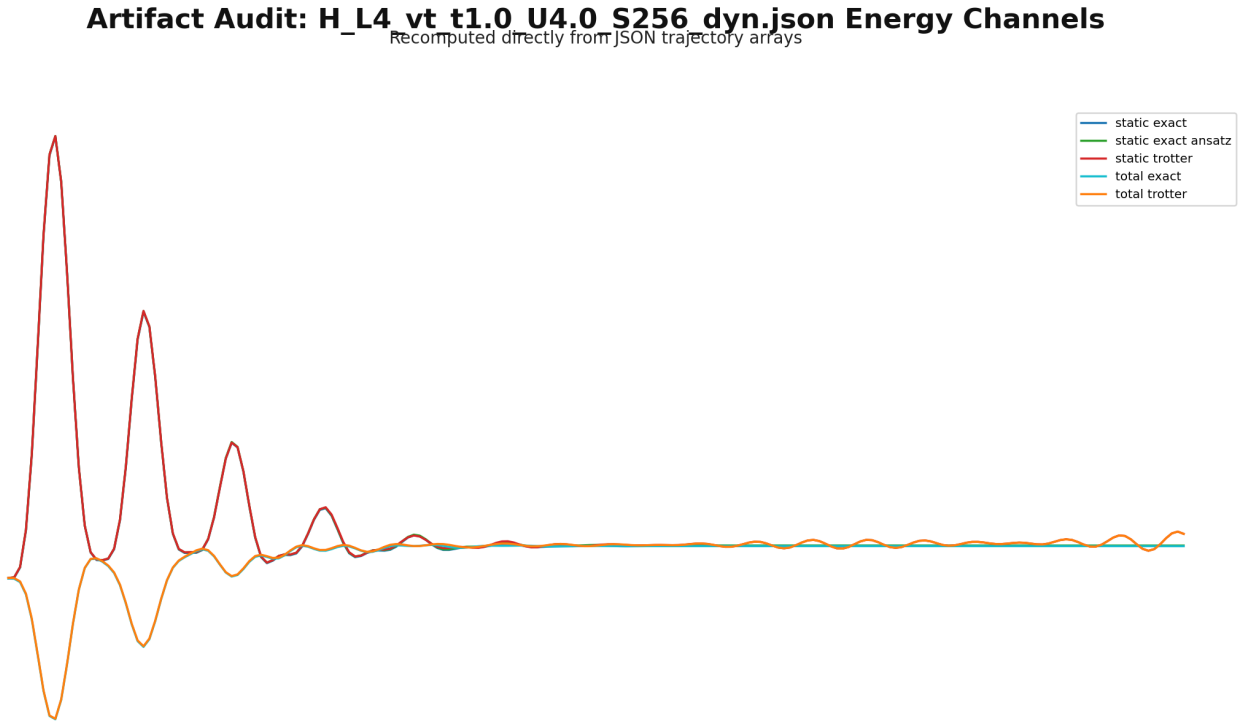
Use: L3 heavy static energy channels. Cross-check against section formulas before drawing conclusions.

1.15.36 A.36 Artifact L3 Site0 Audit



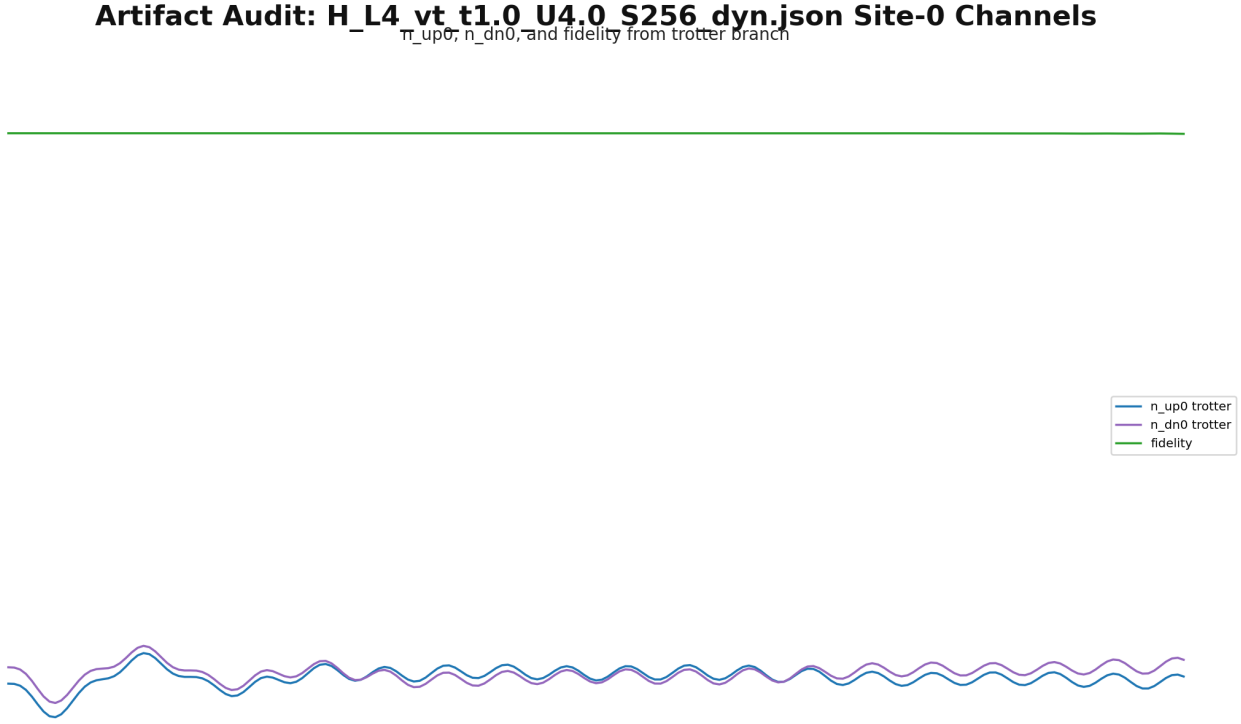
Use: L3 heavy static site-0/fidelity channels. Cross-check against section formulas before drawing conclusions.

1.15.37 A.37 Artifact L4 Drive Energy Audit



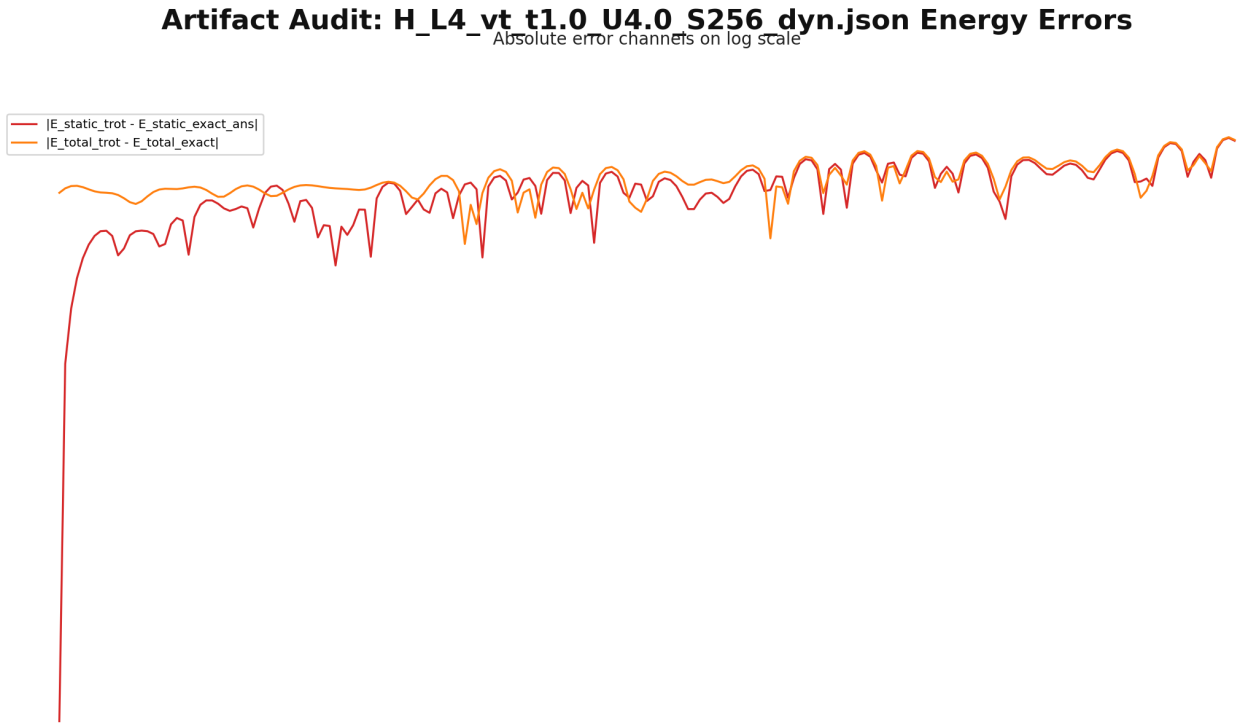
Use: L4 drive-enabled static/total energy channels. Cross-check against section formulas before drawing conclusions.

1.15.38 A.38 Artifact L4 Site0/Fidelity Audit



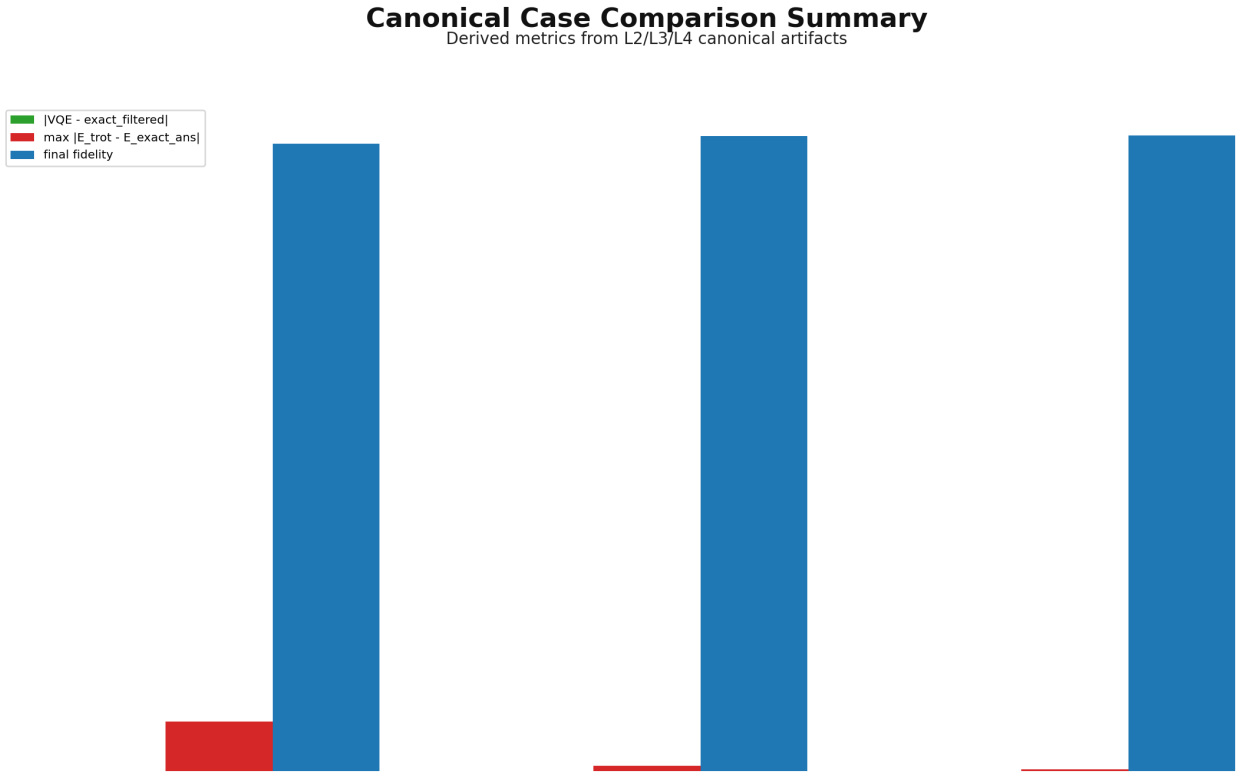
Use: L4 drive-enabled site-0/fidelity channels. Cross-check against section formulas before drawing conclusions.

1.15.39 A.39 Artifact L4 Error Audit



Use: L4 drive energy absolute errors. Cross-check against section formulas before drawing conclusions.

1.15.40 A.40 Case Comparison Summary



Use: L2/L3/L4 derived metric comparison. Cross-check against section formulas before drawing conclusions.

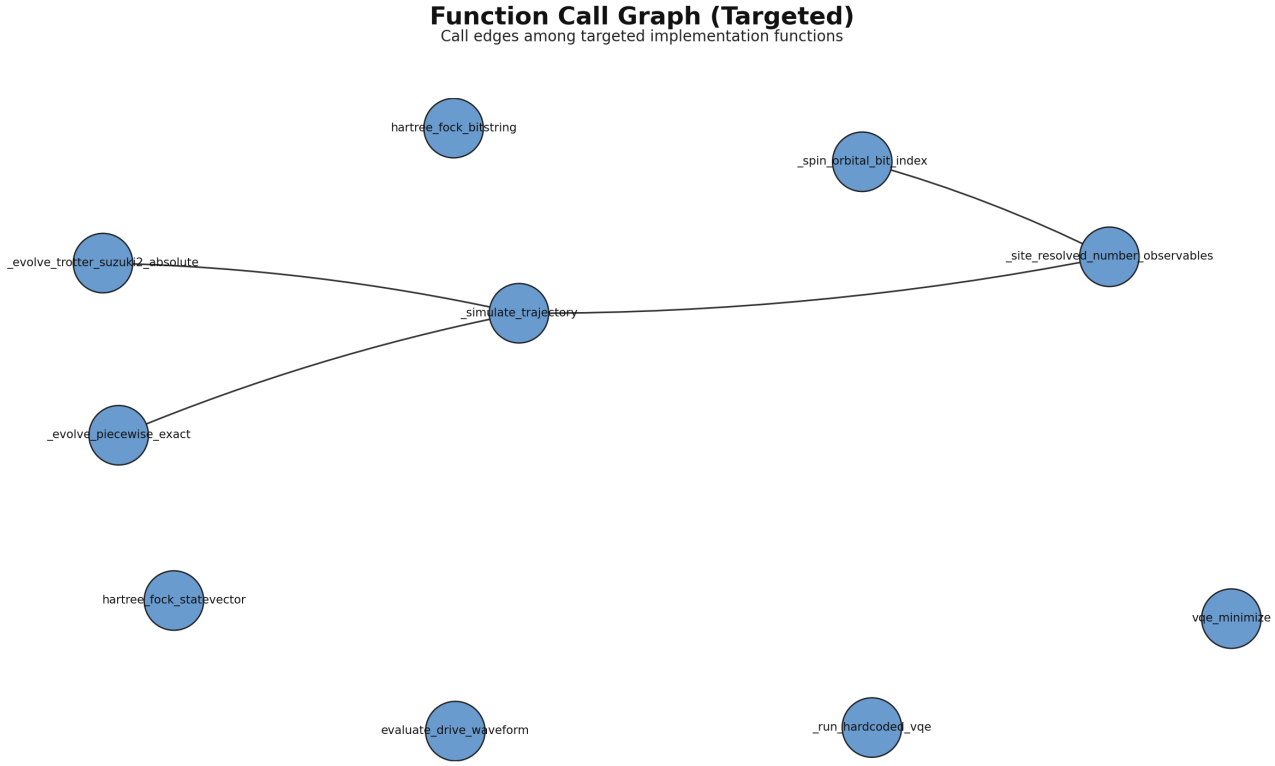
1.15.41 A.41 Function Evidence Map

Function Evidence Map
Target implementation functions with file and line spans

_evolve_trotter_suzuki2_absolute	Fermi-Hamil-JW-VQE-TROTTER-PIPELINE/pipelines/hardcoded_hubbard_pipeline.py	L321-391
_spin_orbital_bit_index	Fermi-Hamil-JW-VQE-TROTTER-PIPELINE/pipelines/hardcoded_hubbard_pipeline.py	L430-436
_site_resolved_number_observables	Fermi-Hamil-JW-VQE-TROTTER-PIPELINE/pipelines/hardcoded_hubbard_pipeline.py	L439-461
_run_hardcoded_vqe	Fermi-Hamil-JW-VQE-TROTTER-PIPELINE/pipelines/hardcoded_hubbard_pipeline.py	L500-577
_evolve_piecewise_exact	Fermi-Hamil-JW-VQE-TROTTER-PIPELINE/pipelines/hardcoded_hubbard_pipeline.py	L875-1033
_simulate_trajectory	Fermi-Hamil-JW-VQE-TROTTER-PIPELINE/pipelines/hardcoded_hubbard_pipeline.py	L1036-1289
evaluate_drive_waveform	src/quantum/drives_time_potential.py	L137-170
hartree_fock_bitstring	src/quantum/hartree_fock_reference_state.py	L101-112
hartree_fock_statevector	src/quantum/hartree_fock_reference_state.py	L115-135
vqe_minimize	src/quantum/vqe_latex_python_pairs.py	L689-794

Use: Target function line spans. Cross-check against section formulas before drawing conclusions.

1.15.42 A.42 Function Call Graph



Use: Call edges among target functions. Cross-check against section formulas before drawing conclusions.

1.15.43 A.43 Invariant Evidence

Invariant Evidence Snippets
Detected invariant contracts with line-level evidence

```
drive_flag_passthrough: found=True
  - Fermi-Hamil-JW-VQE-TROTTER-PIPELINE/pipelines/compare_hardcoded_vs_qiskit_pipeline.py:1169  def _build_drive_args(args: argparse.Namespace) -> list[str]:
  - Fermi-Hamil-JW-VQE-TROTTER-PIPELINE/pipelines/compare_hardcoded_vs_qiskit_pipeline.py:1203  def _build_drive_args_with_amplitude(args: argparse.Namespace, amplitude: float

drive_safe_test_threshold: found=True
  - Fermi-Hamil-JW-VQE-TROTTER-PIPELINE/pipelines/compare_hardcoded_vs_qiskit_pipeline.py:1200  _SAFE_TEST_THRESHOLD: float = 1e-10

jw_mapping_source_of_truth: found=True
  - AGENTS.md:27  - `fermion_plus_operator(repr_mode="JW", nq, j)`
  - AGENTS.md:28  - `fermion_minus_operator(repr_mode="JW", nq, j)`

pauli_string_qubit_ordering: found=True
  - AGENTS.md:21  - **left-to-right = q_(n-1) ... q_0**
  - AGENTS.md:22  - **qubit 0 is rightmost character**

pauli_symbol_convention_exyz: found=True
  - AGENTS.md:16  - Use `e/x/y/z` internally (`e` = identity)

reference_method_metadata: found=True
  - Fermi-Hamil-JW-VQE-TROTTER-PIPELINE/pipelines/hardcoded_hubbard_pipeline.py:51  reference_method_name,
  - Fermi-Hamil-JW-VQE-TROTTER-PIPELINE/pipelines/hardcoded_hubbard_pipeline.py:906  The JSON metadata records ``reference_method`` and

static_vs_driven_reference_split: found=True
  - Fermi-Hamil-JW-VQE-TROTTER-PIPELINE/pipelines/qiskit_hubbard_baseline_pipeline.py:1213  if not has_drive:
  - Fermi-Hamil-JW-VQE-TROTTER-PIPELINE/pipelines/qiskit_hubbard_baseline_pipeline.py:1232  if not has_drive:
```

Use: Line-level invariant snippets. Cross-check against section formulas before drawing conclusions.

1.15.44 A.44 Canonical Metrics Table

Canonical Artifact Metrics Table
Validation and error summary for L2/L3/L4 canonical cases

case	valid	L	drive	|VQE-exact|	max |E_trot-E_exact_ans|
_2_static_t1.0_U4.0_S64_heavy.	True	2	False	3.208e-08	7.799e-02
3_static_t1.0_U4.0_S128_heavy	True	3	False	1.272e-07	8.288e-03
4_L4_vt_t1.0_U4.0_S256_dyn.jso	True	4	True	1.035e-04	2.958e-03

Use: Validation/error summary table. Cross-check against section formulas before drawing conclusions.

1.15.45 A.45 Quality Gate Summary

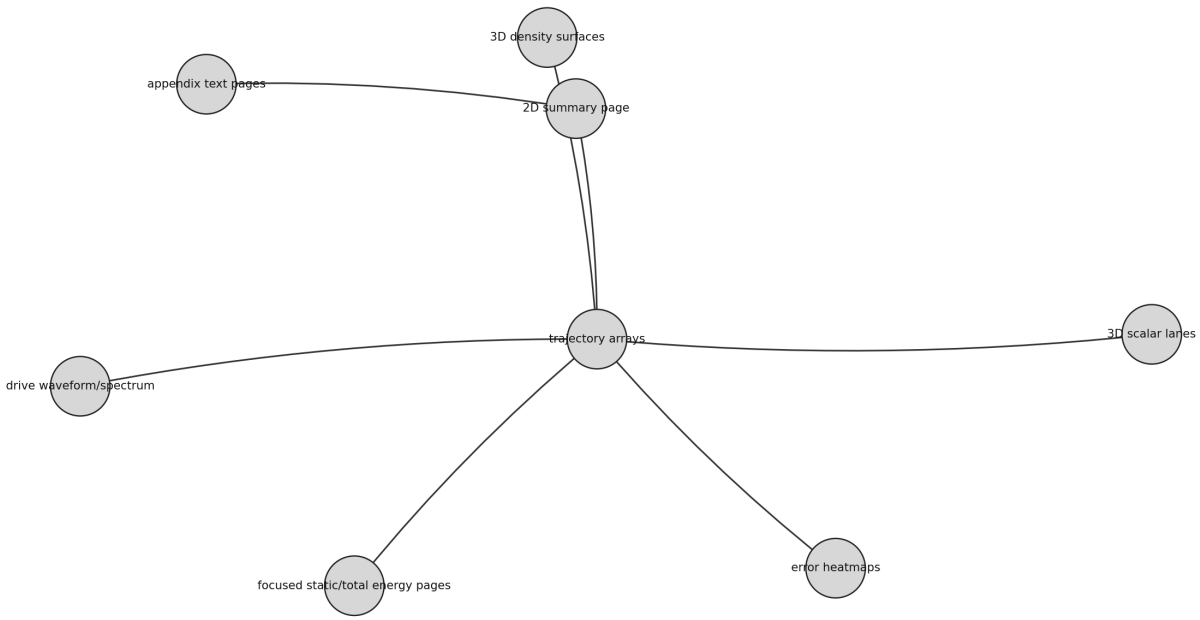
Guide Build Quality Gates
Build script enforces documentation correctness and reproducibility

```
Build quality gates (guide-level)

- operator-core immutability gate
- deterministic diagram regeneration gate
- required section heading gate
- local link/path resolution gate
- figure embedding count gate
- canonical artifact presence gate
- evidence anchor presence gate
- run appendix size cap gate
- pytest snapshot capture gate
- PDF page-count range gate
```

Use: Guide build gate inventory. Cross-check against section formulas before drawing conclusions.

1.15.46 A.46 Plot Meaning Map

Plot Generation Meaning Map
How trajectory channels feed every PDF page family



Use: Trajectory arrays to output pages. Cross-check against section formulas before drawing conclusions.

1.16 Appendix B: Target Function Evidence (Condensed)

AST-derived anchors for implementation review:

- `_evolve_trotter_suzuki2_absolute` -> `Fermi-Hamil-JW-VQE-TROTTER-PIPELINE/pipelines/hardcoded_hubbard_pi` lines 321-391
- `_spin_orbital_bit_index` -> `Fermi-Hamil-JW-VQE-TROTTER-PIPELINE/pipelines/hardcoded_hubbard_pipeline.py` lines 430-436
- `_site_resolved_number_observables` -> `Fermi-Hamil-JW-VQE-TROTTER-PIPELINE/pipelines/hardcoded_hubbard_p` lines 439-461
- `_run_hardcoded_vqe` -> `Fermi-Hamil-JW-VQE-TROTTER-PIPELINE/pipelines/hardcoded_hubbard_pipeline.py` lines 500-577
- `_evolve_piecewise_exact` -> `Fermi-Hamil-JW-VQE-TROTTER-PIPELINE/pipelines/hardcoded_hubbard_pipeline.py` lines 875-1033
- `_simulate_trajectory` -> `Fermi-Hamil-JW-VQE-TROTTER-PIPELINE/pipelines/hardcoded_hubbard_pipeline.py` lines 1036-1289
- `evaluate_drive_waveform` -> `src/quantum/drives_time_potential.py` lines 137-170
- `hartree_fock_bitstring` -> `src/quantum/hartree_fock_reference_state.py` lines 101-112
- `hartree_fock_statevector` -> `src/quantum/hartree_fock_reference_state.py` lines 115-135
- `vqe_minimize` -> `src/quantum/vqe_latex_python_pairs.py` lines 689-794

1.17 Appendix C: Canonical Metrics Snapshot (Condensed)

- All canonical cases valid: `True`
- `H_L2_static_t1.0_U4.0_S64_heavy.json`: L=2, drive=False, |VQE-exact|=3.20842046264147e-08, max|E_trot-E_exact_ans|=0.07798533165196853, final_fidelity=0.9869249072500748
- `H_L3_static_t1.0_U4.0_S128_heavy.json`: L=3, drive=False, |VQE-exact|=1.2720054520798385e-07, max|E_trot-E_exact_ans|=0.008288216471344256, final_fidelity=0.9985135925614043
- `H_L4_vt_t1.0_U4.0_S256_dyn.json`: L=4, drive=True, |VQE-exact|=0.00010348314457075958, max|E_trot-E_exact_ans|=0.0029578796116696005, final_fidelity=0.9993357304628818