PF SIMULATION r34

Private Workflow Development Guide

Reusable scientific programs.
Generated GUI controls.
Private execution and retained results.

A complete operating and development reference

From a mathematical specification or supplied Python program to an activated personal workflow: authoring, contracts, scientific checks, large results, runtime provisioning, and repeatable studies.

19 chapters | 7 importable examples | A complete Floquet spin-chain study

Contents

How to use this guide 3

1. What a private workflow is 4

2. Prerequisites and first use 6

3. Import, validate, activate, and run 7

4. Editing, persistence, and reuse 9

5. Developing a workflow with the AI research assistant 11

6. Bundle structure and a complete first example 14

7. Parameters and generated GUI controls 17

8. Program interfaces and the SDK 19

9. Structured results and presentation 21

10. Validation cases and scientific checks 23

11. Supplied scientific examples 25

12. HTTP API reference 33

13. Runtime provisioning and native integrations 36

14. Architecture and adapter development 38

15. Privacy, project access, and retained evidence 40

16. Limits and troubleshooting 41

17. Reference, checklist, and further reading 45

18. Shared classical reporting and research literature 48

19. Autonomous ISAES after a scientific run 50

Reusable scientific programs, generated GUI controls, and private execution

This guide explains how to turn a scientific Python program into a reusable personal workflow in PF Simulation. You supply a program and its input/output contract, or ask the assistant to draft a program from a precise mathematical specification. PF Simulation retains the proposal, builds its controls, executes its validation cases, and enables repeated runs after you explicitly activate it. The AI research assistant can help prepare that proposal. You can also import one without using an AI provider.

The workflow framework is API-agnostic. A program can use the scientific libraries available in its configured execution environment, including several compatible libraries in the same calculation. This does not mean that every function, hardware service, or external API supported by those libraries is automatically available. The shipped execution adapters are Python adapters; external dependencies and native capabilities must already exist in the isolated runtime.

Product version: r34. This guide documents reusable private workflows and their current ISAES integration. It supplements the AI assistant, SaaS, BYOK, BYOS, scientific integration and autonomous investigation guides. The program-authoring workflow remains explicitly validated and activated by its owner; autonomous investigation is a separate per-workflow delegation.

How to use this guide

Your goal	Start here
Run a supplied workflow through the GUI	Chapters 1-4
Have the assistant adapt a program or draft one from a specification	Chapter 5
Write or adapt a workflow bundle yourself	Chapters 6-10
Use Qiskit, combine scientific libraries, or reproduce the Floquet study	Chapter 11
Call the workflow service from an authenticated client	Chapter 12
Configure the execution environment or extend an adapter	Chapters 13-14
Understand privacy, records, limits, and failures	Chapters 15-17
Analyze retained results and delegate unattended investigations	Chapters 18-19

The PDF edition includes a navigable chapter outline and the same code examples as this Markdown edition.

Code paths in this guide are relative to the extracted PF Simulation package root. Commands using `python` mean the configured Python environment for the relevant task. The development examples do not replace the normal isolated execution path used by the application.

1. What a private workflow is

A private workflow combines a scientific program with a declarative description of its inputs, outputs, and representative checks. The description is a JSON bundle. Its Python modules belong to that private bundle, so a new calculation need not overwrite an existing shared scientific module.

The generic GUI reads the bundle's parameter definitions to create controls. It presents returned metrics, tables, series, data, and downloadable artifacts using built-in rendering. The bundle supplies data and Python source, not browser JavaScript or arbitrary HTML. The saved workflow remains usable after the chat closes and on later application sessions, provided the same account/project context and retained application state remain available.

The scientific program runs only inside the configured execution environment. Importing its JSON into the GUI does not import or execute the program in the web server, browser, or application control process.

1.1 Shared core, private source layers, and private workflows

Component	Purpose	Who changes it?
Shared core	Installed PF application, scientific engines, authentication, validation service, and generic workflow infrastructure	Maintainer through a software update
Private source layer	An existing r34 mechanism for proposed edits to permitted scientific modules in a private source stack	Assistant or user proposal, followed by validation and user activation
Private workflow bundle	New private Python modules, parameters, output definitions, and representative checks for a reusable experiment	User import or assistant proposal, followed by validation and user activation
Executor runtime	Python, installed distributions, native libraries, and execution limits	Deployment operator

A workflow can use the chosen private source profile as its scientific base. The proposal binds that base rather than silently adopting whatever code happens to be active later. Changing the shared core or the relevant private source profile requires a new proposal and validation against the new base.

Creating a private workflow does not add its program to every user's global dropdown. It appears in the owner's **My workflows** interface. In SaaS, another project member does not automatically gain access to it merely because that member can access the project.

1.2 What API-agnostic means in this release

The framework separates the experiment from the interface and execution machinery. A parameter such as `shots`, `samples`, `distance_m`, or `tolerance` is displayed according to its type, not according to a hardcoded engine menu. The same result conventions work for a numerical solver, circuit calculation, or post-processing routine.

For a Python API that is installed and usable offline in the execution image, a workflow can import that API normally. A program may combine libraries in one process when their versions and dependencies

are compatible. A new API does not require a new GUI renderer merely because its scientific functions have different names.

Availability in the ordinary PF research environment is not sufficient to establish availability in the separate assistant executor. Check the actual executor's dependencies. Network-dependent services, proprietary licenses, host devices, and native solvers have separate requirements. The shipped private executor runs with networking disabled; it does not contact remote quantum hardware, cloud services, or license servers on a workflow's behalf.

1.3 Validation has a defined scope

Workflow validation executes the bundle's declared check inputs and evaluates the stated expectations against the returned results. It also checks the input/output contract and records the relevant program, source, and runtime identities. A program cannot establish a trusted pass simply by returning its own `passed` flag.

The expected values in a custom workflow are supplied by its author. Choose them from an independent analytic result, a trustworthy reference, or a separately justified numerical comparison. A passing check demonstrates the tested contract and cases; it does not prove the physical model, cover every parameter combination, or establish experimental accuracy. PF's existing scientific qualification records retain their own scopes.

2. Prerequisites and first use

2.1 Standalone research workbench

Use the normal r34 research installation. On Windows, the existing entry points are `setup_research_worker.bat`, `configure_ai_research_assistant.bat`, and `Launch_PF_Research.bat`. The research workbench uses Python 3.12. `PF_Simulation_GUI.pyw` also uses the configured research interpreter according to the existing launcher rules.

The private workflow interface and retained records do not require a model response. An AI provider is necessary only if you ask the assistant to draft or explain a workflow. Validation and execution require a configured isolated executor even when the proposal was imported manually.

An operator can build the supplied dependency image from the package root:

```
docker build -f Dockerfile.assistant -t pf-ai-research-assistant:r34 .
```

Configure `PF_AI_DOCKER_IMAGE=pf-ai-research-assistant:r34` through the existing assistant settings or deployment environment, then restart the application. Building an image may require internet access for dependencies. Workflow execution itself uses the isolated no-network configuration. The application does not fall back to running generated code directly on the host when Docker is unavailable.

2.2 Hosted SaaS

Sign in to the hosted application and select a project you are permitted to use. The workflow owner is derived from the authenticated user, organization, and project context. A request cannot choose an arbitrary owner, interpreter, image, or host path.

The operator must enable the assistant subsystem, provide its durable state storage, and configure its executor. Existing project permissions, compute/storage admission, and assistant policy controls remain relevant. See `SAAS_DEPLOYMENT.md`, `R34_BYOK_GUIDE.md`, `PF_Simulation_r34_BYOS_Guide.md`, and `docs/SAAS_RESEARCH_POLICY.md` for those deployment and account settings.

2.3 Information to prepare

Before importing or asking the assistant for a workflow, prepare:

1. A complete calculation with an explicit start and finish: either an existing program or a precise mathematical specification for the assistant to implement. Remove interactive prompts and indefinite event loops from the reusable entry point.
2. The inputs to expose, including names, units, defaults, ranges, and permitted choices.
3. The installed distributions and any required native capabilities.
4. The quantities and artifacts to return, with enough labels to interpret them.
5. At least one representative input and independently justified expected result.
6. Captured input files for data that would otherwise require a download during execution.

Small, deterministic checks are useful for first activation. Long parameter sweeps belong in normal runs once the workflow contract is established.

3. Import, validate, activate, and run

The lifecycle is the same in standalone and SaaS. The hosted interface additionally uses the selected project context.

3.1 Import a supplied bundle

Open the **My workflows** tab in the standalone or hosted workbench, or the **My workflows** page in the SaaS portal. Select **Import bundle** to load a JSON file. **New workflow** opens an editable polynomial starter; you can also paste a complete JSON bundle into its proposal editor. Inspect its name, description, requirements, source files, input definitions, output definitions, and checks before saving the proposal.

Select **Save proposal**. Saving a proposal records a draft; it does not execute or activate the code. The initial bundle includes its defaults, so the generated form can be populated without asking the AI to design a new interface each time.

A bundle exported from another installation is still a new proposal when imported. Its source may be reusable, but the old installation's activation does not confer approval on the current source and runtime.

3.2 Validate the exact proposal

Select **Validate** for the saved version. Wait for the operation to finish, then inspect every reported case and any execution error. Check the calculated output, expected value, tolerance, and runtime information rather than relying only on a green status label.

If validation fails, edit the program, inputs, or justified expectations as appropriate, save a new proposal version, and validate that version. Do not change an expected result merely to match unexplained output. A missing dependency or unavailable executor is an environment problem; rewriting the expected numerical value will not repair it.

3.3 Activate explicitly

After a successful, intact validation, select **Activate**. Activation is a separate user operation. The assistant's model tools cannot activate the proposal or silently approve its own code.

Activation binds the workflow to the validated bundle and execution context. If another tab changes the workflow, reload its current record before acting. Revision conflicts protect against applying an action to an obsolete version.

3.4 Change inputs and repeat

Use the generated parameter form, then select **Run workflow**. Inspect the resulting values, plots, tables, artifacts, and retained run record. Change a parameter and run again to perform another calculation using the same activated program. An ordinary change within the accepted input schema does not require the AI to regenerate code or rebuild the GUI.

The workflow remembers ordinary last-run inputs for later use. File controls require explicit file selection; a browser cannot silently repopulate them. The earlier run retains its captured input evidence, but that does not make a new run's file selection implicit.

Saved records belong to application state, not to the open chat. Closing the chat does not remove the workflow. Deleting the application state, changing account/project context, or losing project permission can make it unavailable.

3.5 Inspect results and run history

Every execution is a distinct run. Use **Run history** and **Inspect run** to inspect prior outcomes rather than assuming the current form describes a previous run. **Download run JSON** saves the record. **Reuse these settings** copies its ordinary inputs into the current form, while **Reset to defaults** restores the current bundle defaults. A run records the version and inputs that actually executed.

Tables and charts should include clear names and units from the workflow contract. Downloadable artifacts are retained files from the bounded output directory, with integrity information. Complex scientific outputs that do not fit the built-in displays can be returned as JSON data or an artifact for external analysis.

Failed execution is distinct from an empty successful result. Inspect the error and logs, then correct the cause or adjust a legitimately insufficient resource limit through the operator.

4. Editing, persistence, and reuse

4.1 Versions and revisions

A **version** identifies an immutable bundle proposal. Editing the program, parameter definitions, outputs, requirements, or checks creates a new version. That edit deactivates the previous working selection for the workflow; validate and activate the new version before using it.

A **revision** is the current record's concurrency value. Actions can change the revision even when they do not create a new program version. API clients must use the most recently returned revision. The UI handles that value as part of its normal request flow.

Action	Effect
Save a new workflow	Creates a private draft
Save an edited proposal	Creates a new immutable version and requires validation/activation
Validate	Runs declared checks and records their outcome
Activate	Enables the validated current version for execution
Run with different accepted inputs	Creates another run of the activated version
Deactivate	Stops new runs until the workflow is activated again
Archive	Removes the workflow from the usual menu while preserving its retained records
Restore workflow	Returns the same archived workflow inactive, preserving history and requiring validation
Permanently delete workflow	Removes an archived workflow and its associated records/files after scope confirmation
Export and import	Transfers a reusable proposal; the destination validates it again

4.2 Adopting a changed source profile or runtime

A workflow's scientific base is pinned when proposed. To adopt a newer shared release or a changed active private source stack, create a new proposal version and validate it against that base. Existing historical runs remain associated with the old identity.

If the executor image changes, its identity changes. Revalidation is required before activation or running against the changed runtime. Reusing the same image tag does not guarantee the underlying image is unchanged; the executor records the resolved image identity.

4.3 Export is not a complete deployment backup

Export preserves the workflow definition and private program in a reusable bundle. It does not install dependencies, provide credentials, grant project permissions, or carry an activation that can bypass destination validation.

In **Saved versions**, choose **Version to export** and select **Export selected version**. **Export bundle** exports the current version. The export API can select a retained immutable version using `?version=N`.

Import an older exported version as a new proposal to re-establish validation on the current base; there is no silent rollback of activation.

For a reproducible study, retain the workflow bundle together with its execution records, input artifacts, dependency/runtime information, and scientific explanation. For operational disaster recovery, the operator must back up database metadata and private assistant state together, following the existing r34 procedures. A browser download alone does not replace that backup.

4.4 Archive, restore and permanent deletion

Archiving is a retention operation, not secure erasure. Select **Show archived** to inspect retained entries. **Restore workflow** returns the same workflow inactive with its identity, versions and run history. Validate before activation; changed source or layers can require a new workflow version. Export/Import remains available for backups, sharing and independent copies.

To remove an archived workflow permanently, choose **Permanently delete workflow** and review the confirmation. Its versions, validation records, executions and associated files are removed; separately saved conversations, layers and unrelated results remain. There is no undo. **Cleanup operations** shows **Deletion pending** and **Retry deletion** if physical cleanup is interrupted. Missing files are safe to retry; shared blobs remain while retained objects reference them.

In the AI assistant, **Delete conversation** removes chat messages, provider context, checkpoints and conversation attachments while preserving technical evidence needed by saved layers or results. These are end-user actions, not AI tools, and require no AI key. Exported copies, external-provider records and backups are outside application deletion. See `R34_LIFECYCLE_GUIDE.md` for complete recovery and upgrade instructions.

5. Developing a workflow with the AI research assistant

5.1 Start from an existing program

Select **Create with AI** from **My workflows**. This opens a private workflow-authoring conversation without requiring a completed simulation. In the SaaS portal, it opens the hosted workbench in the same project context. Attach your `.py` program or paste it into the message, describe the controls and outputs you need, and select **Send**. The assistant can prepare a proposal using its bounded workflow tools; use **Open My workflows** to return to the saved proposal and its generated interface.

An authoring conversation explicitly states that no simulation has executed or been validated. Its existence does not provide numerical evidence. The model cannot use the old saved-run rerun tool to invent a baseline in that context. Validation is performed later through the workflow service.

You can still request a proposal from an existing run-linked assistant conversation when the saved run is relevant. In either path, you review the scientific meaning, validation, and activation.

A useful request is:

> Turn this program into a private workflow called "Bell-state probability sweep". Expose angle in radians and the number of sweep points. Return the final probability, a labelled probability curve, and a CSV artifact. Keep the calculation in the supplied program. Include an analytic check at angle zero and a probability-normalization check. Use the installed Qiskit runtime, declare its requirement, and leave the proposal for my validation and activation.

For another engine, replace the program and requirements while retaining the same input/output instructions. Tell the assistant which values are measurements, reference data, numerical results, or assumptions. Do not ask it to invent a passing experimental comparison.

5.2 Start from a mathematical specification

You can request a new Python implementation without supplying an existing scientific program. The same proposal contract accepts complete private Python source. The default **Polynomial sweep** starter provides the bundle structure, `run(inputs, context)` interface, parameter definitions, output declarations, and check format. Its polynomial calculation does not supply a quantum model or establish that another model is correct.

First save the starter as an inactive proposal if you want the assistant to inspect it through `list_workflows` and `read_workflow`. Open **Create with AI**, identify that starter by its saved workflow name or ID, and request a separate new workflow. State explicitly that the assistant should replace the example calculation with the specified model and should leave the new proposal inactive. Attaching the starter JSON is another way to supply its structure.

Provide these scientific decisions before requesting the implementation:

- Equations, gate or solver sequence, numerical method, and units.
- Initial state, boundary conditions, indexing conventions, and supported input ranges.
- Noise or stochastic assumptions, including sampling and reproducibility requirements.
- Exact definitions of observables, reductions, and comparison quantities.
- Independently justified limiting cases and tolerances.
- Required plots, tables, full datasets, and artifact formats.

Ask the assistant to identify unresolved assumptions before drafting. A title, poster, or library name alone may leave scientifically different implementations possible. Section 11.7 specifies a complete Floquet example; section 11.8 provides a request you can use with that specification.

The model may produce an incomplete or incorrect implementation. Review the full source and declared checks, then use the end-user validation and activation controls. This route does not add an autonomous generate-run-debug loop to the r34 assistant. Its workflow tools can propose source, but cannot validate, execute, or activate that workflow themselves.

5.3 Review the proposal before executing it

Check the following concrete items:

- Each GUI control maps to an input the program actually uses.
- Units, defaults, and bounds are meaningful for the calculation.
- File inputs replace unsupported download or host-path assumptions.
- The declared dependency names match distributions in the executor.
- Expected values come from a defensible reference and tolerances fit the numerical method.
- Returned values are actually calculated and are labelled correctly.
- Runtime and dataset sizes fit the available resources.

The assistant can help explain or revise the proposal. Saving a revision still creates a new version that must pass validation before activation.

5.4 What the model can and cannot do

The workflow tools allow the assistant to list, read, and propose workflows within its permitted context. They do not let it choose another user's owner scope, install packages, alter authentication, change the shared core, activate a proposal, or choose an arbitrary host command/runtime.

The ordinary assistant investigation remains bounded by the configured provider, tool, and time limits. A partial or interrupted model response is not a finished proposal. A model's statement that a program is correct is not execution evidence.

A complete program is encoded inside the proposal tool arguments and can require a substantial model response. `PF_AI_MAX_OUTPUT_TOKENS` defaults to 4,096 per response and is configurable from 128 to 32,768, subject to the selected provider/model's own support. This budget is separate from the 2 MiB bundle limit. `PF_AI_TIMEOUT_SECONDS` defaults to 45 seconds per request, while `PF_AI_TOTAL_SECONDS` defaults to 180 seconds for the investigation; their configured upper bounds are 120 and 600 seconds respectively. These are resource settings, not guarantees that a complex program will be completed. Ask the operator to review the supported settings if generation is truncated or times out. Preserve a completed bundle and use manual import when appropriate; do not mistake partial tool arguments for a saved valid proposal.

Manual JSON import, validation, activation, and later runs do not require a model call. Once a workflow is established, the generated form and stored program provide the reusable interface.

5.5 Disclosure and external AI

Supplying code to the assistant may disclose permitted source excerpts to the configured model provider. Hosted organization/project research policies and required consent apply before those provider requests. Review the selected policy and consent information before authorizing transmission.

An external model interaction and a workflow execution are separate operations. The model may be remote under the applicable policy; the shipped scientific executor still has no network. A successful conversation does not mean a remote scientific service can be reached from a private job.

6. Bundle structure and a complete first example

A bundle is a finite JSON object. Its code is represented by UTF-8 text in the `files` array. Use JSON-aware tools to construct it; manually escaping a long program into one JSON string is error-prone.

The following program computes an ordinary polynomial sweep. It uses only the Python standard library and provides an independently checkable result: with coefficient 2 at sample coordinate 2, the value is 8.

```
def run(inputs, context):
    x = list(range(inputs['samples']))
    y = [inputs['coefficient'] * value ** 2 for value in x]
    return {
        'metrics': {'last': y[-1]},
        'series': [{'name': 'curve', 'x': x, 'y': y}]
    }
```

6.1 Generate the JSON bundle

Save this authoring script as `make_polynomial_workflow.py` in your own development folder. Running it writes a JSON proposal; it does not execute the workflow or register it in PF Simulation.

```
import json
from pathlib import Path

program = """def run(inputs, context):
    x = list(range(inputs['samples']))
    y = [inputs['coefficient'] * value ** 2 for value in x]
    return {
        'metrics': {'last': y[-1]},
        'series': [{'name': 'curve', 'x': x, 'y': y}]
    }
"""

bundle = {
    'schema_version': 1,
    'name': 'Polynomial sweep',
    'description': 'Evaluate a polynomial and plot its samples.',
    'adapter': 'python',
    'entrypoint': 'main.py:run',
    'requirements': [],
    'parameters': [
        {
            'name': 'coefficient', 'label': 'Coefficient',
            'type': 'number', 'default': 2,
            'minimum': -100, 'maximum': 100, 'required': True
        },
        {
            'name': 'samples', 'label': 'Samples',
            'type': 'integer', 'default': 5,
            'minimum': 2, 'maximum': 100, 'required': True
        }
    ],
    'outputs': [
        {'name': 'last', 'label': 'Final value', 'kind': 'number'},
        {'name': 'curve', 'label': 'Polynomial', 'kind': 'series'}
    ],
    'checks': [
        {
            'name': 'Known polynomial',
            'inputs': {'coefficient': 2, 'samples': 3},
            'expectations': [
                {
```

```
                    'output': '/metrics/last',
                    'operator': 'close', 'value': 8,
                    'atol': 1e-12, 'rtol': 0
                }
            ]
        }
    ],
    'files': [{'path': 'main.py', 'content': program}]
}

Path('polynomial.workflow.json').write_text(
    json.dumps(bundle, indent=2, allow_nan=False) + '\n',
    encoding='utf-8'
)
```

Run the authoring script, then import `polynomial.workflow.json` through **My workflows**:

```
python make_polynomial_workflow.py
```

Validate and activate it. With the default five samples and coefficient 2, the curve is 0, 2, 8, 18, 32, and the final value is 32. Change the coefficient to 3 and run again; the final value becomes 48. These values are calculations by the supplied program, not canned PF outputs.

6.2 Top-level fields

Field	Meaning
`schema_version`	Bundle contract version; use integer 1
`name`	Human-readable workflow name
`description`	Purpose, assumptions, and concise usage description
`adapter`	Registered execution adapter: `python` or `python_script`
`entrypoint`	Private Python path and, for `python`, callable name
`requirements`	Installed distribution names and exact version requirements
`parameters`	Typed definitions used by request validation and GUI controls
`outputs`	Named output declarations used to interpret returned results
`checks`	Representative inputs and expected-result assertions
`files`	Private Python files, each with relative `path` and text `content`

Unknown fields and unsupported shapes are rejected rather than treated as instructions. The bundle cannot supply an owner identity, Docker image, shell command, credentials, or an alternative validation engine.

6.3 File layout and modules

Keep the entry point small. Place calculation, parsing, and reporting helpers in additional private Python modules. For example:

```
main.py
calculation.py
reporting.py
```

The `files` array contains those three relative paths and their source text. `main.py` can import `calculation` and `reporting` in the isolated program environment. Keep module names distinct from installed libraries and from PF's own module names to avoid accidental shadowing.

The bundle's new files belong to its private namespace. They do not become shared core files. Paths must be relative and safe; absolute paths, traversal components, symbolic links, and unsupported source-file types are rejected. Runtime datasets belong in declared file inputs, not in hardcoded host paths.

7. Parameters and generated GUI controls

Each parameter has a stable `name`, a user-facing `label`, and a `type`. A helpful `help` description should explain units and assumptions. The program reads values by name from `inputs`; changing a label does not change that programmatic name.

Type	Intended input	Typical use
`integer`	A whole number	Samples, iterations, circuit size
`number`	A finite real number	Time step, angle, coupling constant
`boolean`	True or false	Enable a calculation branch
`string`	Text	A descriptive identifier or bounded text setting
`choice`	One declared choice	Select a supported method or model
`json`	Finite structured JSON	A matrix, coefficient list, or nested configuration
`file`	An explicitly uploaded file	CSV data, captured ephemeris, or a numerical array

7.1 Defaults, bounds, and requirements

Use `default` for a sensible ordinary value and `required` to state whether an input must be supplied. Use `minimum` and `maximum` on numeric values when the method has a defined domain or operational bound. Declare the permitted `choices` for a choice parameter.

The GUI helps users enter values, but the server checks the contract again. A user cannot bypass a maximum simply by modifying the browser request. Boolean values are not accepted as integers just because Python normally treats `bool` as an integer subclass.

JSON parameters are not a mechanism for arbitrary code execution or a substitute for engine-specific validation. For structured inputs, validate dimensions, value relationships, normalization, and physical assumptions in your program. Raise a useful exception when an input is invalid. For example, a density matrix needs checks beyond "this is valid JSON."

7.2 Names and units

Prefer names such as `duration_s`, `distance_m`, `angle_rad`, and `temperature_k` when units matter. Use labels such as "Angle (rad)" and help text that identifies the convention. Convert units explicitly in the calculation.

Use a meaningful input definition to expose scientific choices rather than internal paths or credentials. A `backend` choice should select only methods the workflow actually implements in its available runtime. A text control does not make an arbitrary executable or cloud account available.

7.3 File parameters

Declare a parameter of type `file` and select a file in the GUI for each applicable run. In a function workflow, access the captured file using the SDK context:

```
import csv


def run(inputs, context):
    path = context.input_file('measurements')
    with path.open(newline='', encoding='utf-8') as stream:
        rows = list(csv.DictReader(stream))
    if not rows:
        raise ValueError('The measurements CSV has no data rows')
    values = [float(row['value']) for row in rows]
    mean = sum(values) / len(values)
    return {'metrics': {'mean': mean}}
```

This is a program fragment; add the `measurements` parameter, the `mean` output, and representative checks to its bundle before importing it. Validate the CSV column names, finite values, size, and scientific units needed by your full calculation. Do not assume a file extension proves its contents.

Uploaded bytes are captured for the execution. The original desktop path is not an execution path, and the program does not gain access to neighboring files on the user's machine.

8. Program interfaces and the SDK

8.1 Function adapter

Use `adapter: "python"` and an entry point such as `main.py:run`. The callable receives two arguments:

```
def run(inputs, context):
    x = inputs['x']
    return {'metrics': {'square': x * x}}
```

The returned value must match the structured result contract. Declare the `x` parameter, `square` output, and independent checks in its full bundle. Import-time code should be small and deterministic; put the calculation in `run` rather than in module-level side effects.

8.2 Script adapter

Use `adapter: "python_script"` and an entry point such as `main.py`. A script obtains the request and emits its calculated result through the SDK:

```
from pf_workflows.sdk import load_request, emit_result

inputs, context = load_request()
x = inputs['x']
emit_result({'metrics': {'square': x * x}})
```

The script must finish normally and call `emit_result`. Printing a dictionary to stdout does not provide a structured result. Stdout and stderr serve as logs. An interactive `input()` prompt or a GUI event loop is not appropriate inside this batch execution contract.

`load_request()` and `emit_result()` are execution APIs, not a command for bypassing the workflow service. They require the request environment established by the isolated executor.

8.3 Context reference

Member	Meaning
`context.output_dir`	Writable directory for this run's output artifacts
`context.core_source`	The materialized scientific source bound to this execution
`context.file_inputs`	Captured file-input paths keyed by parameter name
`context.input_file(name)`	Retrieve the captured path for a declared uploaded input
`context.output_path(relative_path)`	Create safe parent directories and obtain an artifact path beneath the output directory

These are paths inside the selected execution environment. Do not record them as portable paths for a later workstation. The source tree is read-only in production. Work in the output directory and return relative artifact paths.

8.4 Write a downloadable artifact

```
import csv


def run(inputs, context):
    x = list(range(inputs['samples']))
    y = [value * value for value in x]
    destination = context.output_path('tables/squares.csv')
    with destination.open('w', newline='', encoding='utf-8') as stream:
        writer = csv.writer(stream)
        writer.writerow(['x', 'square'])
        writer.writerows(zip(x, y))
    return {
        'metrics': {'last': y[-1]},
        'artifacts': [
            {'name': 'csv', 'path': 'tables/squares.csv'}
        ]
    }
```

Declare the corresponding outputs in the bundle. Artifact paths must identify actual regular files produced under the output directory. A reference to a host path, a symbolic link, a parent-directory traversal, or a missing file is rejected. Artifact collection and total retained evidence are bounded. Section 11.10 shows the same pattern for full quantum-state arrays; chapter 16 gives the exact inline JSON and artifact limits.

8.5 Develop modularly

Separate scientific calculation from file parsing and display preparation. A calculation function that accepts arrays and returns arrays is easier to test than one that also reads GUI state or writes application metadata. Keep `run(inputs, context)` as a thin adapter that validates scientific assumptions, calls the calculation, and serializes outputs.

Use explicit function arguments rather than hidden environment state. Record random seeds when a stochastic algorithm requires reproducible sampling. Prefer exact deterministic checks for basic behavior and statistically justified checks for sampled outcomes. Make errors identify the invalid input and the violated condition.

Convert library-specific objects to ordinary JSON-compatible values at the result boundary. For NumPy, use `.tolist()` for arrays and `float(...)` or `int(...)` for scalar values where appropriate. Complex arrays require an explicit representation, such as separate real and imaginary arrays; bare Python complex numbers are not JSON values. Conversion with `.tolist()` does not remove size, node-count, or depth limits. For large arrays, use the artifact pattern in section 11.10.

9. Structured results and presentation

Return a JSON-compatible dictionary with any of the five standard collections below. Omitted collections are normalized to empty dictionaries or arrays. Every actual named result must match one declared output, and every declared output must be returned with its declared kind. Returning extra undeclared results is an error.

Result collection	Contents	Declared output kind
`metrics`	Mapping of names to finite numbers or strings	`number` or `string`
`data`	Mapping of names to finite JSON values	`json`
`tables`	Array of named tables	`table`
`series`	Array of named numeric x/y series	`series`
`artifacts`	Array of named relative file references	`file`

Names must be unique across all five collections. For example, `metrics.energy` and `data.energy` cannot both be returned.

9.1 A complete result shape

```
{
  "metrics": {"mean": 2.0, "method": "arithmetic mean"},
  "data": {"settings": {"samples": 3}},
  "tables": [
    {
      "name": "samples",
      "columns": ["index", "value"],
      "rows": [[0, 1.0], [1, 2.0], [2, 3.0]]
    }
  ],
  "series": [
    {"name": "curve", "x": [0, 1, 2], "y": [1.0, 2.0, 3.0]}
  ],
  "artifacts": [
    {"name": "csv", "path": "samples.csv", "mime_type": "text/csv"}
  ]
}
```

This example requires six output declarations: `mean:number`, `method:string`, `settings:json`, `samples:table`, `curve:series`, and `csv:file`. The program must create `samples.csv` inside its output directory before returning.

9.2 Tables

A table has `name`, `columns`, and `rows`. Column names are distinct nonempty strings. Each row is an array with exactly as many cells as columns. The schema permits up to 128 columns and 10,000 rows, subject to the total result-size limit. For larger datasets, return a compact summary table and a bounded downloadable artifact.

The GUI presents a table through its own renderer. Table values are data; they cannot inject executable HTML into the page.

9.3 Series

A series has `name`, `x`, and `y`. Both arrays must have the same nonzero length, at most 10,000 points, and contain finite real numbers. For multiple curves, return multiple named series and declare each one.

The original run viewer includes a general x/y display. The shared classical reporting framework now adds automatic descriptive analysis, scientific state checks, matrix heatmaps, distributions, exploratory PCA, declared structure diagrams and HTML/PDF exports. Declare units, basis and dataset types when physical interpretation needs them. Private analyzers and renderers support specialized methods such as error bars, logarithmic axes or publication formats. Do not imply an uncertainty or physical meaning that was never supplied. See chapter 18 and R34_CLASSICAL_REPORTING_GUIDE.md.

9.4 Numerical serialization

`NaN`, positive infinity, negative infinity, complex values, NumPy objects, and arbitrary Python class instances are not accepted as ordinary result values. Handle undefined quantities explicitly. For example, return a JSON data object explaining that a ratio is undefined when its denominator is zero, rather than emitting infinity as a numerical metric.

The metric type distinguishes numbers from strings. A numeric-looking string such as `"0.5"` is still a string and does not satisfy a `number` output. Booleans belong in JSON data rather than numeric metrics. The complete structured result must also fit all three independent limits: 2 MiB of compact UTF-8 JSON, 100,000 visited value nodes, and depth 32 with the root at depth 0. See section 16.1 for exact counting and section 11.10 for a full-state artifact example.

10. Validation cases and scientific checks

Each bundle requires 1-16 checks. Each check contains a human-readable `name`, an `inputs` object, optional captured `files`, and 1-64 `expectations`. Inputs and file captures are validated with the same contract used for ordinary runs.

A check executes the submitted program with its stated inputs. The trusted service then evaluates each expectation against the structured result. Paths are JSON pointers into that result, such as `/metrics/last`, `/data/settings/samples`, or `/tables/0/rows/0/1`.

10.1 Expectation operators

Operator	Meaning
`eq`	Equal JSON value, using the contract's serialized value comparison
`ne`	Not equal under that comparison
`lt` / `le`	Numeric less than / less than or equal
`gt` / `ge`	Numeric greater than / greater than or equal
`close`	Numeric agreement within absolute plus relative tolerance
`contains`	Membership in a list, dictionary keys, or substring in text
`length`	Exact length of an array, dictionary, or string

For `close`, the service checks:

```
abs(actual - expected) <= atol + rtol * abs(expected)
```

The defaults are `atol = 1e-8` and `rtol = 1e-5`. Specify tolerances explicitly for scientific checks so their meaning is reviewable. Only `close` accepts `atol` and `rtol`; both must be nonnegative finite numbers. The expected value for numerical comparison operators must be a finite number.

`eq` preserves JSON-type distinctions, including an integer representation versus a floating-point representation. Use `close` when comparing numerical results that can reasonably serialize as either integer or float. A missing pointer or incompatible actual type fails the expectation. Large actual or expected JSON values are summarized in the validation display with their size and hash; the complete values remain in the retained bundle or execution report.

10.2 Choose checks that test the calculation

For a numerical method, include a known analytic case, a meaningful nontrivial case, and an invariant where appropriate. For example, a quantum state calculation can check normalization and a known limiting state. A regression test that only asserts "the program returned a dictionary" provides little scientific evidence.

Input-schema bounds prevent invalid entry but do not establish the numerical method's accuracy across the entire range. Test boundary behavior deliberately. A parameter sweep may require convergence checks or comparison with an independent implementation. Preserve the distinction between verifying implementation behavior and validating a model against measurements.

Avoid exact sampled-frequency expectations for stochastic calculations. Use a controlled seed or a statistically justified tolerance, and document what the seed and tolerance establish. The supplied statevector example computes probabilities without finite-shot sampling, making exact limiting checks easier to interpret.

10.3 Validation with file inputs

A validation case can carry a small base64-encoded input file under the same parameter-keyed `files` structure used by run requests. For example:

```
{
  "name": "Three reference values",
  "inputs": {},
  "files": {
    "measurements": {
      "name": "measurements.csv",
      "content_base64": "dmFsdWUKMQoyCjMK"
    }
  },
  "expectations": [
    {"output": "/metrics/mean", "operator": "close", "value": 2,
     "atol": 1e-12, "rtol": 0}
  ]
}
```

The encoded text is `value` followed by the values 1, 2, and 3 on separate lines. It is a deliberately small reference fixture for the CSV-mean program fragment. Include this case in a full bundle with a required `measurements:file` parameter and a `mean:number` output.

10.4 Revalidation after changes

A changed program, input/output schema, declared dependency, expectation, scientific source base, or runtime identity must not inherit a stale pass. Save the applicable new proposal and validate it. The application records source and runtime identities alongside results so a later reader can determine what actually ran.

11. Supplied scientific examples

The `examples_workflows` directory contains the six original complete importable JSON bundles. A seventh, the Floquet study accompanying this revised guide, is in `documentation/workflow_examples/`. Use these files directly for a first run rather than reconstructing a bundle from a code fragment in this guide. They illustrate the same generic workflow interface with different calculations.

Bundle	Calculation and interface features
`stdlib_oscillator.json`	Modular analytic oscillator, numeric controls, trajectory chart, CSV artifact
`stdlib_script_statistics.json`	Script adapter, structured JSON input, mean and sample standard deviation
`csv_file_statistics.json`	Captured CSV input, statistics, and a returned data table
`numpy_scipy_linear_system.json`	NumPy arrays, SciPy linear solve/eigenvalues, residual and table
`qiskit_entangled_state.json`	Two-qubit statevector probabilities and analytic limiting checks
`qiskit_scipy_parameter_fit.json`	Qiskit probability evaluation, SciPy root finding, NumPy entropy calculation
`qiskit_floquet_spin_chain.json` (documentation examples)	Full noisy Floquet evolution, seven controls, six observables, curves, CSV, and complete state artifacts

The standard-library examples have an empty requirements list. The numerical examples declare exact pins: Qiskit `2.5.2`, NumPy `2.3.5`, and SciPy `1.18.1` as applicable. These are the bundle requirements, not a promise that every installed PF runtime already has those exact versions. Inspect the executor's dependency result. An operator can provision the matching runtime, or you can create and validate a deliberate new proposal for another compatible installed version.

11.1 Modular oscillator

Import `examples_workflows/stdlib_oscillator.json`. The entry point calls the separate private `oscillator.py` helper to evaluate an analytic cosine trajectory. Its controls expose amplitude, frequency, duration, and sample count.

With defaults, amplitude is 1, frequency is 1 cycle per second, duration is 1 second, and the returned final displacement is 1. The half-period validation case uses amplitude 2 and duration 0.5 seconds, producing final displacement -2. The chart and `oscillator.csv` contain the actual sampled analytic values.

This example illustrates modular source and artifacts. It is not a general differential-equation integrator or evidence of agreement with a physical measurement. To turn it into a numerical integration study, add the integration method, step-size controls, and convergence checks in a new proposal.

11.2 Script and CSV statistics

Import `stdlib_script_statistics.json` to use the script adapter. Its default JSON array is `[1, 2, 3]`; mean and sample standard deviation are 2 and 1. Changing the scale to 2 gives 4 and 2. The returned JSON data contains the scaled values.

Import `csv_file_statistics.json` to use a file control. Validation uses the embedded small fixture; for an ordinary run, explicitly select `examples_workflows/sample.csv`. That file contains a `value` column and three numerical rows. The ordinary run independently captures those bytes.

11.3 NumPy and SciPy linear system

Import `numpy_scipy_linear_system.json`. It solves the fixed system with matrix `[[2, 1], [1, 3]]` and the input vector `[b0, b1]`. At the defaults `[1, 2]`, the solution is `[0.2, 0.6]`. The residual is computed from the actual matrix-vector product, and the returned table contains the symmetric matrix's eigenvalues.

The GUI does not require a special SciPy page: it displays the declared `b0` and `b1` controls, metrics, and table. The independent residual calculation helps verify that the returned solution satisfies this particular system.

11.4 Qiskit entangled state

Import `qiskit_entangled_state.json`. The program creates two qubits, applies an RY rotation to qubit 0, and then applies CX from qubit 0 to qubit 1. It uses Qiskit's statevector calculation to obtain basis-state probabilities:

```
from qiskit import QuantumCircuit
from qiskit.quantum_info import Statevector


def run(inputs, context):
    circuit = QuantumCircuit(2)
    circuit.ry(inputs['theta'], 0)
    circuit.cx(0, 1)
    state = Statevector.from_instruction(circuit)
    probabilities = state.probabilities().tolist()
    return {
        'metrics': {
            'p00': probabilities[0],
            'p11': probabilities[3],
            'normalization': sum(probabilities)
        },
        'tables': [{
            'name': 'probabilities',
            'columns': ['basis', 'probability'],
            'rows': [
                [format(i, '02b'), value]
                for i, value in enumerate(probabilities)
            ]
        }]
    }
```

At `theta = pi/2`, the two nonzero probabilities are 0.5 and 0.5. At zero angle, `p00` is 1; at pi, `p11` is 1. The supplied checks test those limits and normalization. The statevector is calculated locally; this example does not submit hardware jobs, model measurement noise, or generate finite-shot counts.

IBM's Statevector API reference documents the circuit-to-statevector and probability methods used here. Consult the documentation for the version deployed in your runtime when adapting the example.

11.5 Combine Qiskit, SciPy, and NumPy

Import `qiskit_scipy_parameter_fit.json`. Its input is a target probability between 0 and 1. Qiskit evaluates a one-qubit RY circuit; SciPy finds an angle that achieves the target; NumPy computes the binary entropy from the calculated probabilities.

For target probability 0.25, the angle is approximately 1.0471975512 radians. For target 0.5, the angle is pi/2 and entropy is 1 bit. The program returns the achieved probability and residual so the fit can be checked directly.

This is a concrete multi-library workflow in one isolated Python runtime. It does not orchestrate arbitrary remote services or provision mutually incompatible environments. If two engines need different runtimes or a remote service, an operator must design and qualify the corresponding execution support.

11.6 Adapt an example

Export or copy its JSON and make a deliberate new proposal. Keep the known checks while adding the new behavior and independently justified checks. Update the parameters and outputs together with the program so the GUI and execution contract remain synchronized.

A useful progression is: first reproduce the supplied case, then change an ordinary input, then add a result, then add a method choice. Each code/schema change creates a new version. This makes failures easier to localize than changing the calculation, runtime, data format, and display simultaneously.

11.7 Worked example: specify the Floquet spin chain

This example begins with a mathematical specification and uses the Polynomial starter only for integration structure. The complete importable bundle is `documentation/workflow_examples/qiskit_floquet_spin_chain.json`. The mathematical specification and usage notes are `documentation/workflow_examples/floquet_spin_chain/SPECIFICATION.md` and `README.md`. The complete Python source is embedded in the bundle's `files` entry with path `main.py`; no separate Python file is needed for import. You can inspect that source in the proposal editor or print it without executing it:

```
import json
from pathlib import Path

path = Path('documentation/workflow_examples/qiskit_floquet_spin_chain.json')
bundle = json.loads(path.read_text(encoding='utf-8'))
program = next(item['content'] for item in bundle['files']
               if item['path'] == 'main.py')
print(program)
```

The model is deterministic local density-matrix evolution with a separate noise-free statevector. It requires Qiskit 2.5.2 and NumPy 2.3.5 in the configured executor. It does not use finite-shot samples, a polynomial surrogate, a remote backend, or quantum hardware.

Control	Default	Supported values and meaning
`n`	6	Integer 2-8; Hilbert-space dimension is `2**n`
`cycles`	8	Integer 0-16; include the initial state at cycle 0
`J`	0.34	Radians, 0 through pi; ZZ interaction parameter
`h`	0.71	Radians, 0 through pi; transverse rotation parameter
`p`	0.025	0 through 0.5; independent phase-flip probability
`boundary`	`ring`	`open` or `ring`
`initial_state`	`neel`	`neel` or `zero`

Computational-basis integer indexing is little-endian: q0 is the least significant bit. The Neel product state has q0=0, q1=1, q2=0, and so on; `zero` means every qubit starts at zero. An open chain has edges (0,1), (1,2), through (n-2,n-1). A ring adds (n-1,0) only when n is greater than 2. Each undirected edge is applied once, so the two-qubit ring does not double the interaction.

Each complete cycle performs these operations in order:

1. Apply `RZZ(2*J) = exp(-i*J*Zi*Zj)` to every defined edge.
2. Apply `RX(2*h) = exp(-i*h*Xi)` to every qubit, after all ZZ gates.
3. Apply `rho -> (1-p)*rho + p*Zi*rho*Zi` once on each qubit, independently.

The ideal statevector follows the identical unitary gates without noise. Record all observables initially and after every complete cycle including noise. The six-qubit default therefore has nine trajectory rows. These conventions specify the implemented model; they should not be inferred from a circuit title alone.

Final metric	Definition
`fidelity`	Real part of the noisy density-matrix expectation in the current ideal statevector
`purity`	Real part of `trace(rho @ rho)`
`magnetization`	Mean expectation of Z over all n qubits
`zz_correlation`	Mean expectation of ZiZj over the defined undirected edges
`half_entropy_bits`	Von Neumann entropy of subsystem A, with logarithms base 2
`mutual_information_bits`	`S(A) + S(B) - S(AB)` for the noisy state

Here fidelity is explicitly `Re(<psi_ideal|rho|psi_ideal>)`. It is not the return probability to the initial state or a different Loschmidt-echo definition. Subsystem A contains the lowest `floor(n/2)` qubits; B contains the rest. Entropies use reduced density matrices and `S = -sum(lambda*log2(lambda))`, taking the zero-eigenvalue contribution as zero. The implementation rejects materially negative density-matrix eigenvalues and clips eigenvalue roundoff to the physical range when evaluating entropy.

The result includes these six metrics, six cycle series, an observables table, `observables.csv`, settings/conventions/dependency data, and complete final states as real and imaginary arrays in `final_states.json`. For up to seven qubits, state arrays are also returned inline. At eight qubits, the corresponding JSON data fields instead hold artifact locations, as explained in section 11.10.

11.8 A request for specification-based AI authoring

To try fresh authoring, save the Polynomial starter as described in section 5.2 and open **Create with AI**. Supply the specification from section 11.7 or attach the accompanying `SPECIFICATION.md`. Replace the bracketed starter identifier below with its actual name or ID. For a fresh-generation comparison, supply the specification and starter, without supplying the finished Floquet source or bundle.

> Read my saved Polynomial sweep workflow [name or ID] as an integration template. Create a separate new private workflow called "Noisy Floquet Ising chain" from the attached mathematical specification. Generate the full Python calculation; replace the polynomial calculation entirely. Use schema_version 1, adapter python, entrypoint main.py:run, and complete source files. Declare the installed Qiskit 2.5.2 and NumPy 2.3.5 requirements.

> Implement the specified 2-8 qubits and 0-16 cycles, with six qubits and eight cycles by default. Preserve the exact RZZ-then-RX-then-phase-flip order, little-endian indexing, Neel/all-zero states, and open/ring edge convention. Calculate the noisy density matrix and a separate noiseless ideal statevector with Qiskit. Use the specification's fidelity and base-2 entropy definitions. Ask about any unresolved scientific assumption before choosing it.
>
> Expose all seven controls with the specified bounds. Return the six final metrics, their cycle curves, a complete observables table, and an actual CSV artifact. Save complete unrounded real/imaginary final states to final_states.json. Use compact artifact pointers at eight qubits so the structured result fits 100,000 value nodes, depth 32, and 2 MiB. Declare every returned output, including file artifacts.
>
> Include independent analytic checks for the zero-cycle Neel state, noiseless fidelity/purity, and the stationary all-zero state with h=0. State expected values and tolerances explicitly. Use actual scientific calculations without fabricated reference data or hardware claims. Explain the source and assumptions for my review, submit the complete proposal, and leave validation, activation, and running to me.

This is a task specification, not a guarantee of a provider's response quality or capacity. Review truncation, dependency assumptions, and complete tool arguments as described in section 5.4. If the model produces text rather than a saved proposal, obtain a complete valid bundle before importing it. Do not activate a draft merely because its description repeats the requested equations.

11.9 Review, import, validate, activate, and run the example

The supplied bundle allows you to run the completed example without another model call. A newly generated proposal requires the same review and lifecycle steps:

1. Confirm that the isolated executor has the exact declared dependencies. Ordinary workstation imports do not establish what is installed inside that executor.
2. For the supplied example, use **My workflows > Import bundle** and select `documentation/workflow_examples/qiskit_floquet_spin_chain.json`. For a newly generated example, use **Open My workflows** to inspect the saved draft.
3. Review the full source, seven input definitions, declared outputs, and three analytic checks. Confirm the edge handling, factor-of-two gate angles, noise order, state indexing, and observable definitions against section 11.7. For a manual import, select **Save proposal** after review.
4. Select **Validate** and inspect every case. Zero-cycle Neel values are fidelity 1, purity 1, mean Z 0, mean ZZ -1, and half entropy and mutual information 0 for the six-qubit ring. The noiseless case must retain fidelity and purity 1. The h=0 all-zero case remains stationary as a density matrix under the ZZ phases and phase flips. The supplied numerical expectations use `atol=2e-10`, `rtol=0`.
5. After successful validation, select **Activate** explicitly. Validation alone leaves the proposal inactive. A changed source file, dependency, schema, or expectation requires a new proposal and fresh validation.
6. Leave the seven default inputs and select **Run workflow**. Inspect the newly completed run, all six curves, its nine-row table, and downloadable CSV and full-state JSON.
7. Set `n=8` and `cycles=16` for the upper-bound case, then run again. Its table has 17 rows. Download `final_states.json` to inspect the complete 256 by 256 complex density matrix and 256-component ideal statevector; inline fields contain pointers for this case.
8. Export the reviewed bundle and retain run inputs, artifacts, and identities with your interpretation. Ordinary accepted input changes use the activated program; they do not require fresh AI authoring.

For the documented six-qubit defaults, the internally measured final values below provide a comparison with that recorded execution. They are calculated outputs, not constants to substitute for running the program, and are not experimental measurements.

Metric	Recorded default result
Fidelity	0.328747734733427
Purity	0.125175877675581
Mean Z	Approximately 0, within floating-point roundoff
Mean edge ZZ	-0.271016509391629
Half entropy (bits)	2.71727841704199
Mutual information (bits)	0.946440931588551

A successful three-case activation check is narrower than the independent test suite described in section 11.11. For research changes, compare complete trajectories and states against an independent method, test limiting and boundary cases, and retain the evidence. Do not relax a tolerance just to hide a disagreement.

11.10 Full quantum states as downloadable artifacts

At eight qubits the density matrix has dimension 256 by 256. Separate real and imaginary arrays contain `2 * 256 * 256 = 131072` scalar numbers before counting array containers, the ideal statevector, or any other outputs. They therefore exceed the 100,000-value-node limit even if a particular compact JSON encoding fits within 2 MiB. Rounding the numbers or removing whitespace cannot fix the node count.

Write the complete arrays to an output artifact and return compact metadata instead. This changes serialization only: it does not change the density-matrix evolution, reduce numerical precision, discard state entries, increase a quota, or require a shared-core modification. The companion program writes this file for every size and returns pointers instead of inline state arrays when `n=8`.

The following helper illustrates the pattern. Its arguments are the calculated NumPy density-matrix and ideal-statevector arrays, the qubit count, and the workflow context. It is a serialization fragment, not a replacement scientific program:

```
import json


def serialize_states(rho_array, ideal_array, n, context):
    states = {
        'final_density_matrix': {
            'real': rho_array.real.tolist(),
            'imag': rho_array.imag.tolist()
        },
        'final_ideal_statevector': {
            'real': ideal_array.real.tolist(),
            'imag': ideal_array.imag.tolist()
        }
    }
    path = context.output_path('final_states.json')
    with path.open('w', encoding='utf-8') as stream:
        json.dump(states, stream, allow_nan=False, separators=(',', ':'))
    data = states if n <= 7 else {
        name: {
            'artifact_path': 'final_states.json',
```

```
            'json_pointer': '/' + name
        }
        for name in states
    }
    return {
        'data': data,
        'artifacts': [{
            'name': 'final_states_json',
            'path': 'final_states.json',
            'mime_type': 'application/json'
        }]
    }
```

Declare these three outputs when using that fragment, in addition to every metric, table, series, and other field returned by the full program:

```
[
  {"name": "final_density_matrix", "kind": "json",
   "label": "Final noisy density matrix"},
  {"name": "final_ideal_statevector", "kind": "json",
   "label": "Final ideal statevector"},
  {"name": "final_states_json", "kind": "file",
   "label": "Full final states JSON"}
]
```

Merge the fragment's `data` and `artifacts` into the full result; do not replace the other declared outputs. The complete packaged bundle also returns `state_artifact` metadata describing shape and storage. A JSON output may contain arrays at one size and pointer objects at another, but the program must document that convention for consumers.

The `artifact_path` and `json_pointer` strings are this workflow's metadata convention. They do not cause the generic GUI or expectation evaluator to load a file automatically. Download the retained `final_states_json` artifact and select the named object to recover its real and imaginary arrays. A check against an inline pointer cannot establish properties of the file's matrix. The independent verification in section 11.11 read the actual retained artifact to compare those arrays.

The JSON writer uses the calculated floating-point values without deliberate rounding and rejects NaN/infinity. It preserves the available numerical precision; it does not create exact real-number arithmetic. Artifact files remain subject to the 32 MiB per-file/total-selected bounds, safe-path and integrity checks, and owner/storage evidence quotas. The inline result still must satisfy all limits in chapter 16.

11.11 What the internal Floquet test establishes

The 2026-09-25 internal test used a fresh Codex agent to generate a complete program from the actual Polynomial starter and a detailed specification. It also inspected r34 source contracts. The first six-qubit/eight-cycle program passed without a numerical repair. A deliberate eight-qubit stress run exposed the inline-result limit; a revised private proposal moved large state output to an artifact, then passed fresh validation and explicit activation. The numerical evolution and shared core were unchanged.

The actual r34 assistant tool broker and workflow service accepted the proposal and enforced lifecycle controls. Numerical execution used the designated developer executor in a separate Python process. A separately authored NumPy reference used dense linear algebra and a Hamming-distance dephasing formula, without Qiskit or access to the generated implementation.

The final workflow passed 16 cases, covering 131 trajectory points and 368 numerical/output comparisons. Cases included the default study, zero cycles, zero noise, zero rotations/couplings, maximal supported phase flips, open/ring boundaries, the two-qubit edge convention, both initial states, and eight qubits over 16 cycles. Comparisons included trajectories, full final states, output

consistency, trace, Hermiticity, positive semidefiniteness within roundoff tolerance, normalization, and retained-artifact hashes. Observable and full-state comparison tolerances were 1e-9 and 1e-10 respectively.

No live PF AI-provider connection or production Docker executor was configured for that test. It therefore demonstrates Codex-authored code, actual r34 proposal/lifecycle behavior, and real numerical execution in the developer test environment. It does not establish live-provider authoring reliability, production container isolation, success on arbitrary research problems, or sufficiency of the PDF alone. The user review, execution, comparison, and repair coordination occurred outside the r34 model's workflow tools.

12. HTTP API reference

The common prefix is `/api/assistant/workflows`. Standalone and SaaS use the same workflow payloads. Their authentication differs.

- Standalone uses the local launch session and `X-PF-Token` header, with the existing loopback/origin checks. Open the GUI through its launcher.
- SaaS uses the authenticated session cookie and requires `X-CSRF-Token` on writes. Supply the project as `X-PF-Project` or the `project_id` query parameter. Conflicting project selections are rejected.

Authentication and project access must already be established through the normal application. Do not place session credentials in a workflow bundle or in its scientific program.

12.1 Endpoint table

Here `{id}` is a workflow ID and `{run_id}` is a run ID. IDs come from service responses.

Method and relative path	Request / response
`GET /`	Returns `{"workflows": [...]}`, including archived metadata for client filtering
`POST /`	Accepts a bundle proposal; returns `{"workflow": ...}`
`POST /authoring-sessions`	Accepts optional `title`; returns a new workflow-authoring `session`
`GET /{id}`	Returns `{"workflow": ...}`
`POST /{id}/validate`	Accepts `expected_revision`; returns updated workflow
`POST /{id}/activate`	Accepts `expected_revision`; returns updated workflow
`POST /{id}/deactivate`	Accepts `expected_revision`; returns updated workflow
`POST /{id}/delete`	Compatibility alias for archive; returns updated workflow
`POST /{id}/archive`	Archives the workflow; accepts `expected_revision`
`POST /{id}/restore`	Restores an archived workflow inactive; accepts `expected_revision`
`GET /{id}/deletion-preview`	Returns the permanent-deletion scope
`POST /{id}/purge`	Permanently deletes an archived workflow; accepts `expected_revision` and returns cleanup operation
`POST /{id}/run`	Accepts `inputs`, optional `files`, and `expected_revision`; returns `{"run": ...}`
`GET /{id}/runs`	Returns `{"runs": [...]}`
`GET /runs/{run_id}`	Returns `{"run": ...}`
`GET /runs/{run_id}/artifacts?path=...`	Downloads one retained artifact by its recorded path

Method and relative path	Request / response
`GET /{id}/export`	Downloads the current reusable workflow JSON
`GET /{id}/export?version=N`	Downloads retained immutable version N

A create request contains `bundle`. To edit an existing workflow, include `workflow_id` and its current `expected_revision`. `source_session_id` optionally binds a proposal to an existing permitted assistant session. The service determines ownership and verifies the scientific source context.

12.2 Request examples

New proposal:

```
{
  "bundle": {
    "schema_version": 1,
    "name": "A complete bundle goes here"
  }
}
```

The abbreviated object above illustrates the envelope only; replace `bundle` with a complete valid bundle such as the polynomial example. It is not itself an importable workflow.

Validation or activation:

```
{"expected_revision": 3}
```

Ordinary run:

```
{
  "expected_revision": 5,
  "inputs": {"coefficient": 3, "samples": 5}
}
```

Run with a captured file:

```
{
  "expected_revision": 5,
  "inputs": {},
  "files": {
    "dataset": {
      "name": "sample.csv",
      "content_base64": "dmFsdWUKMQoyCjMK"
    }
  }
}
```

Revision values here are illustrative. Always obtain the current revision from the service. A file parameter's optional value in `inputs` must agree with its uploaded basename; it cannot be a host path. `files` is keyed by declared parameter name, not by an arbitrary filename.

12.3 Small authenticated Python client

The following helper uses only Python's standard library. Run it in a development client with an already established session. Configure either `PF_LAUNCH_TOKEN` for standalone or `PF_SESSION_COOKIE`, `PF_CSRF_TOKEN`, and `PF_PROJECT_ID` for SaaS. `PF_SESSION_COOKIE` contains the authenticated Cookie header value. These client variables are not workflow parameters or new PF server settings.

```
import json
import os
from urllib.request import Request, urlopen
```

```
base = os.environ['PF_BASE_URL'].rstrip('/')
prefix = '/api/assistant/workflows'


def workflow_request(path='', body=None):
    headers = {'Accept': 'application/json'}
    if os.environ.get('PF_LAUNCH_TOKEN'):
        headers['X-PF-Token'] = os.environ['PF_LAUNCH_TOKEN']
    else:
        headers['Cookie'] = os.environ['PF_SESSION_COOKIE']
        headers['X-PF-Project'] = os.environ['PF_PROJECT_ID']
        if body is not None:
            headers['X-CSRF-Token'] = os.environ['PF_CSRF_TOKEN']
    data = None
    if body is not None:
        headers['Content-Type'] = 'application/json'
        data = json.dumps(body, allow_nan=False).encode('utf-8')
    request = Request(base + prefix + path, data=data, headers=headers)
    with urlopen(request, timeout=3600) as response:
        return json.load(response)
```

Use `workflow_request()` to list workflows. To submit a saved JSON file:

```
from pathlib import Path

bundle = json.loads(Path('polynomial.workflow.json').read_text('utf-8'))
created = workflow_request(body={'bundle': bundle})['workflow']
```

Use returned IDs and the current revision for later actions. Validation can be long-running; align client and reverse-proxy timeouts with the configured operation bound. A client timeout is not proof that the server failed to commit an action. Fetch current state and run history before deciding whether to submit another run.

The artifact endpoint returns bytes, and the export endpoint returns a JSON download; a production client should handle those responses according to their content rather than assuming every route uses the same envelope.

12.4 AI tool interface

The model-facing tools use the same controlled service through bounded callbacks:

Tool	Purpose
`propose_workflow`	Submit a JSON-encoded bundle proposal
`list_workflows`	Discover workflows in the permitted owner context
`read_workflow`	Read a bounded character range from a workflow record

The `propose_workflow` tool's `bundle` argument is a JSON string to support the provider tool protocol. The manual HTTP proposal endpoint instead accepts a JSON object. `read_workflow` uses `workflow_id`, `offset`, and `limit`; a read is bounded to 40,000 characters. None of these model tools activates a workflow.

13. Runtime provisioning and native integrations

13.1 Declare installed distributions

A requirement contains `distribution` and `version`, for example:

```
{"distribution": "qiskit", "version": "2.5.2"}
```

The version is an exact installed distribution version. Ranges such as `>=2`, pip command lines, package URLs, extras, and installation instructions do not belong in this field. The launcher checks distribution metadata inside the actual executor before importing the private program. A missing or mismatched distribution produces an unavailable execution, not an automatic installation.

The Python import name and distribution name can differ. Use the distribution name reported by the installed package metadata. See Python's importlib.metadata reference for the metadata API used by the runtime.

Dependencies not listed in the bundle may still be imported if installed, but declaring the dependencies important to the result makes requirements and compatibility explicit. Pinning listed Python distributions alone does not fully describe native libraries, operating-system packages, devices, or scientific assumptions. Retain the container identity and relevant deployment records.

13.2 Operator responsibilities

The operator selects and builds the execution image, installs licensed/native prerequisites where applicable, reviews dependency compatibility, and sets resource limits. The browser and AI do not choose an interpreter or installation command.

Setting	Default	Meaning
`PF_AI_DOCKER_IMAGE`	Must be configured	Isolated scientific dependency image
`PF_AI_EXECUTION_TIMEOUT_SECONDS`	300	Wall-time bound per scientific execution
`PF_AI_MEMORY_MIB`	2048	Container memory allowance
`PF_AI_CPUS`	1.0	CPU allowance
`PF_AI_PIDS`	128	Process-count allowance
`PF_AI_MAX_OUTPUT_BYTES`	268435456	Overall execution output allowance
`PF_AI_DOCKER_RUNTIME`	Docker default	Optional administrator-installed container runtime

SaaS workflow requests use `PF_SAAS_WORKFLOW_MAX_REQUEST_BYTES` (24 MiB by default); other application JSON routes retain their existing limits. Base64 encoding expands binary uploads, so a lower request allowance can become the effective input limit.

The service clamps executor administrator values to its supported ranges. These are not user-supplied workflow inputs. A validation operation can execute several cases sequentially, so its total duration can exceed one case's execution timeout. Configure API/proxy timeouts and organization admission limits accordingly.

Changing image or runtime components invalidates the old activation binding. Revalidate affected workflows after a deliberate environment update. A missing dependency should be resolved by provisioning the runtime or changing and revalidating the proposal, not by enabling host execution.

13.3 Native and remote capability boundaries

The shipped Python adapters can call installed scientific libraries and, where available, native tools inside the same sandbox. An integrated API that ultimately needs a native executable is usable only when that executable and its dependencies are actually provided inside the image and can operate within the execution limits.

The shipped executor has no network and does not inherit service credentials or arbitrary host-device access. Consequently, internet APIs, IBM hardware submission, network license servers, live downloads, and directly attached instruments do not become available simply because a Python client package is installed. Import captured data for offline work. A separate operator-designed and qualified adapter/backend is needed when the scientific operation fundamentally requires another execution model.

Windows desktop availability of a proprietary program also does not imply its availability in the Linux container used by the private executor. Follow the corresponding PF integration guide for that engine and check the actual execution environment.

14. Architecture and adapter development

The implementation separates contract validation, state management, execution, transport, and presentation. This separation keeps engine-specific program logic out of GUI and authorization code.

Module	Responsibility
`pf_workflows/contracts.py`	Strict bundle, parameter, result, path, and expectation contracts
`pf_workflows/service.py`	Owner scope, immutable versions, lifecycle, source bindings, persistence, and integrity
`pf_workflows/executor.py`	Isolated staging, execution orchestration, runtime identity, and bounded artifact collection
`pf_workflows/runtime.py`	Launcher running inside the selected execution environment
`pf_workflows/adapters.py`	Maintainer-controlled execution adapter registry
`pf_workflows/sdk.py`	Small function/script context and result interface
`pf_workflows/http.py`	Shared route and payload dispatch contract
`pf_gui/assistant.py`	Standalone authentication and assistant transport integration
`pf_saas/assistant.py`	Hosted project authorization, admission, and transport integration
`pf_gui/static/private-workflows.js`	Generic workflow lifecycle, controls, and results interface

14.1 Adding another Python library

A new installed Python scientific library usually requires no framework adapter change. Write its calculation in a private `python` or `python_script` bundle, declare its exact required distribution version, expose inputs, normalize outputs, and add justified checks. Provision its dependencies in the executor and validate the bundle there.

This is the common path for API-agnostic reuse. Avoid adding a new global dropdown and bespoke GUI page for each private experiment when the existing schema can describe its controls and results.

14.2 Adding another execution model

An adapter is trusted application code, not a private bundle plugin. The current registry exposes `python` and `python_script`. An operator cannot add a new runtime merely by naming it in JSON; unsupported adapter names are rejected by the schema.

A maintainer adding an execution model must update the registered adapter, schema allowlist, runtime identity, staging/collection rules where needed, and tests. Preserve the common inputs and outputs so the existing UI can remain generic. The adapter's execution method follows the current form:

```
def execute(self, bundle, program_root, inputs, context):
    # Runs only inside the designated execution environment.
    # Returns the common result, or emits it for a script adapter.
    ...
```

The production control process must never import a submitted program to inspect its function signature or discover its GUI fields. Discovery is declarative. Any new backend must retain the source, runtime, owner, revision, input, output, and permission checks at the same boundaries.

14.3 Development and review practices

Use narrow modules and explicit data contracts. Validate payloads before costly snapshots or execution reservations. Treat user Python as untrusted execution content and renderer inputs as data. Preserve exact revision checking for mutating actions and verify retained artifacts before returning them.

Test meaningful behavior: a changed input changes the calculated result; another owner cannot retrieve it; a viewer or revoked project cannot perform forbidden operations; stale validation cannot activate a new runtime; missing dependencies fail visibly; malformed output and unsafe file references are rejected; reopening the application retains the workflow and current settings.

The existing developer-only executor constructor is for authored fixtures. Evidence produced with it must be labelled as developer fixture execution. It is not a substitute for container-isolation, deployed-host, native Windows, live-provider, or hardware testing.

14.4 Extending the output vocabulary

If a future engine needs a new built-in display kind, add a strict result contract and a generic safe renderer. Do not accept arbitrary model-authored HTML or JavaScript as a shortcut. A bounded file artifact is appropriate when an established external viewer is needed and a generic renderer would distort the scientific content.

15. Privacy, project access, and retained evidence

15.1 Owner and project boundaries

Standalone workflows belong to the local assistant owner context. SaaS workflows are private to the authenticated user within the selected organization/project context. A caller cannot supply a replacement owner field. Existing B2 project restrictions and permission checks still apply to discovery, mutation, execution, and retrieval.

After access is revoked, a stale browser page is not authorization to retrieve or run a workflow. The server checks the current permission context. Changing the selected project changes the workflow context; a similarly named workflow in another project is a different record.

15.2 Source and runtime integrity

A retained workflow version records its bundle identity and scientific base. Validation and run records identify the source and runtime used, plus inputs, captured files, execution status, and retained artifact hashes. The service checks these bindings before activation and execution, and verifies retained bytes when read.

A hash identifies bytes; it does not prove that a model is scientifically valid. Its purpose is to prevent a later artifact from being silently substituted for the one reviewed or executed.

Private workflow programs can inspect the permitted materialized scientific source inside their execution context, but cannot write the installed shared core through this interface. Authentication, provider controls, registry policy, and other protected application behavior remain maintainer-owned.

15.3 BYOK and research-data policy

Personal AI keys configure the assistant provider under existing r34 BYOK rules. They are not execution credentials for arbitrary scientific services and are not passed into private workflow programs. Organization/project AI policy and consent controls still apply when the assistant proposes a workflow or reads permitted source for a response.

Manual import and ordinary repeated workflow runs do not need the assistant to contact an AI provider. This allows a previously activated workflow to remain useful without paying for repeated GUI generation or model calls.

15.4 BYOS and operational persistence

Workflow bundles and retained execution evidence use the existing assistant evidence storage integration. When BYOS is configured, its verification, availability, and recovery behavior applies to these artifacts as well. PF retains trusted metadata and permissions; moving data to user storage does not move the authorization decision into the file contents.

If user storage is unavailable or retained bytes do not match their recorded identity, the application reports the failure rather than substituting unverified data. Keep the storage connector available for retrieval and operation. Consult `PF_Simulation_r34_BYOS_Guide.md` for supported storage modes, connector setup, and recovery.

Back up the metadata database and private state together. Restore compatible snapshots together. A code package upgrade alone does not carry user state, and a bundle export alone does not contain all run evidence.

16. Limits and troubleshooting

Limits bound input parsing, generated controls, execution, and retained data. An operator or hosting plan can impose lower allowances. Do not confuse a schema limit with the smaller effective capacity of an HTTP request or browser upload.

16.1 Contract and evidence limits

Item	Current bound
Bundle JSON	2 MiB
Private Python files	1-32 files, at most 256 KiB each
Declared distributions	At most 32 exact requirements
Parameters	At most 64
Choice values	1-128 scalar values per choice parameter
Declared outputs	1-64
Validation cases	1-16
Expectations	1-64 per case
File input, core service limit	8 MiB per file; 16 MiB total
GUI file upload	8 MiB per file; 16 MiB total
Workflow HTTP JSON request	24 MiB by default, including base64 and all other fields
Structured result bytes	2 MiB of compact UTF-8 JSON
Finite JSON value nodes	At most 100,000 visited values, including root and containers
Finite JSON depth	At most 32, with the root at depth 0
Table	At most 128 columns and 10,000 rows
Series	1-10,000 equally sized numeric x/y points
Artifact	At most 32 MiB per file and 32 MiB total selected artifacts
Workflows	At most 64 per owner, including archived records
Versions	At most 128 per workflow
Retained workflow runs	At most 512 per owner, including validation executions
Retained workflow evidence	At most 256 MiB per owner, also subject to shared assistant/storage quotas

The `finite_json` check counts the root value once and recursively counts every object, array, and scalar value. Object keys are not counted as separate nodes, but do contribute encoded bytes. Each descent

to an object value or array element increases depth by one. A root at depth 0 may have descendants through depth 32; depth 33 is rejected. The count must not exceed 100,000 for the value being checked. Finite-JSON checks apply at contract boundaries, including bundles and structured results; passing the byte bound alone is insufficient.

For the result byte check, encoding uses `json.dumps(..., allow_nan=False, ensure_ascii=False, separators=(",", ":"))` followed by UTF-8 encoding, with a maximum of 2,097,152 bytes (2 MiB). A worker can also reject its serialized result at the execution boundary. Reducing nesting or shortening strings fixes only the applicable bound; it does not raise the others. Use the full-state artifact pattern in section 11.10 when a dataset does not belong inline.

Names used for parameters and outputs begin with a letter and contain only letters, digits, and underscores, up to 64 characters. Human-readable names and labels can contain ordinary spaces. File paths use safe relative components; uploaded filenames must be basenames.

For large raw datasets, return compact results and bounded artifacts, or use an appropriate existing PF scientific workflow. The private workflow feature is not an unlimited streaming-data service. Archiving retains records, so it does not reclaim every quota.

16.2 Troubleshooting table

Symptom or message	Action
My workflows is empty	Confirm the current account and project; refresh; use Show archived if applicable
Imported JSON is rejected	Use a complete bundle; check exact field names, types, duplicate names, Python paths, and quotas
Unknown workflow input	Match input keys to declared parameter names; remove obsolete fields
File requires a new upload	Select the input again; do not send a desktop path or rely on a prior browser selection
Dependency unavailable	Inspect installed versus required versions in the runtime result; ask the operator to provision the needed environment
Docker image not configured	Configure the isolated assistant image and restart; manual import alone does not create an executor
Workflow changed; reload it	Refresh the current record and use its latest revision; another action or tab changed it
Active source layers changed	Save a new proposal against the chosen current source profile, then validate and activate
Runtime requires validation	Revalidate against the changed image/runtime before activation
Actual result fields/types do not match	Return exactly the declared named outputs with the correct kinds
Non-finite JSON value	Convert unsupported library values and handle undefined quantities explicitly
JSON nesting or item quota exceeded	Count value nodes and depth, not only bytes; put large arrays in bounded artifacts and keep compact inline metadata
Workflow JSON byte quota exceeded / result exceeds size limit	Inspect result serialization and output size; use section 11.10 without rounding scientific data to force a pass
AI proposal is truncated or never saved	Review provider token/time limits in section 5.4; recover a complete valid bundle before import or execution
Unsafe or missing artifact	Write a regular file beneath `context.output_dir` and return a safe relative path
Validation expectation fails	Compare actual and expected values, units, tolerances, and method assumptions; fix the cause
Execution timeout or memory failure	Reduce workload or request a justified operator resource change; use bounded validation cases
Permission denied after project change	Reconfirm the selected project and its current grant; do not rely on a stale page

Symptom or message	Action
Evidence integrity or storage failure	Stop treating that result as verified; restore/check the retained storage and matching metadata
Request timed out	Fetch workflow state and run history before resubmitting; the server may have completed the action

16.3 A GUI change does not change the result

First inspect the retained run inputs. Confirm the new value was sent and passed server validation. Then check that the program uses that key rather than a hardcoded constant or a different field. Confirm the display belongs to the newly completed run, not an older history entry.

If the program legitimately predicts the same result over that input range, explain the reason and provide another observable that demonstrates the parameter's effect. Do not alter results merely to make a control appear responsive.

16.4 A new calculation does not fit the schema

If the calculation needs a compatible installed Python API, add it to the program and normalize its output. If it needs an entirely different runtime, remote access, devices, custom client code, or larger retained data than the supported limits, it needs an additional reviewed platform capability. A private bundle cannot grant that capability to itself.

17. Reference, checklist, and further reading

17.1 Authoring checklist

Before the first activation, establish all of the following:

1. The program performs the stated calculation and returns actual results.
2. Every input has a clear name, unit/convention, default where useful, and appropriate validation.
3. Runtime requirements match the provisioned environment.
4. Returned names and kinds exactly match output declarations.
5. File inputs and artifacts use the SDK paths and retained evidence contract.
6. Representative expected values have an independent justification.
7. Failure cases are visible and scientifically interpretable.
8. The user has reviewed the validation and explicitly activated the current version.

For later studies, retain the run's exact inputs and identities with the scientific interpretation. A workflow's name alone is not enough to reconstruct an execution.

17.2 Terminology

Term	Meaning
Bundle	Declarative workflow definition plus its private Python source
Proposal	A saved bundle version awaiting or undergoing validation
Version	Immutable program/schema/check definition
Revision	Current mutable record sequence used for concurrency checking
Activation	Explicit user authorization to execute the validated current version
Adapter	Maintainer-controlled mechanism for launching a supported program form
Runtime	Actual interpreter, dependencies, container identity, and execution limits
Source profile	Chosen private layer stack over the retained scientific core
Run	One actual execution with captured inputs, status, and results
Expectation	An author-declared assertion evaluated by the trusted service
Artifact	A retained, verified input or output file
Archive	Disable/remove from ordinary selection while preserving history
Restore	Return an archived workflow inactive while retaining identity and history
Permanent deletion	Irreversibly remove the selected item and its scoped files, with retryable cleanup

17.3 Related package documentation

- `R34_PRIVATE_WORKFLOWS_GUIDE.md`: release entry point and upgrade summary.
- `AI_RESEARCH_ASSISTANT_GUIDE.md`: assistant setup, existing source layers, executor, and investigation workflow.
- `R34_INVESTIGATION_RECOVERY_GUIDE.md`: bounded investigations and continuation.
- `R33_AI_PROVIDERS_GUIDE.md`: provider configuration and protocol scope.
- `R34_BYOK_GUIDE.md`: personal AI credentials and hosted configuration.
- `PF_Simulation_r34_BYOS_Guide.md`: user storage, evidence persistence, and recovery.
- `docs/SAAS_RESEARCH_POLICY.md`: organization/project research-AI policy and disclosure controls.
- `SAAS_DEPLOYMENT.md`: hosted service, database, and operator deployment.
- `examples_workflows/`: six original complete importable examples and the sample CSV.
- `documentation/workflow_examples/`: the complete Floquet bundle with embedded source, mathematical specification, and usage notes accompanying this guide.

17.4 Scope of this guide

This guide documents the delivered workflow contracts and operating procedures. The release's validation records identify which tests, operating systems, execution modes, and external integrations were actually exercised. Do not interpret a runnable example, a developer fixture, or a successful workflow check as a full platform regression, native hardware test, or general scientific certification.

The framework provides a reusable private GUI around compatible scientific programs. The scientific content, required environment, and validated scope remain explicit for each workflow.

17.5 Reproduce documentation checks and the PDF

From the extracted package root, use the research/development environment to verify the guide's code-block syntax, exact polynomial authoring example, documented HTTP/SDK interfaces, and referenced bundle schemas:

```
python release_tools/validate_private_workflow_guide.py
```

This is a documentation check, separate from the execution and authorization test suites. An optional `--output guide-validation.json` writes a report with source-file hashes.

The PDF is generated from this Markdown with ReportLab and DejaVu Sans fonts:

```
python documentation/build_private_workflow_guide.py
```

Use `--font-dir` to select a directory containing `DejaVuSans.ttf`, `DejaVuSans-Bold.ttf`, and `DejaVuSansMono.ttf` when those fonts are not at the default Linux location. `--source` and `--output` select alternative paths. PDF generation is a documentation utility and does not execute workflow programs.

18. Shared classical reporting and research literature

The r34 reporting extension works without an external AI provider or API credentials. Open **My workflows**, upload research materials, select a workflow, select its associated materials, and choose **Save analysis and literature settings**. This single action saves the analysis/reporting settings, selected literature and roles together. Several documents may supplement one workflow and the same document may be reused by several workflows.

The adjacent status shows **Unsaved changes**, **Saving...**, then **Saved** after the server confirms success. A save error leaves the edits available to correct and retry. Button color identifies the primary action; it does not indicate save status. Saving settings does not run a workflow or regenerate an existing report. Use the separate run or report action when needed.

A completed run opens an automatic classical report when inspected. The shared core chooses applicable scientific rules, statistics/classical ML, factual NLG and optional bigram wording selection. Researchers need not select those methods. Missing data or conflicting requirements are reported explicitly. Report regeneration reads retained results and never repeats the scientific program.

Download the structured report, self-contained HTML, PDF or ZIP containing PNG/SVG figures. Cancel, retry and regenerate operate on separately retained report jobs. Existing scientific run identities, results and validation evidence remain unchanged.

Workflow bundles may add optional `reporting` metadata: dataset kinds, sources, units, basis, required/suppressed analyses, presentation order, explicit literature reference mappings and private Python extension callbacks. Scientific source layers can customize `pf_research/reporting_overrides.py`. The catalogue is extensible through the Analyzer and Registry APIs; inherited policy fields can be explicitly replaced.

The complete user/developer instructions, API table, metadata and extension examples, deployment steps, limits and validation scope are in R34_CLASSICAL_REPORTING_GUIDE.md (../R34_CLASSICAL_REPORTING_GUIDE.md). The supplied Floquet example now includes state metadata and an interaction-topology reporting extension while retaining the original physics program.

PDF text is extracted locally. Scanned pages and unsupported formats remain available as original materials with explicit limitations. Passage relevance does not establish scientific applicability; a numerical comparison requires declared units, assumptions, reference values and source locations.

18.1 Retaining and deleting research literature

Use **Archive document** for literature already in use. Existing associations, original files, extracted text and reporting remain available. Associated archived papers stay visible and selected when other choices are saved. **Restore document** returns a paper to active selection; restore it before adding a new association.

Check deletion eligibility inspects historical associations, workflow versions, report inputs, run metadata and private layers. **Permanently delete document** is available only when no retained dependencies exist. The server rechecks immediately before deletion. Missing evidence and stale previews block deletion. Detaching today's association does not remove historical dependencies.

Deletion is irreversible and removes only the unreferenced document and index. Interrupted cleanup can be retried in **Cleanup operations**. BYOS shared bytes remain while referenced elsewhere. Backup recovery refuses deletion if restored dependencies exist. Independent backups and exports are not erased.

See `R34_CLASSICAL_REPORTING_GUIDE.md` for details. Verify this release with `python release_tools/validate_r34_literature.py verify --package`.

19. Autonomous ISAES after a scientific run

An owner can opt a workflow into an unattended investigation of its completed runs. This begins from retained scientific results and an identified report; it does not replace the workflow's original program-authoring, validation or activation sequence. Existing **Discuss report** proposals still use their separate review/apply/run controls.

Declare meaningful outputs, units, frames/bases, sampling and reporting requirements in the workflow contract. Associate literature with the intended evidence role. Configure the existing provider and isolated executor, then save a standing autonomous policy that explicitly permits provider sharing and the desired capabilities. The policy can enable contextual online research, generated analysis/presentation code and follow-up scientific runs. Every investigation remains subject to cumulative budgets, owner authorization, storage integrity and lifecycle dependencies.

The investigator can select or generate methods beyond the ordinary reporting catalog using compatible installed scientific libraries. It retains the source code, actual execution, derived artifacts, claims, assessments and presentation relationships. Original solver and result bytes are preserved; a proposed or described computation does not count as an executed child run. Adaptive analyses remain exploratory unless their exact plan meets the separate preregistration conditions. Holdout roles and conflicting evidence must remain visible.

Stopped or incomplete investigations retain a bounded partial JSON/HTML/ZIP report through the existing artifact downloads. The service formats retained state only, without calling a provider or performing a new calculation after the stop. It records the stop reason, progress, evidence references, missing stages and limitations; prior presentations can be historical or stale. These snapshots do not replace the current complete native scientific presentation required for completion. Resume and completion clear the current marker but retain immutable artifacts. A storage failure remains an explicit failure to publish the partial report.

See ISAES_AUTONOMOUS.md (ISAES_AUTONOMOUS.md) for exact setup, policy/API controls, privacy, stop/resume behavior and limitations. See ISAES_AUTONOMOUS_ACCEPTANCE.md (ISAES_AUTONOMOUS_ACCEPTANCE.md) for the integrated requirements and recorded evidence scope. Local tests cannot substitute for live-provider scientific review, native Windows execution, representative researcher participation or applicable production/BYOS qualification.