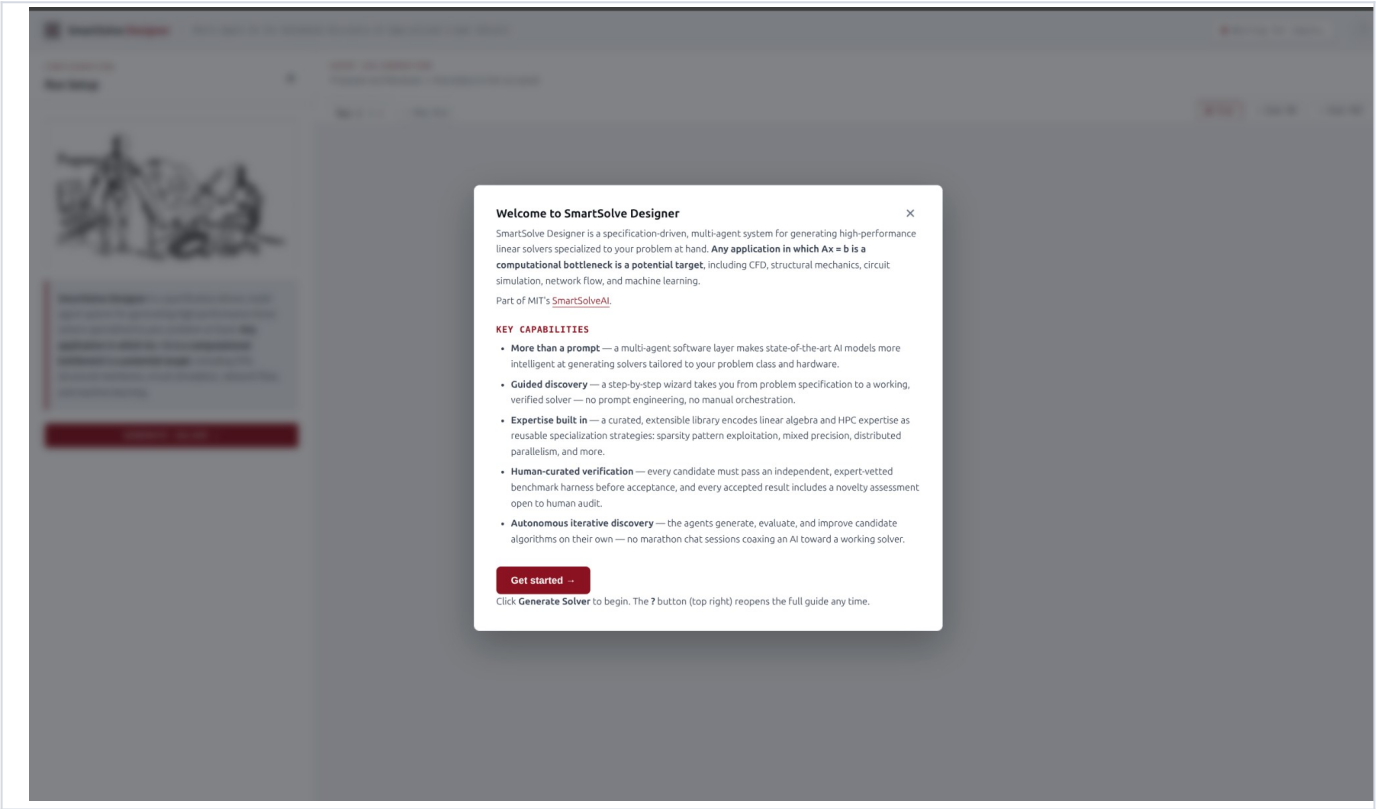
SMARTSOLVE DESIGNER

Quick Start Tutorial

SmartSolve Designer is a specification-driven, multi-agent system for generating high-performance linear solvers specialized to your problem at hand. Any application in which Ax = b is a computational bottleneck is a potential target.



The welcome guide summarizes the system and provides the entry point for a new solver-discovery run.

More information at https://smartsolveai.mit.edu/.

BEFORE YOU BEGIN

Install the tool and prepare the optional backends

SmartSolve Designer requires Claude Code and Julia 1.10 or newer. PETSc and report-generation tools are optional: missing optional components reduce the available outputs but do not prevent a Julia run.

Component	Required for	Notes
Claude Code	All runs	Install and sign in before launching SmartSolve Designer.
Julia 1.10 or newer	Julia solver and verification harness	Required even when the C/PETSc backend is also selected.
PETSc + mpicc	C/PETSc backend	Optional. The Julia backend remains available without PETSc.
pandoc + xelatex	PDF run report	Optional. Markdown and text outputs are still written when unavailable.

Install SmartSolve Designer

```
git clone https://github.com/Smart-Solve-AI/SmartSolveDesigner
cd SmartSolveDesigner
./install.sh
```

Restart Claude Code after installation. Then press Shift+Tab to turn on auto-accept mode, and launch either the terminal wizard or the graphical interface:

```
/smartsolve       # terminal wizard
/smartsolvegui    # local browser-based GUI
```

Important: enable auto-accept mode first

The Proposer-Reviewer loop performs many tool calls. Without Claude Code's auto-accept mode, the run repeatedly pauses for approval and becomes difficult to operate unattended.

How the run is controlled

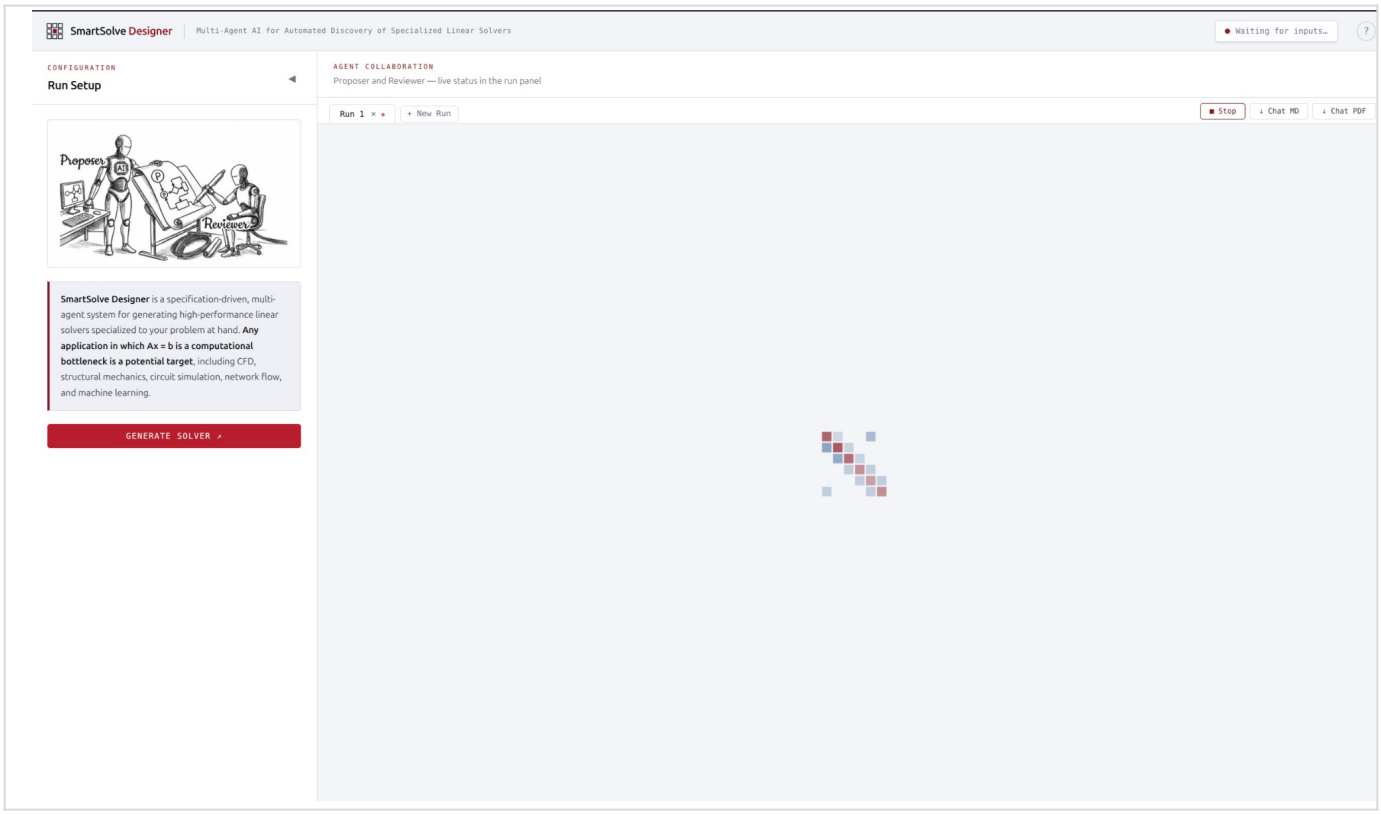
The wizard freezes a run specification before discovery begins. The Proposer creates solver candidates tailored to your problem specification; the Reviewer runs a fixed, human-curated harness and returns diagnostics until a candidate is accepted or the round budget is exhausted.

STEP 0 - LAUNCH

Open the workspace and start a new run

After /smartsolvegui opens the local web page, the workspace presents a run-configuration panel on the left and the agent collaboration stream on the right.

- Click Generate Solver in the left panel to open the eleven-step wizard.
- Use New Run to keep multiple experiments in separate tabs (risk of benchmark collision).
- The top-right status indicator reports whether the tool is waiting, preparing an environment, running, or complete.



The main workspace before inputs are supplied: configuration on the left, agent collaboration on the right.

First-run recommendation

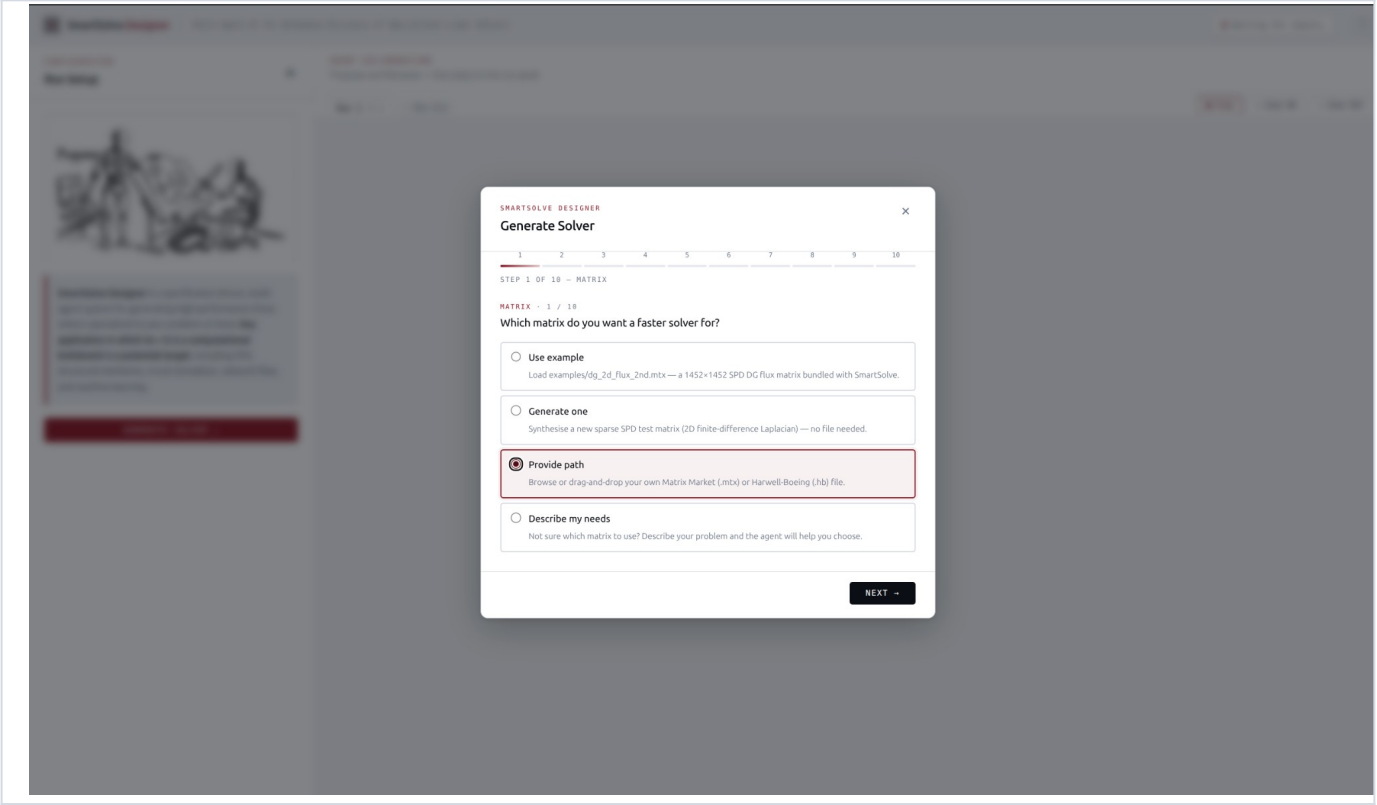
Use one representative matrix and the recommended defaults. Establish a successful baseline run before enabling multiple advanced strategies or both language backends.

STEP 1A - MATRIX

Choose how to provide the representative matrix

The first wizard step identifies the matrix for which the tool should seek a faster solver. The matrix acts as the representative problem used for structural analysis, baseline verification, and candidate benchmarking.

- Use example: fastest path for validating the installation.
- Generate one: creates a synthetic sparse SPD test matrix.
- Provide path: upload or point to a Matrix Market (.mtx) file.
- Describe my needs: ask the agent to help choose an appropriate matrix input.



Matrix-source choices in the first wizard step.

Choose a representative matrix

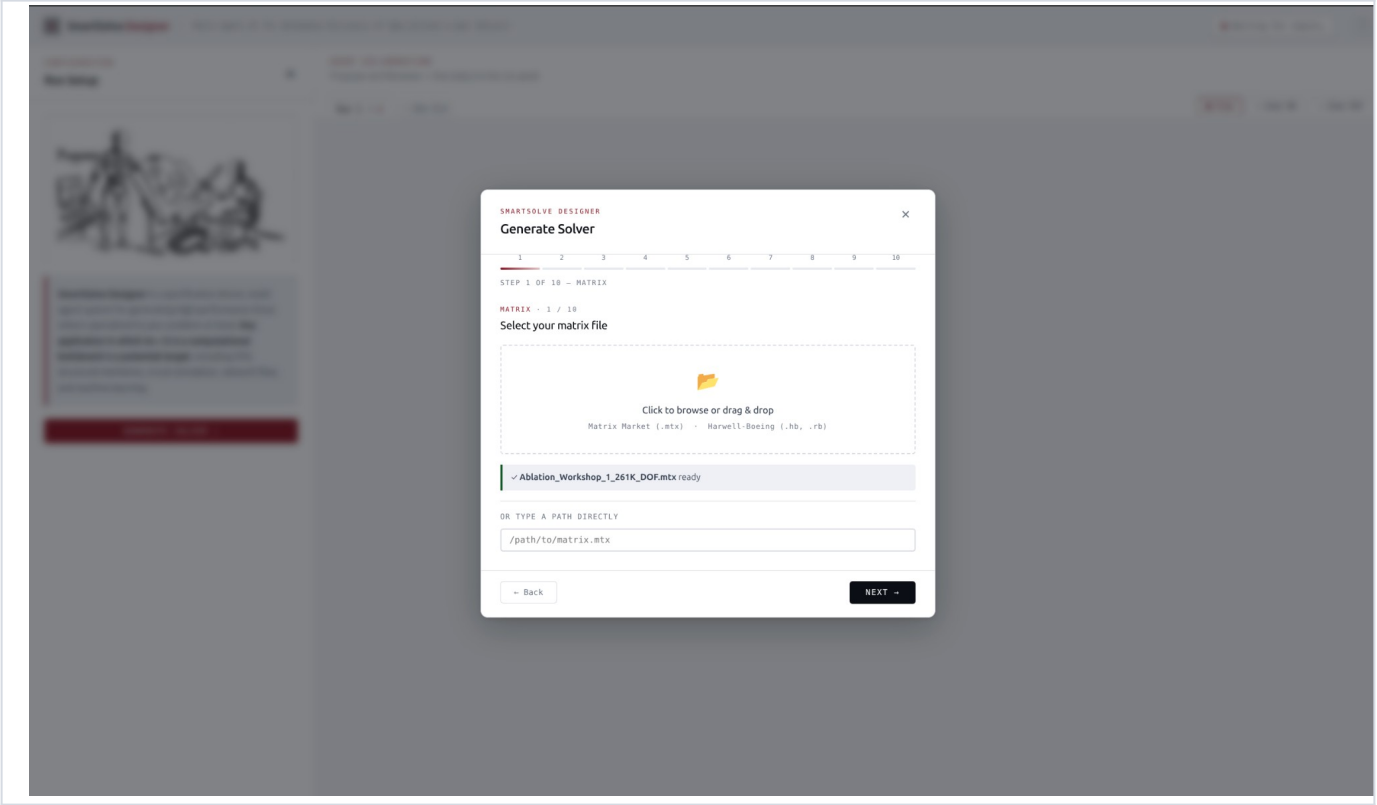
Prefer a matrix whose sparsity pattern and entry-scale distribution reflect the production workload. A solver discovered from a nonrepresentative matrix may not transfer to later systems.

STEP 1B - MATRIX

Upload the matrix file or enter its path

When Provide path is selected, drag a file into the drop zone, browse to it, or type an accessible path directly. The interface confirms the selected file before continuing.

- The GUI accepts sparse matrices in Matrix Market (.mtx) format.
- For large matrices, a direct local path avoids copying through the browser.
- Verify the filename before clicking Next; the run specification is fixed after confirmation.



A Matrix Market file loaded and marked ready for the run.

Matrix values matter

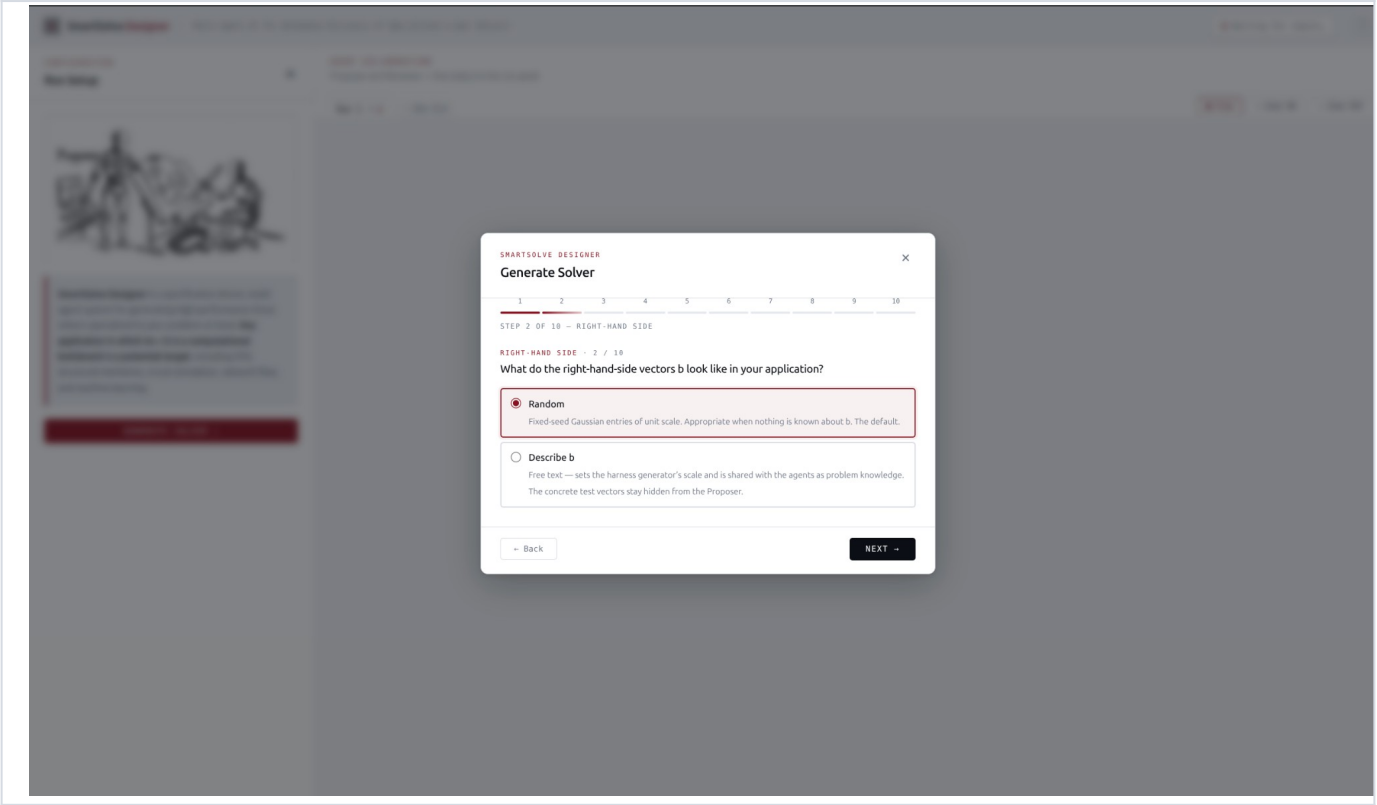
SmartSolve Designer can exploit both structure and numerical properties. Preserve the real matrix values when available; a sparsity-only surrogate may hide conditioning or value-dependent behavior.

STEP 2 - RIGHT-HAND SIDE

Describe the right-hand-side vectors

At evaluation time, the harness generates held-out test vectors from a fresh per-run seed. This step controls their scale and shares application-level knowledge without revealing the concrete vectors; the seed is logged for reproduction. On experimental CUDA, the Reviewer supplies and records the fresh seed.

- Random is appropriate when no application-specific information about b is available.
- Describe b when entries have a known magnitude, structure, distribution, or physical interpretation.
- The candidate must remain general in b; it cannot hard-code a particular right-hand side.



The default random, unit-scale right-hand-side option.

Be explicit about scale

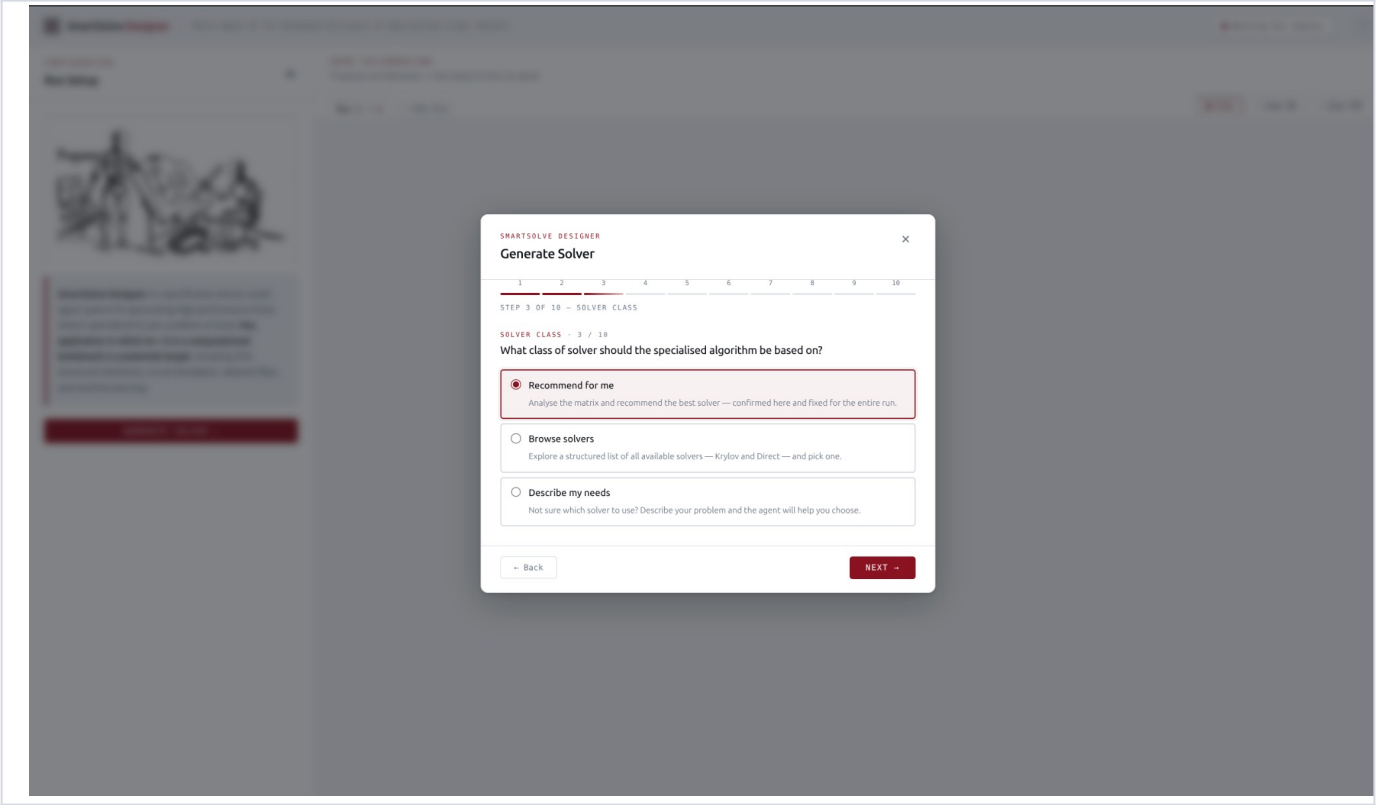
If application right-hand sides are much larger or smaller than unit scale, describe that magnitude. Residual behavior and mixed-precision choices can depend on the scale of b.

STEP 3 - SOLVER CLASS

Select the baseline solver class

Choose the solver family from which the verified baseline will be constructed. For a first run, the automatic recommendation lets SmartSolve analyze the matrix and select a suitable class.

- Recommend for me: analyze the matrix and freeze the recommended solver for the run.
- Browse solvers: choose from the available direct and Krylov methods.
- Describe my needs: provide convergence, robustness, or application constraints in free text.



Solver-class selection with automatic recommendation enabled.

The baseline is part of the experiment

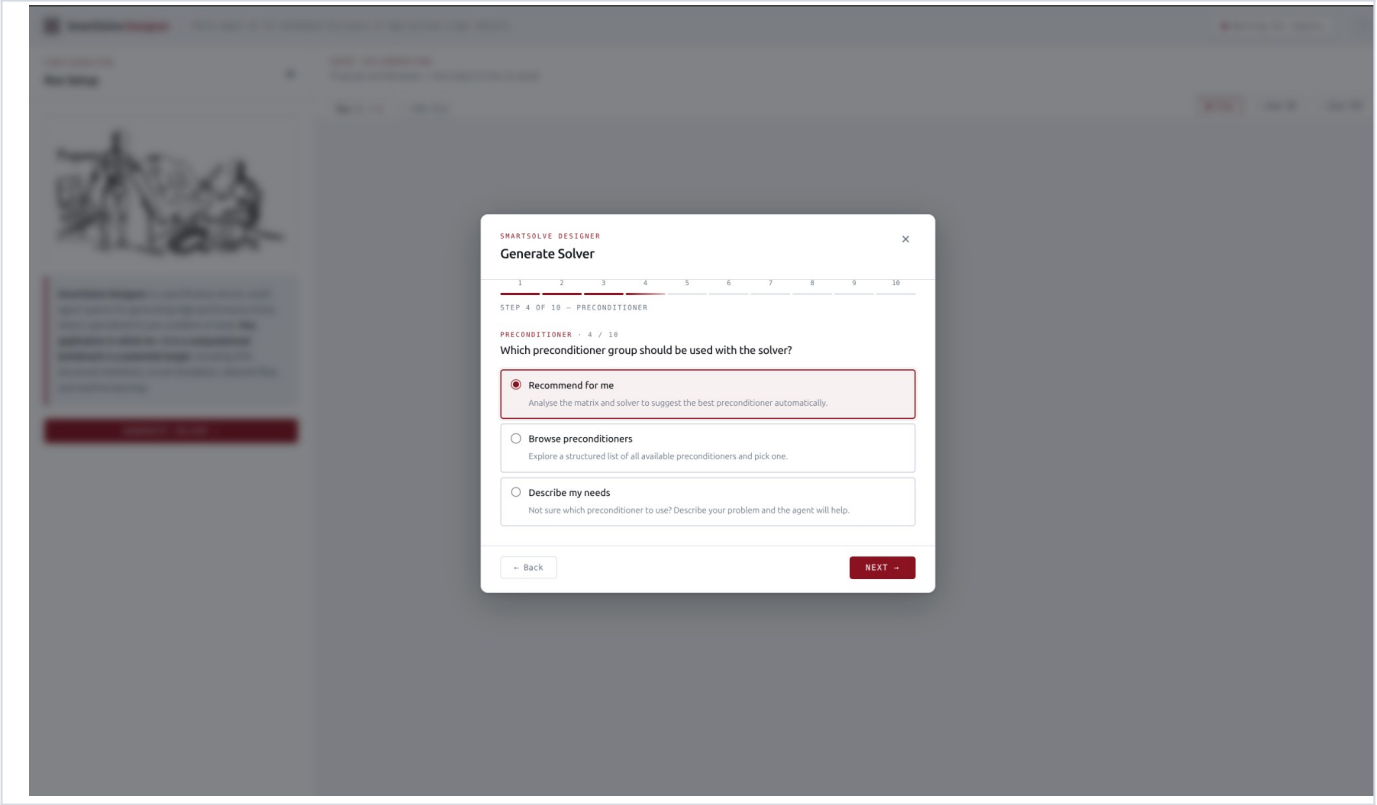
Speedups are measured relative to a verified same-language baseline, not against an arbitrary library default. A weak or nonconvergent baseline should not be used to claim improvement.

STEP 4 - PRECONDITIONER

Select the preconditioner group

For iterative methods, the preconditioner is selected separately from the solver. The recommendation can use matrix and solver information together to propose a compatible baseline configuration.

- Recommend for me is the safest first-run choice.
- Browse preconditioners when the application already has a trusted baseline.
- Describe my needs to communicate constraints such as setup cost, fill level, decomposition, or available libraries.



Preconditioner selection with automatic recommendation enabled.

Direct methods may not need a separate preconditioner

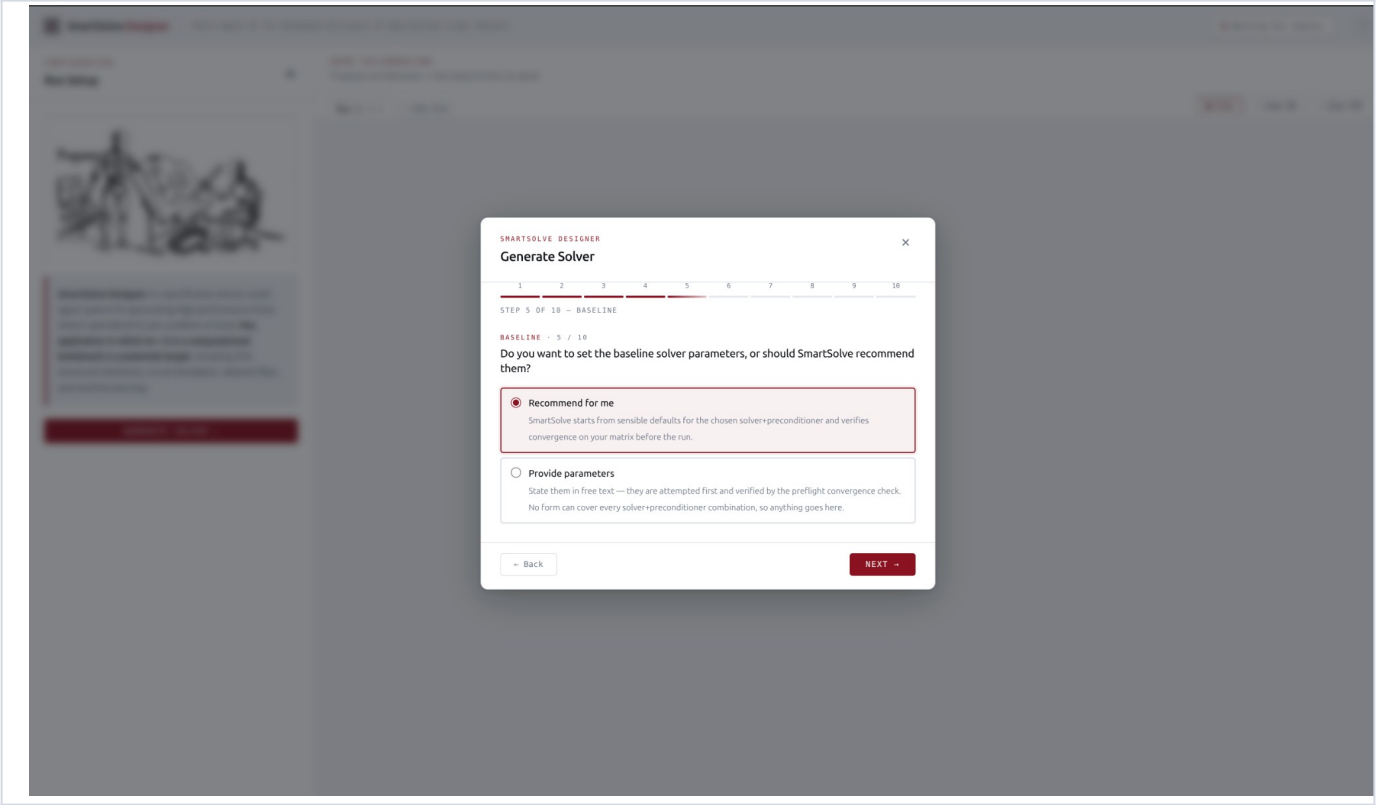
The interface still treats solver and preconditioner selection as explicit specification steps so the benchmark configuration is reproducible and auditable.

STEP 5 - BASELINE

Let SmartSolve verify baseline parameters

This step controls parameters such as restart length, overlap, fill level, subdomain count, or stopping tolerance. The preflight check attempts the supplied configuration first, then verifies a convergent baseline before discovery begins.

- Recommend for me starts from sensible defaults and automatically verifies convergence.
- Provide parameters accepts free text because solver-preconditioner combinations expose different controls.
- The verified parameters are frozen and used for every candidate comparison in the run.



Baseline-parameter selection. Automatic recommendation is suitable for the first run.

Do not skip baseline validation

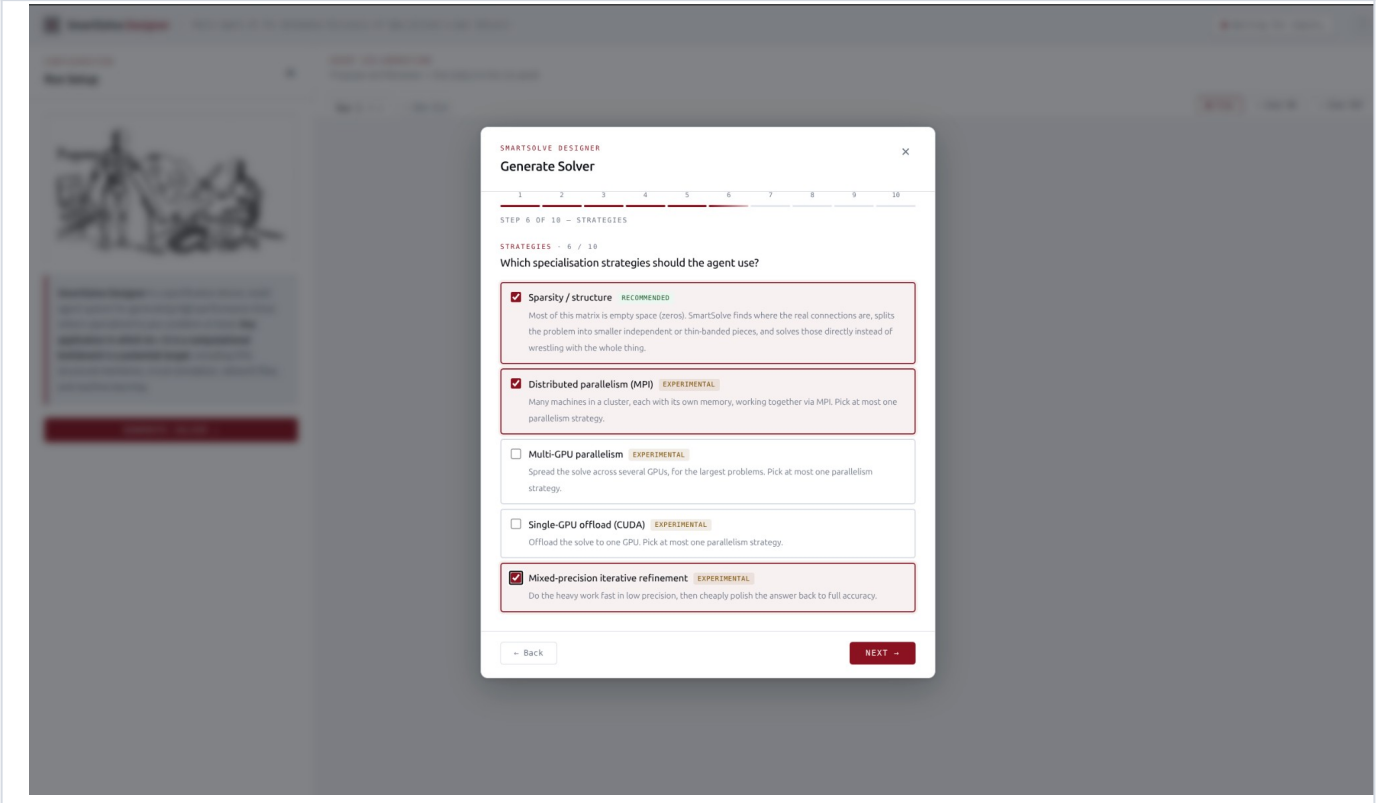
A solver that reports convergence may still have an unacceptable true residual. SmartSolve evaluates solutions against the original system Ax = b and establishes the baseline before candidate timing begins.

STEP 6 - STRATEGIES

Enable the specialization strategies

Strategies are reusable specifications that guide the Proposer toward particular classes of optimization. Select only techniques that are permitted by the intended deployment environment.

- Sparsity / structure: explicit-zero removal, independent components, bandwidth reduction, block structure, and repeated work.
- Distributed parallelism (MPI): distribute genuinely independent work across ranks.
- Single-GPU or multi-GPU: offload or distribute the solve across accelerators.
- Mixed-precision iterative refinement: use lower precision for expensive stages and recover accuracy with higher-precision residual correction.



Strategy selection. The screenshot illustrates several enabled strategies; a first run should usually begin with sparsity/structure only.

At most one parallelism strategy

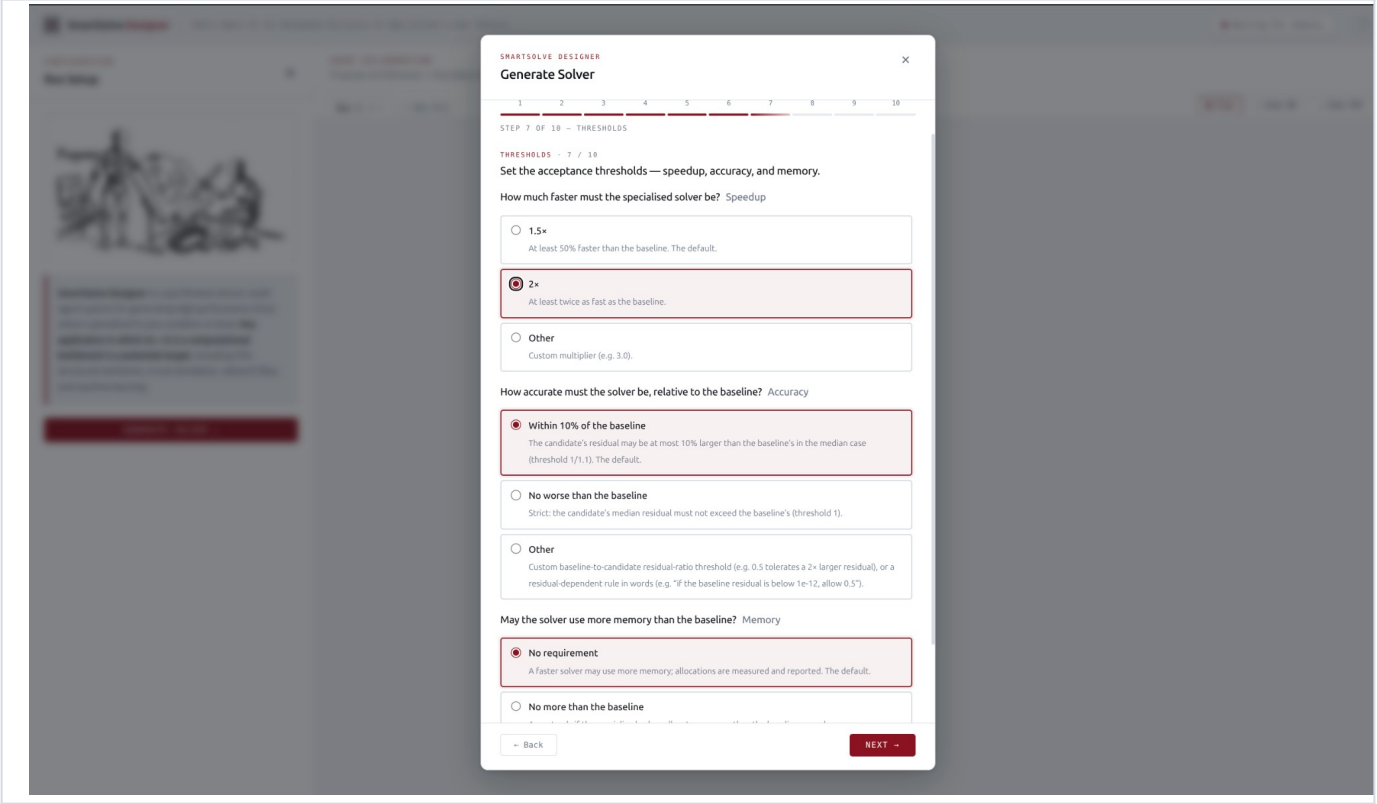
MPI, single-GPU, and multi-GPU strategies are mutually exclusive. When none is selected, candidates target shared-memory CPU execution.

STEP 7 - ACCEPTANCE THRESHOLDS

Define what counts as an acceptable candidate

The Reviewer applies speed, accuracy, and memory requirements to the candidate relative to the verified baseline. These thresholds define the run contract and should reflect the downstream application.

- Speedup: 1.5x is the default; 2x is a more demanding target; custom ratios are supported.
- Accuracy: the default allows the median candidate residual to be at most 10% larger than the baseline residual.
- Memory: no requirement is the default; stricter options can require parity or improvement.



Acceptance-threshold panel with a 2x speed target, default accuracy tolerance, and no memory requirement.

Use realistic thresholds

An aggressive speedup target can consume the entire round budget without an accepted solver. Begin at 1.5x, then rerun with tighter requirements after confirming the workflow and baseline.

STEP 8 - USAGE PATTERN

Define what happens between solves

This step fixes what work may be reused by both the baseline and candidate. It is part of the run contract and can materially change the honest speedup.

- Solve once: rebuild all setup every solve; caching-only speedups do not count.
- Reassemble A every step: reuse pattern-derived structure, but recompute value-dependent work and numeric factorization.
- One fixed A, many right-hand sides: both sides may reuse the numeric factorization.
- Other: describe the solve loop; SmartSolve must map it to an unambiguous symmetric contract before evaluation.

Apply reuse symmetrically

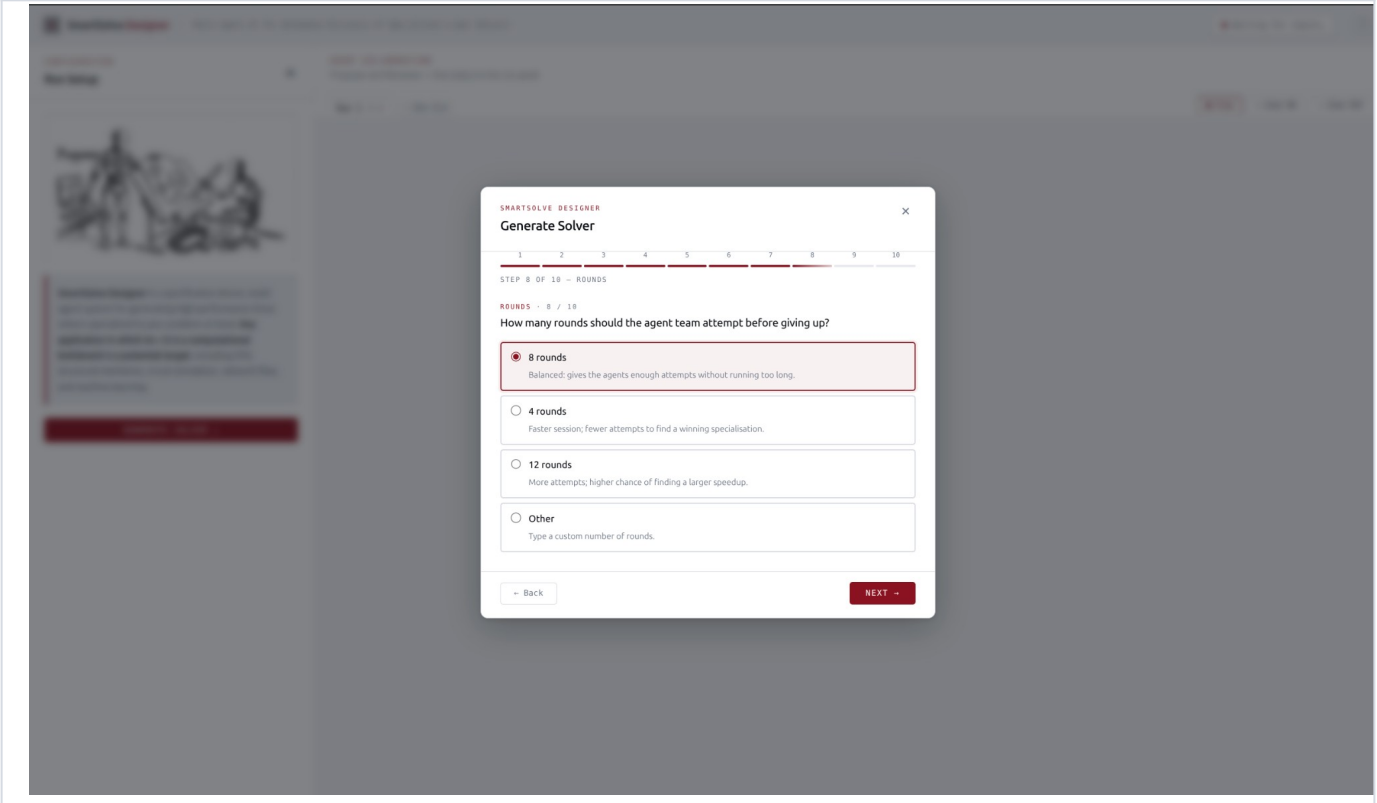
Any setup or factorization reuse allowed for the candidate must also be allowed and verified for the baseline. Choose Solve once when unsure.

STEP 9 - ROUNDS

Set the discovery budget

The round budget limits the number of Proposer-Reviewer iterations. Each rejected candidate produces structured diagnostics that guide the next proposal.

- 4 rounds: quick exploratory run.
- 8 rounds: balanced default for most first experiments.
- 12 rounds: larger search budget with higher runtime and model usage.
- Other: enter a custom budget when the experiment has a fixed resource limit.



The default eight-round discovery budget.

A round is not just another prompt

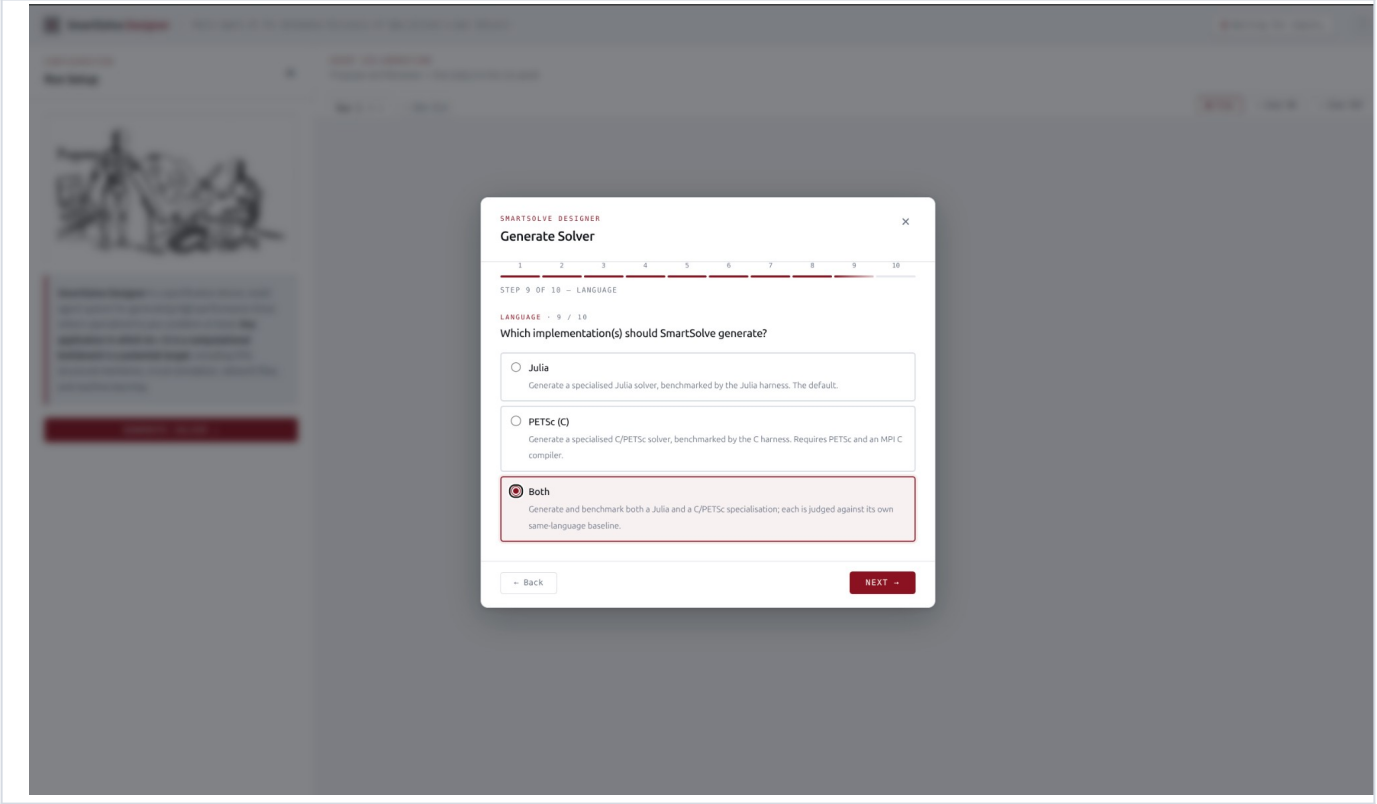
Each iteration may include code generation, compilation, benchmark execution, diagnostics, and revision. Use auto-accept mode and avoid closing the Claude Code session while the run is active.

STEP 10 - IMPLEMENTATION LANGUAGE

Choose the generated backend

SmartSolve Designer can generate a Julia implementation, a C implementation using PETSc, or both. Each backend is evaluated only against a baseline in the same language.

- Julia: recommended first-run choice and available with the core requirements.
- PETSc (C): requires PETSc and an MPI C compiler such as mpicc.
- Both: produces independent Julia and C/PETSc candidates and reports.



Language selection with both Julia and C/PETSc enabled.

Do not compare backends as a language benchmark

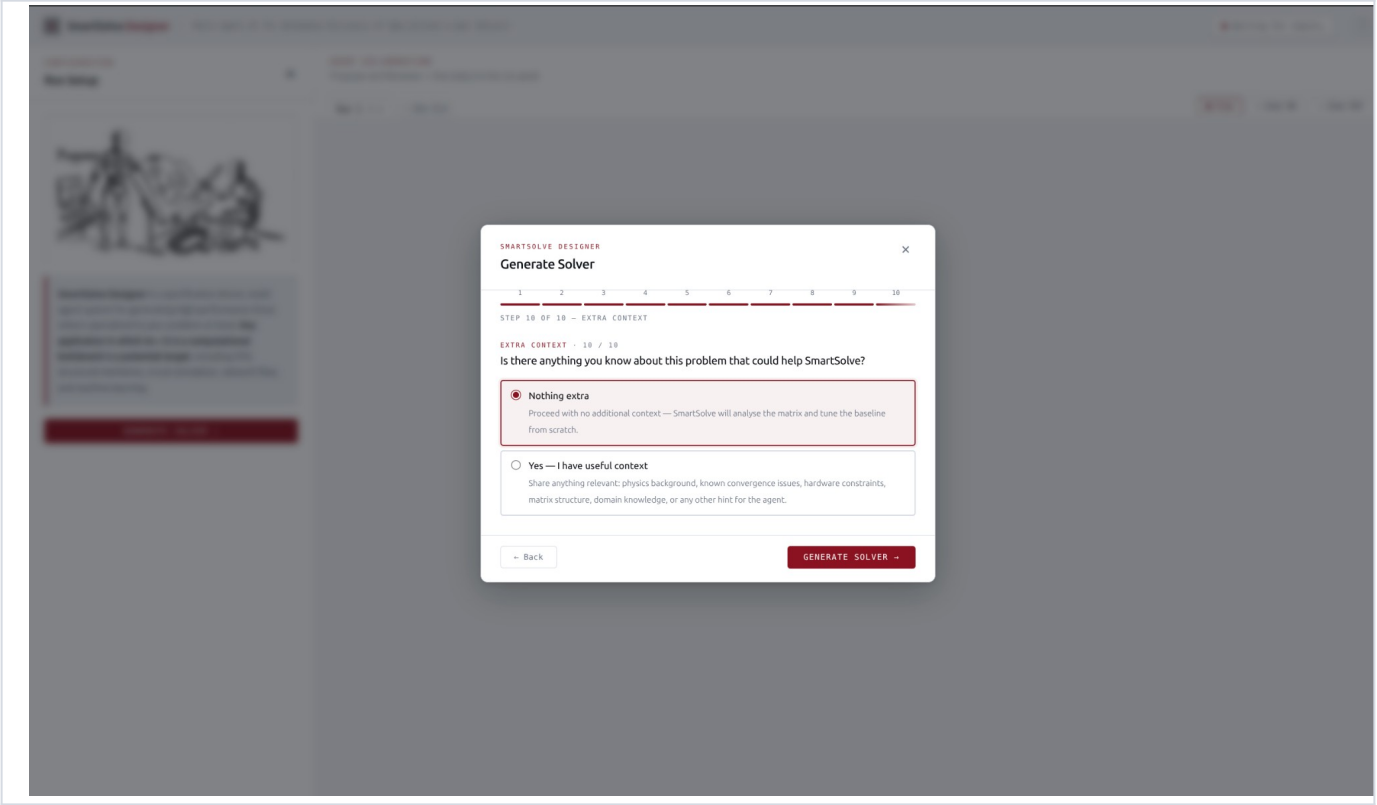
Same-language baselines isolate algorithmic improvement from differences in compilers, runtime systems, libraries, or threading models.

STEP 11 - EXTRA CONTEXT

Add domain knowledge and launch the run

Use the final step to provide information that cannot be inferred reliably from the matrix alone. The text becomes part of the fixed specification used by the agents.

- Useful context includes the physics, expected matrix sequence, known convergence issues, and hardware constraints.
- Record known failure modes, trusted parameter ranges, or application invariants the agents should verify.
- Specify the fastest objective only when engineering optimizations are acceptable; the default objective emphasizes strategy-grounded specialization.



The final wizard step. Click Generate Solver after reviewing all selections.

Keep context concise and testable

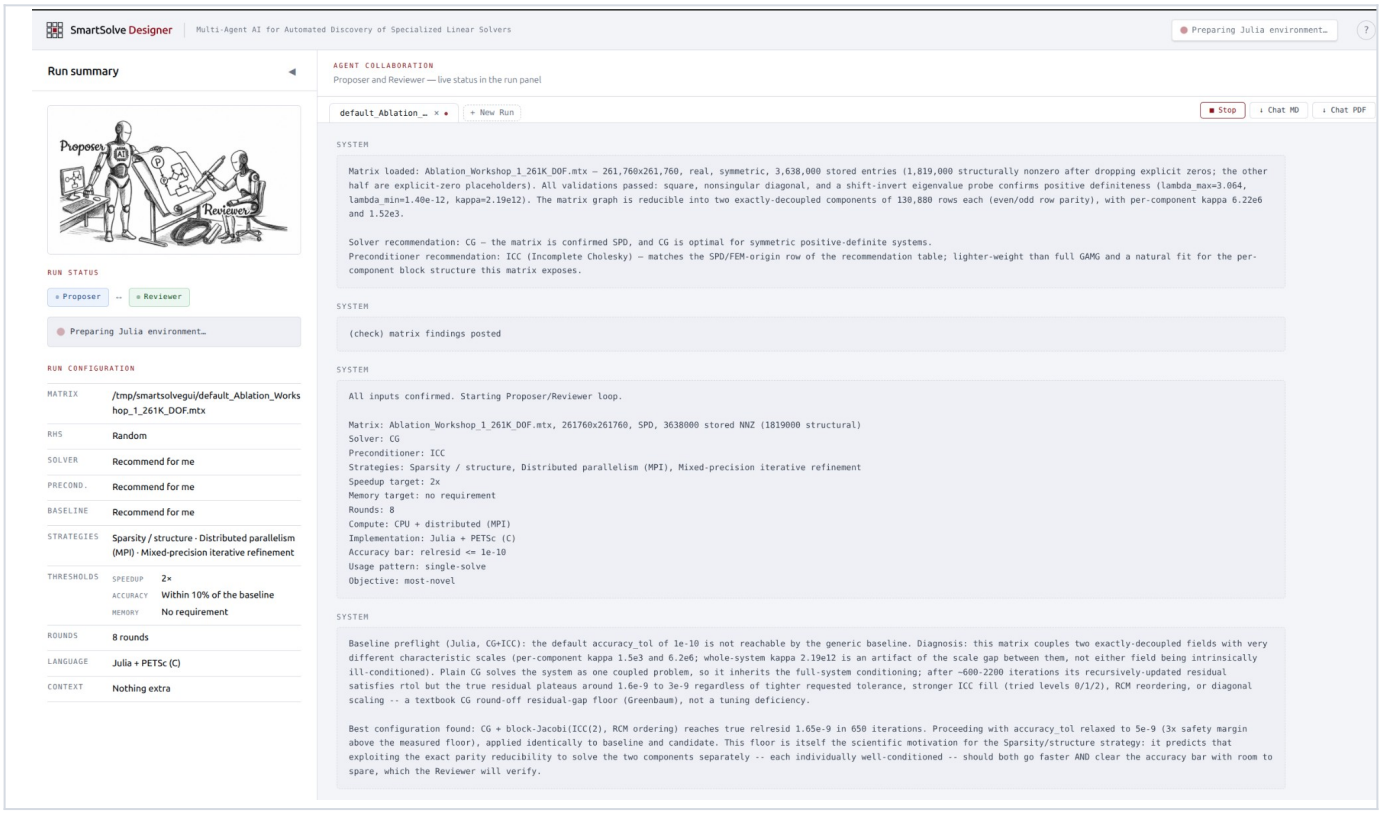
Prefer concrete facts such as "3D elasticity with three DOFs per node" or "must run on four MPI ranks" over broad requests such as "make it very fast."

STEP RUN - AGENT COLLABORATION

Monitor baseline verification and the discovery loop

After launch, the configuration is summarized in the left panel while the right panel records system analysis, baseline preflight, Proposer activity, Reviewer decisions, and diagnostics.

- Confirm the loaded matrix dimensions, symmetry/definiteness checks, and any detected structural facts.
- Review the recommended solver and preconditioner before candidate generation begins.
- Watch for preflight warnings: the tool may replace a nonconvergent or misleading baseline configuration.
- Use Stop only when the run must be interrupted; stopping early may leave incomplete artifacts.



An active run after matrix analysis and baseline preflight. The run summary preserves the fixed specification.

Read the diagnostics, not only the final status

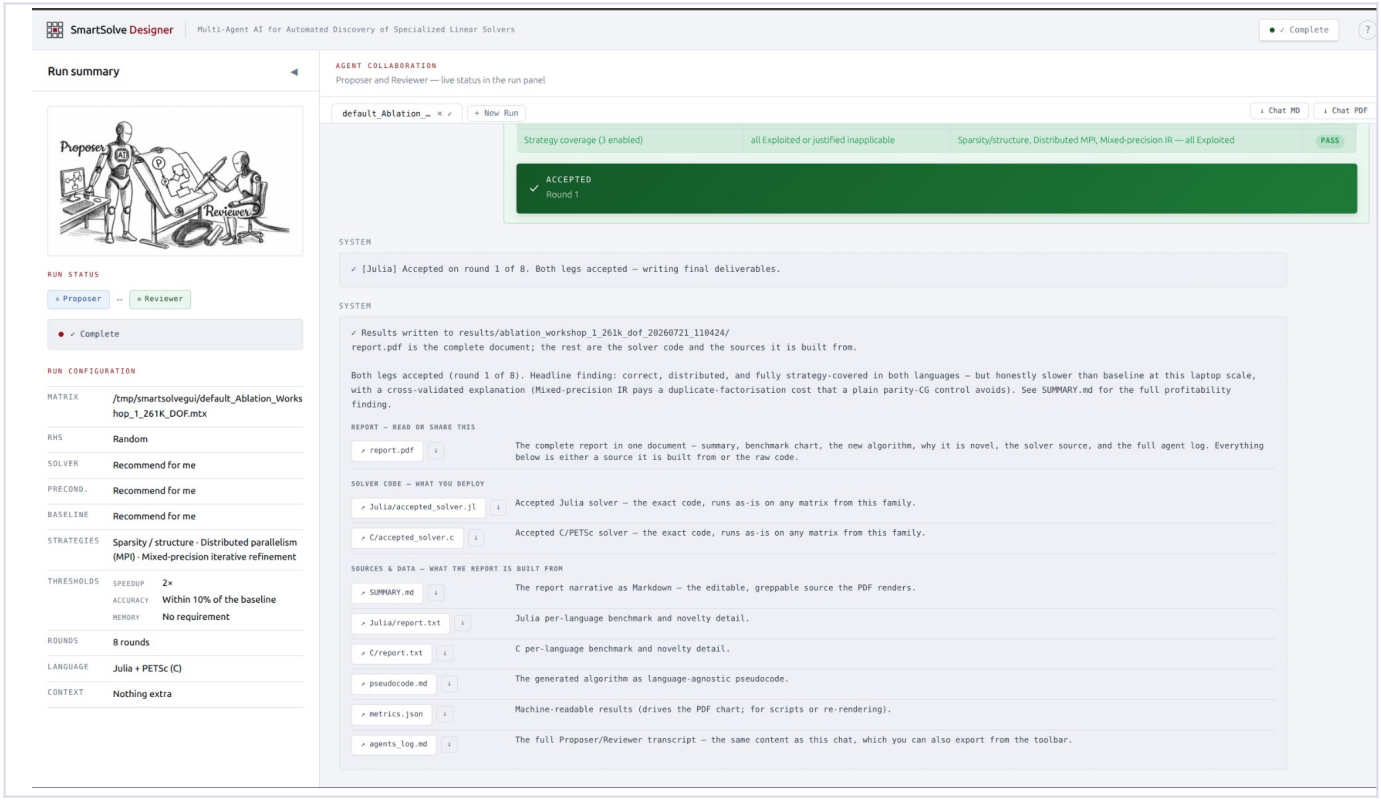
Rejected candidates provide useful evidence about compilation, residuals, convergence, runtime, allocations, and strategy applicability. This evidence often explains why a later candidate succeeds.

STEP COMPLETE - RESULTS

Review the verdict and download the artifacts

At completion, the interface presents the Reviewer verdict and links to the generated report, accepted source code, pseudocode, metrics, and full agent transcript.

- report.pdf: consolidated narrative, benchmark evidence, algorithm description, and agent log.
- accepted_solver.jl / accepted_solver.c: deployable generated source for the selected backend.
- SUMMARY.md and backend report.txt files: concise and language-specific findings.
- pseudocode.md: language-agnostic algorithm description.
- metrics.json: machine-readable benchmark and verification results.
- agents_log.md: unedited Proposer-Reviewer transcript for audit and reproducibility.



The completion view exposes the report, accepted solver code, structured metrics, pseudocode, and full transcript.

Validate in the full application before deployment

Harness acceptance is evidence that the candidate satisfies the run contract on the benchmark matrix and held-out right-hand sides. Integrate the solver into the target application and confirm end-to-end accuracy, performance, and scaling.

REFERENCE

Recommended settings for a first successful run

The following configuration minimizes optional dependencies and keeps the search focused while you verify the installation and workflow.

Wizard input	Recommended first run
Matrix	Bundled example or a representative .mtx matrix
Right-hand sides	Random, unit scale, unless the application scale is known
Solver / preconditioner / baseline parameters	Recommend for me
Strategies	Sparsity / structure only
Acceptance thresholds	1.5x speedup; within 10% of baseline residual; no memory requirement
Usage pattern	Solve once (conservative default)
Rounds	8
Language	Julia
Extra context	Nothing extra, or concise domain information

Where the results are written

```
results/<matrix>_<timestamp>/
  report.pdf        SUMMARY.md       metrics.json
  pseudocode.md     agents_log.md
  Julia/accepted_solver.jl   Julia/report.txt
  C/accepted_solver.c        C/report.txt
```

Quick troubleshooting

- The wizard pauses repeatedly: press Shift+Tab to enable auto-accept mode before launching SmartSolve.
- No C/PETSc output: verify PETSc discovery and that mpicc is available; run Julia only meanwhile.
- No PDF report: install pandoc and xelatex; use SUMMARY.md and text reports meanwhile.
- Baseline does not converge: provide known parameters or allow the preflight to recommend alternatives.
- No candidate reaches the target: lower the speed threshold, increase rounds, or narrow the enabled strategies.
- A generated solver does not scale in production: treat the discovered structure as guidance for an iterative preconditioner or a different hardware strategy, then rerun with realistic context.

Reproducibility checklist

Keep the representative matrix, run configuration, metrics.json, accepted source, and agents_log.md together. Record the software commit, hardware, and full-application validation separately from the discovery report.

Project resources: SmartSolve Designer repository | SmartSolveAI | Claude Code