BEFORE THE RUN: THE CHECKS A MODEL MUST PASS

A step-by-step guide of everything that has to go right with an interconnection model package **before an MQT simulation can even begin**, whether a person or a program does the checking, and why doing it by hand or with custom-built tools is slower, costlier, and less reliable.

The failure that matters most is not the one that stops you. It's the one that lets you continue.

A model that fails loudly is easy. The expensive case is the one that reads in cleanly, runs to completion, and produces a confident-looking answer that is quietly wrong. This trap is in the file and the physics; **it is just as invisible to a human reviewer as to a custom-built script**. The difference is reliability: a human or a custom-built script is far more likely to wave the bad model through, while purpose-built software is built to catch it.

8
distinct stages between upload and a trusted result

5+
incompatible file formats to read (.sav .raw .dyr .pscx .tsa)

20+
manufacturer model families to recognize

~35
dynamic-model block types to interpret

8
file layouts that must each be parsed differently (the filename won't say which)

1,000s
MQT runs completed

STAGE 1
Making sense of the package

STAGE 2
Decoding the binary case (.sav)

STAGE 3
The network as text (.raw)

STAGE 4
Dynamics & vendor identity (.dyr)

STAGE 5
The electromagnetic model (.pscx)

STAGE 6
A real plant at a real point

STAGE 7
The gate: fatal or advisory

STAGE 8
The run itself, across all conditions

How to read this brief. No power-systems background is required. Each stage is one step in the journey from "a customer uploaded a ZIP" to "a quality test can run." For each, we say in plain terms what has to be true, a few real ways it breaks, and (the point of the whole document) why that difficulty is **inherent to the work itself**, not to any one method of doing it. What matters is the sheer number of independent things that must all go right before anyone, by any means, can trust the result.

ACT I

JUST GETTING THE BOX OPEN

STAGE 1 Making sense of the package

Before any physics, **someone** has to make sense of a folder a human assembled and zipped by hand. The most common first problems aren't physics at all; they're a misassembled package.

- The archive is unreadable or empty, or it contains nothing recognizable as a model.
- There are two of something where there must be exactly one (two base networks, two PSCAD projects) and no one can know which is real without asking.
- A dynamics file is missing, a Mac left a hidden `__MACOSX/` junk folder, or a `/` in a filename breaks the machine that runs the simulation later.

WHY IT'S HARD

The ambiguity is in the package, not the tool. **A human or a custom-built script** is unreliable here: a quick script has to anticipate every one of these cases one at a time, and a person eyeballing an unfamiliar folder simply guesses, and a wrong guess quietly corrupts everything downstream. Purpose-built software recognizes the layout, strips the junk, and reconstructs the intended package automatically, and where an input genuinely is missing it identifies exactly what, up front, instead of discovering it mid-run.

STAGE 2 Decoding a proprietary binary (the `.sav` case)

A `.sav` file is an **undocumented binary memory dump**: no labels, no schema, no map. To read it at all, its electrical network has to be rebuilt by **parsing the raw admittance arrays** and walking an internal linked list, byte by byte.

- The layout silently shifts between software versions: record offsets drift, and in older files bus numbers are stored `×1000`. Filenames lie about which version they are; only the raw bytes can be trusted.
- It often doesn't fail loudly. It "succeeds" and quietly drops data; a missing admittance array silently removes every line and transformer. The case looks loaded; the grid is gone.

WHY IT'S HARD

This format **cannot be read by eye**. **A human or a custom-built script** is unreliable here: a person opens it in a viewer that carries the same version blind spots and never sees the dropped transformers, and a quick script reads the wrong layout and drops them silently. Reconstructing it byte by byte, version-aware, is the only thing that reliably catches it.

ACT II

READING THE MODEL

STAGE 3 The network as text (`.raw`)

The `.raw` file lists the grid as text, but **the column layout changed between versions**, and the file rarely says so cleanly.

- A v33 file read with v35 rules shifts one column, so a status flag is read out of a voltage field and the parse breaks.
- A single off-by-one once dropped every branch owner; older files have no section markers, so blank-but-present header lines must be counted exactly.

WHY IT'S HARD

The version traps live in the file, not the reader. **A human or a custom-built script** is just as likely to misread a shifted field and pass a half-empty network; a person is slower, and a single-pass script has nothing checking it. The reliable way is two independent parsers that cross-check and refuse to return a hollow network.

STAGE 4 The dynamics, and whose equipment it is (`.dyr`)

The dynamics file says how each device behaves in a disturbance. But first you have to **recognize whose device it is**, from inconsistent free-text names.

- Vendor "sentinel" values such as `±999,999` mean "unlimited," not a real limit; taken literally, they corrupt the model.
- Malformed vendor data (a maximum below its minimum, `qmax<qmin`) has to be caught and flagged, not silently used.
- Every new manufacturer family is one more to support; the catalog is never "done."

WHY IT'S HARD

You have to know each manufacturer's quirks cold. **A human or a custom-built script** that hasn't accounted for 20+ vendor conventions takes a sentinel literally or misses an impossible limit: unreliable, and inconsistent from one reviewer or script to the next. A maintained, cross-checked library is what catches it every time.

STAGE 5 The electromagnetic model (`.pscx` , PSCAD)

A PSCAD model carries its protection logic as a **drawing, not a list**. There is no wiring table to read.

- The circuit has to be reconstructed from how the wires are **positioned on the page**, and the relay logic traced backward.
- Manufacturer identity is inferred from names, and several patterns can match one file at once.
- Some vendors ship **compiled black-box files** whose settings cannot be read at all and must be hand-entered from a separate document.

WHY IT'S HARD

The logic only exists as a diagram, so it has to be traced connection by connection. **A human or a custom-built script** does this slowly and unreliably (hours of eye-tracing, or a script that misreads the geometry) and both are equally blocked by the compiled black-box files. Software built to reconstruct the circuit from its geometry is what makes it tractable.

ACT III

DOES IT PHYSICALLY MAKE SENSE?

STAGE 6 A real plant at a real point on the grid

Even a perfectly read model has to describe a **sane plant at a defined point of measurement**. Get this wrong and every downstream number is wrong.

- The point of measurement can fail to resolve in five distinct ways (no reference bus, several, multiple paths, parallel ties…).
- Rated output has no reliable source inside the file; it has to come from the interconnection agreement.
- Picking the wrong transformer silently yields the wrong voltage class.

WHY IT'S HARD

The ambiguity is in the data. **A human or a custom-built script** guesses from the same unclear file and moves on; purpose-built software reports the specific reason it cannot decide rather than quietly choosing: the distinction a quick check rarely makes.

STAGE 7 The gate: fatal, or just a warning?

Every problem found so far has to be triaged: **does it block the work, or is it advisory?** The wrong call either halts good work or passes bad work.

- Blocking: unreadable archive, malformed vendor data, unsupported software version.
- Advisory: surfaced and logged, but work may proceed.
- Fail-safe rule: anything uncertain is treated as **blocking**; "not sure" means "stop."

WHY IT'S HARD

Triaging every issue as fatal or advisory is a judgment call. **A human or a custom-built script** makes it inconsistently; it varies by who is reviewing and how tired they are, or by whatever the script's author happened to anticipate. A coded rule applies the same standard every time and defaults to safe.

ACT IV

THE DEEP END

STAGE 8 Pass the base case, fail when it's actually run

A model can solve cleanly at its starting point and still **fail once it is actually exercised**, and there is no way to know until every condition has been run.

- It passes the base case, then **fails in a leading power-factor condition**, or fails in lagging, or fails at a different dispatch level.
- It is stable at one operating point and diverges at another, sometimes a full second into the run.
- A setting that is fine for one manufacturer's model is too coarse for another, and the instability only appears mid-run.

WHY IT'S HARD

This is the failure a pre-submission check cannot catch. **A human or a custom-built script** looks at the base case, sees it solve, and signs off; the failures hide in the conditions no one ran. The only way to surface them is to run the full matrix of conditions, which is the test itself.

THE SAME GAUNTLET, THREE DIFFERENT ENGINES

ENGINE	TYPE	ITS OWN DISTINCT DIFFICULTY
PSS/E	RMS	A full steady-state setup, then a stability solve that must hold across every operating condition; three proprietary file formats; vendor library compatibility.
PSCAD	EMT	A compiler toolchain and license certificates; opaque compiled vendor files; long runtimes; protection logic recovered from a drawing.
TSAT	RMS	In our pipeline TSAT runs on PSS/E-format inputs, so it inherits PSS/E convergence issues and adds its own dependencies and version drift.

HONEST BOUNDARIES

Some of this remains genuinely hard **for everyone**: the newest binary revision has no reference data to check against yet; a handful of low-level numeric edge cases are unavoidable. We name the frontier explicitly, because the surface is large enough that **"it was validated" only means something when you can say exactly what was, and was not, checked.** A manual reviewer rarely can.

CONCLUSION

THE WORK IS HARD, AND SO IS THE SOFTWARE THAT CATCHES IT

The work is inherently hard. A quick check, by a human or a custom-built script, is unreliable and likely to wave it through. Software built for this, and tested against it, catches it far more consistently.

THE WORK THAT MUST HAPPEN	A HUMAN OR A CUSTOM-BUILT SCRIPT	WITH PURPOSE-BUILT SOFTWARE
Read a proprietary binary case	Can't be read by eye; opened in a viewer with the same version blind spots, so dropped data goes unseen	Reconstructed byte-by-byte, version-aware, in seconds
Track layout drift across file versions	Held in one expert's memory, or hard-coded for one version in a custom-built script; one misread silently drops data	Encoded and regression-tested for every version
Recognize 20+ vendor model families	Memorized per reviewer, or a regex a custom-built script never finishes; sentinels taken literally	A maintained, cross-checked library applied the same way every time
Catch the quiet, plausible-but-wrong result	A quick check, by a person or a script, signs it off; it surfaces later, once it's been relied on	Guarded checks fail loudly and save a snapshot

Catching these reliably takes machinery that neither a human nor a custom-built script has:

Built on real failures

Thousands of MQT runs completed on real model packages; the failure modes here are ones we have already hit and handled, not theory.

A regression safety net

Every fix is pinned by a test, so a model that broke once never silently breaks the same way again.

A growing vendor library

Each new manufacturer's model format is supported as it appears; knowledge no single reviewer holds in their head.

A precise failure taxonomy

Coded categories and severities turn "it didn't work" into an exact, actionable reason.

A quick check, by a person or a program, can pass a model that later fails when it is studied.

That is why a quick check alone is not enough: the failures it waves through become costly cycles of delay.

GridStrong

MODEL QUALITY, VALIDATED · GRIDSTRONG.AI