AI-Assisted Development Guide

How `mascot_qc_report.py` was actually built.

What this document is

This repository is a working QC dashboard for Mascot Server. But it is also a teaching artefact. The script was built across two Claude Code sessions in dialogue with a proteomics scientist, and the interesting part — the part worth sharing — is not the final script. It is how the final script's shape emerged from iteration.

This guide presents that iteration honestly. You will see mid-thought course corrections and moments where the model was pointed at the right answer rather than guessing it. You will also see, in the appendix, what the same project might look like expressed as a single "starting fresh" prompt — so you can compare the iterative route to the synthesised one and draw your own conclusion about which produces better scientific software.

Our own take, stated up front: the appendix prompt is useful as a launch pad, but it is the transcript that teaches the method. Consolidation always loses the corrections that sharpened each decision.

The twelve pivotal prompts

Filler, confirmations and pure bug reports are omitted. Session 1 is the main build; Session 2 is a short follow-up where the user also commissioned this guide. Prompts are presented in the user's own voice and original phrasing; spelling has been lightly corrected.

Turn 1/12 — Initial scope

Session: 1

> "I want to make a simple QC example report, just one or two plots that show an overview of the whole run. This is an example for a blog post. Use the /mascot-parser skill. We will call this script after the search completes. I was thinking of a 2D plot of retention time vs m/z and spot size by intensity and color related to score. Use diffuse shapes rather than clean circles. Use Nature publishing guidelines for the colors. The script will be called by a Mascot Daemon external task and we can pass it the url to the results file. There are some other example graphs in the notes directory."

Why it mattered: In one paragraph the user established the purpose (blog-post-grade QC example), the runtime context (post-search Daemon external task, URL-driven), the central visual (RT vs m/z, intensity-sized, score-coloured), the aesthetic constraint (Nature publishing colour guidance, diffuse marker shapes) and the reference corpus (`notes/`). Almost every

later decision was a refinement of this scaffold rather than a departure from it.

Outcome: Claude loaded the `mascot-parser` skill, skimmed the notes directory and produced the first single-panel `matplotlib` scatter — a working skeleton with placeholder score colouring.

Turn 2/12 — Dashboard layout and DIA switch

Session: 1

> "Underneath the main 2D plot we want to show a visual representation of the TIC. It could be a spark plot or it could be a bar or 1D gel style with varying intensity. Adding a RT vs ID count plot is also a good idea. We want to lay it out in a dashboard format, where the extra plots are aligned with the main 2D plot. 2, make a switch to report rank 1 only or matches that are significant so that it can be used with DIA data too."

Why it mattered: This is where the script stopped being "a plot" and became "a dashboard." The user specified (a) column-aligned small multiples, (b) a TIC representation expressed as options ("spark / bar / 1D gel") rather than a fixed design — letting Claude choose — and (c) a first-class DIA mode via a rank-1-vs-significant switch. The alignment requirement drove the eventual `gridspec` layout.

Outcome: Claude refactored to a multi-panel `gridspec` figure — RT/ID-count line at top, TIC strip below the main scatter, all sharing the X axis — and added a CLI flag to toggle rank-1 vs significant PSMs.

Turn 3/12 — Scoring must be scientifically correct

Session: 1

> "The circles in the 2D plot are too large. Rather than having three sizes they should be dynamic/relative to intensity. What are you plotting with regard to the score? There should be ~24k significant matches in each run. You should use the Mascot score not the identity threshold. There is machine learning involved so use the corrected score after the ML has been applied."

Why it mattered: The user caught two things in one breath: a cosmetic issue (discrete marker size bins) and a scientific one (Claude was plotting the wrong quantity). The "~24k significant matches" is the user testing the script against a known-good reference number — a smell-test Claude hadn't thought to run. This pivoted the whole scoring pipeline onto the Percolator-rescored score.

Outcome: Claude switched marker sizing to a continuous intensity map and began wiring in Percolator / MS²Rescore results to use the rescored score.

Turn 4/12 — Percolator the right way

Session: 1

> "The plots use rescored score. That is still not correct. C:\Users\richardj\Qsync\dev\Matrix Science\peptide_view2 does it properly. Also Percolator integration - do it properly. It might be fragile but it is scientific data, it needs to be done with the published method."

Why it mattered: A blunt correction: Claude's first Percolator integration was approximate. The user pointed to an existing reference implementation (`peptide_view2`) and stated a principle that carried through the rest of the project — *"it might be fragile but it is scientific data, it needs to be done with the published method."* No fudging.

Outcome: Claude studied `peptide_view2`, rewrote the score extraction to follow the published Percolator → Mascot-score mapping, and updated the `mascot-parser` skill itself so the correct recipe would be reusable.

Turn 5/12 — Hexbin with meaningful axes

Session: 1

> "Make a hexbin plot too. I like the two distribution plots at either side of this example https://seaborn.pydata.org/examples/hexbin_marginals.html The score is not directly related to the intensity, RT or m/z. In general you hope for a relationship — the higher the intensity the better the spectra the better the score — but it is not guaranteed."

Why it mattered: The user articulated *why* the hexbin matters scientifically — it visualises a soft, non-deterministic correlation (intensity → spectral quality → score) rather than a law. That framing justified the fourth panel's existence and set up the later two-layer design: the point isn't to prove a relationship but to reveal where it holds and where it breaks.

Outcome: Claude added a `seaborn.jointplot`-style hexbin of precursor intensity vs rescored score as the fourth panel.

Turn 6/12 — Score-to-Mascot-score formula, pinned

Session: 1

> "The hexplot in the combined figure should not be cut off and I think it needs a y-axis too. This means extending the range above what is currently plotted. Are you using the /mascot-parser routine to calculate the score from the PEP value? The actual calculation should be: Note that, in general, the scores are lower after switching to Percolator. The Posterior error probability is tabulated in the expect column. A Mascot score is calculated from the expect value and the single score threshold, which we describe as the identity threshold, has a fixed value of 13 (-10 log 0.05). By keeping the score, threshold, and expect value consistent, we hope to avoid breaking any third party software that expects to find these values."

Why it mattered: The user pasted the exact Matrix Science documentation prescribing how Percolator scores become Mascot scores — fixed identity threshold of 13, derived from `-10 log 0.05`. This made the formula non-negotiable and canonicalised it in the script (and the reusable skill).

Outcome: Claude codified the `score = -10 × log10(expect) + 13` relationship, fixed the hexbin Y-axis clipping, and verified score ranges matched across the scatter and hexbin panels.

Turn 7/12 — Palette discipline and colour-role separation

Session: 1

> "Check the way /mascot-parser converts the Percolator PEP score to a replacement Mascot score — it is different to what they are using here. Also the Percolator bar shows scores up to say 80 but the hexplot shows scores higher. Shouldn't their ranges match? I think we need to choose a different set of colors for the hex plot as people will get confused that in one plot the highest color is score and the other the highest color is count of queries."

Why it mattered: A visualisation-literacy point masquerading as a cosmetic one. Two panels with the same colour ramp but different semantics (score vs PSM count) creates a cognitive trap. The user also demanded consistent numeric ranges between panels — so a reader's eye can trust them as a set.

Outcome: Claude split the colour roles: a sequential "count" colormap for the hexbin, a distinct qualitative/diverging one for score, and aligned the score ranges across the scatter and hexbin colourbars.

Turn 8/12 — Nature / Lancet palette steer

Session: 1

> "Grays are not doing it for me. The colors used originally are great for the hexplot. Change up the colors used for the 2D plot. We want a mixture of colors just not the same ones as the hex plot. See this guide https://research-figure-guide.nature.com/figures/preparing-figures-our-specifications/#we-require"

(Followed shortly by: *"lots more palettes here https://cloud.r-project.org/web/packages/ggsci/vignettes/ggsci.html"* and *"Lancet is good for the 2D plot"*.)

Why it mattered: Rather than dictate hex codes, the user supplied *sources of authority* (Nature figure guide, ggsci palette catalogue) and let Claude pick within them, then confirmed "Lancet" for the scatter. This is delegated-taste direction — fast and durable.

Outcome: Claude implemented ggsci-derived palettes — Lancet for the scatter's score colouring and a distinct sequential map for the hexbin — giving the dashboard its final look.

Turn 9/12 — Two-layer hexbin (the key scientific-visual idea)

Session: 1

> "Identified peptides are defined as significant matches. For the 2D plot only identified peptides should have a color. For the hexplot I don't think there is an easy way to mark the unidentified peptides unless you use a separate gray scale color for the number of peptides and the hex is either a gray or color depending on if the majority are unidentified or identified. Kinda complicated. I'm not sure if you could draw black edges along the hexes where the transition from majority unidentified to identified happens — that would be a better option."

Why it mattered: The central design idea of the final figure. The user is thinking aloud through a hard problem — how to show *all* PSMs (for context) while emphasising *identified* ones (for conclusions) — and arrives at the two-layer concept: grey background of all queries, coloured overlay of significant PSMs. This is the "grey context + coloured significant PSMs" commit message.

Outcome: Claude implemented the two-layer hexbin — a grey underlay of all queries with a coloured layer for significant PSMs on top — which became the final panel design.

Turn 10/12 — Daemon runtime and output-location constraint

Session: 1

> "Are we expecting it to be saved in the MGF task directory? All the images are in the C:\inetpub\mascot\data\20260415\ directory. If Mascot Daemon is on a different computer to Mascot Server then that won't work. In general we always want to save to the task directory. Also comment out the generation of the qc_joint.png image."

(Immediately followed by: *"we need to pass it* `<task_directory>`*"* — the literal Daemon substitution variable.)

Why it mattered: This is where the script's operational model crystallised. The user corrected a cross-machine assumption — writing into the Mascot data directory assumes local access — and set the rule that outputs must land in the Daemon *task* directory passed as `<task_directory>`. Trimming redundant outputs (`qc_joint.png`, later the PDF) sharpened the deliverable.

Outcome: Claude added a `<task_directory>` CLI argument, made it the output root, resolved it before any `chdir`, and pruned the joint/PDF outputs.

Turn 11/12 — Remote execution and HTTP refactor

Session: 1

> "On a fundamental side I think we should be opening the results on the Mascot Server and streaming the data out of them rather than downloading them locally. How is the actual script working?"

> "We need to use the msparser HTTP client for this to work correctly when Mascot Daemon is running on a different computer to Mascot Server. Refactor"

Why it mattered: The biggest architectural pivot of the project. The original code assumed local filesystem access to `.dat` files. The user rejected that as fundamentally wrong for the Daemon-on-a-different-box case, and named the correct tool — `msparser`'s HTTP client. This reshaped the I/O layer of the script.

Outcome: Claude refactored file access onto the msparser HTTP client, later supplementing with an authenticated pull of the intermediate `.msr` via `ms-status.exe?Show=RESULTFILE` (informed by the user's follow-up URL hint) — giving the script a genuine remote-Mascot capability.

Turn 12/12 — Extract repo, hoist licence, generalise paths

Session: 1 (late)

> "This is an example script that is used to demonstrate AI development tool use with Mascot. The plots generated can be used in a real QC application later on. I think we need to split this off to a separate small repo that can be shared with customers. Call it MascotQCexample. [...]"
>
> "The script also requires msparser so we need to set that up. [...] note that the sys.path.insert(0, r"C:\Users\richardj\Qsync...") uses my local path, that's not good! There are a bunch of Mascot paths that all presume you have local access to the Mascot Server and that you are running on Windows. This script needs to run as if it is remote from Mascot Server and be platform independent although the most likely platform it will be used on is Windows."
>
> "Change the license from MIT to GNU."

Why it mattered: The shift from "my script" to "shareable example." The user demanded three things at once: extract into a public-facing repo (`MascotQCexample`), remove all personal/local path assumptions (platform-independent, environment-driven), and switch the licence from MIT to GPL v2+ (visible in the git log as commit `005d174`). This converted the code from a one-off into a teaching artefact — which is why this guide is possible at all.

Outcome: Claude scaffolded the new repo, moved msparser resolution to environment variables and `.env`, added `.env.example`, introduced `*.bak` to `.gitignore`, and replaced the MIT licence with GPL v2+.

Honourable mentions

Three prompts were strong but either overlapped the chosen twelve or were narrower than the "design turning point" bar:

- *"Make the data points smaller so they can be seen clearer. Have a quick look for how you should display 70 points on a graph in a readable fashion. Are there libraries for this?"* — a nice example

of asking Claude to **research the right idiom** rather than dictating a solution; led to alpha-blending and the first hexbin consideration.
- *"The upper mass of the main plot should auto adjust based on the max mass of the data. Say plus 10%."* — small but principle-bearing: axes should be data-driven, not hardcoded.
- Session 2's *"I want to use this example as a guide to using an AI generated report [...] We could either create a transcript of the major design prompts or create a prompt that would generate a similar report. What do you think is best?"* — the meta-prompt that commissioned this very guide, and a good closing note for the teaching narrative: the development session itself becomes curriculum.

What the twelve prompts teach about the method

Reading them in order, three patterns carry the project:

1. **The user sets principles, not pixels.** "Nature publishing guidelines", "the published Percolator method", "task directory, not data directory", "platform-independent". Claude fills in the specifics, the user keeps the invariants.
2. **Scientific correctness trumps convenience.** The Percolator rescoring was redone twice because the user wouldn't accept an approximation, even when told it might be fragile.
3. **Constraints arrive as pivots, not up-front specs.** The remote-Mascot / HTTP-client refactor, the `<task_directory>` output rule and the two-layer hexbin all emerged mid-build. The script's shape came from iteration, not from a one-shot prompt.

If you take one thing from this transcript: budget for iteration. The consolidated prompt below looks like a spec you could hand to the model and walk away. It isn't. It is a summary of decisions that already happened. The real script was built the hard way, and that is what made it correct.

Appendix A — Consolidated "starting fresh" prompt

A distillation of the requirements that survived the twelve turns above, expressed as a single prompt you could hand to a fresh Claude Code session. Use it as a launch pad, not a substitute for iteration. Expect the model still to surface wrong assumptions you will need to correct — exactly the decisions you just read about in the transcript.

> **Task**
>
> Write a single Python script, `mascot_qc_report.py`, that produces a one-page PNG dashboard summarising the quality of a Mascot Server search result. It will be invoked as a Mascot Daemon "Execute after search" external task:

```
python mascot_qc_report.py <result_url_or_path> <task_directory>
```

Write the output PNG to `<task_directory>\<result_basename>_qc.png`.

Runtime context

- Must work when Daemon and Mascot Server are on **different machines**. For remote access, use the msparser HTTP client (`ms_http_client`, `ms_http_helper`) to pull the `.msr` via `x-cgi/ms-status.exe?Show=RESULTFILE` before handing the local copy to `createResfile`.
- No hardcoded paths. Resolve Mascot Server locations from `MASCOT_HOME` (default `C:\inetpub\mascot` on Windows, `/var/www/mascot` on Linux) with individual overrides: `MASCOT_DATA_DIR`, `MASCOT_CACHE_DIR`, `MASCOT_CGI_DIR`, `MASCOT_DAT`.
- Credentials from a `.env` file (`MASCOT_URL`, `MASCOT_USER`, `MASCOT_PASS`); ship a `.env.example`.
- Platform-independent code, but Windows is the common target.

Panels — single shared-RT figure

1. **ID count vs RT —** filled-line histogram of significant PSMs per RT bin, across the top.
2. **TIC strip —** 1-D "gel-style" heatmap of precursor ion current along the gradient, directly below panel 1.
3. **RT vs m/z scatter —** every query as a diffuse marker (alpha-blended, not hard circles). Marker size = log(precursor intensity). Colour = Percolator-rescored score for significant PSMs; unidentified queries shown as pale grey ghosts. Y-axis upper bound = `max(m/z) × 1.1`.
4. **Intensity vs score hexbin —** two layers: a grey underlay of *all* queries for context, and a coloured overlay of *significant PSMs only*. Y-axis extends above the highest score — no clipping. The X/Y ranges of the overlay must match the score ranges shown on the scatter's colourbar.

Panels 1, 2 and 3 share the X axis and are column-aligned via `gridspec`.

Aesthetic

- Follow the Nature figure preparation guidelines (`research-figure-guide.nature.com`).
- Use **ggsci** journal palettes: **Lancet** for the scatter (score colour scale), **NPG** for the hexbin (density colour scale). The two panels MUST use visually distinct palettes, because "high colour" means different things in each.
- Expose via `--palette`: `npg`, `aaas`, `nejm`, `lancet`, `jama`, `jco`,

`bmj`, `frontiers`, `okabe_ito`, plus matplotlib perceptual maps (`viridis`, `magma`, `plasma`, `cividis`).

Scoring — non-negotiable

This is scientific data. Follow the published Mascot Parser Percolator workflow exactly, even where it looks fragile:

1. `chdir` to the Mascot CGI directory before opening the result file (msparser resolves paths relative to CWD).
2. Open with a writable cache directory.
3. Call `setPercolatorFeatures(mascot_options, '', adapter_params)` after setting `PercolatorExeFlags` via `getPercolatorRtFlags()`.
4. Retrieve expected `.pip` / `.pop` filenames with `getPercolatorFileNames()`.
5. If the hash-derived filenames do not match the files the server generated (option drift between runs), **copy the real cache files to the expected names —** the workaround recommended in the msparser docs.
6. Construct `ms_peptidesummary` with the two-argument form (via `ms_mascotresults_params`) and set `MSPEPSUM_PERCOLATOR` in `flags2`.

With the flag set, `pep.getIonsScore()` returns `-10 × log10(PEP)` and the identity threshold is a fixed 13 (`-10 × log10(0.05)`). Label the scatter's colourbar "Percolator score (-10 log10 PEP)".

Fall back to raw rank-1 Mascot ions score (`--rank1`) only when Percolator cache files are absent.

CLI

```
positional: <result_url_or_path> <task_directory>
--significant (default)       filter by Percolator q-value
--rank1                       all rank-1 matches, raw ions score
--fdr 0.01                    q-value cutoff for --significant
--palette lancet              palette for score colour scale
--render scatter|density|hexbin  main panel style
--adapter-param KEY=VALUE     repeatable, passed to the ML adapter
```

ML adapter params (e.g. `MS2Rescore.ms2pip_model=HCD2019`) are also accepted inline in the result URL.

What to avoid

- Do not plot raw rescored scores and call it "Percolator integration" — use the published method or flag that Percolator is off.
- Do not write outputs into the Mascot data directory; outputs go to `<task_directory>`.

> - Do not use discrete marker-size bins; size is continuous in `log(intensity)`.
> - Do not share a colour ramp between two panels with different semantics.
> - Do not emit extra PNG/PDF files — one dashboard, one PNG.
>
> **Licence**: GPL v2 or later.
>
> Produce the script and a `README.md` documenting installation, Daemon external-task configuration, environment variables, and all CLI flags.

A note on comparing the two routes

The transcript is twelve turns; the consolidated prompt is two pages. The temptation is to treat the short one as superior because it is shorter.

But the consolidated prompt only exists because the twelve turns happened. Every bullet under "What to avoid" is a mistake that was made and corrected. Every non-negotiable in the Percolator section is a shortcut the model initially took. Every named palette is the result of "Lancet is good for the 2D plot." Strip those twelve turns away, hand the appendix prompt to a fresh session, and you will still need roughly twelve turns of correction — just different ones, probably less scientifically informed ones.

The lesson for AI-assisted scientific development is not "write a better prompt." It is: **treat the prompt as the first move in a conversation, not the last**. Iterate in public, with the domain knowledge kept with the human, and the boilerplate kept with the model. The code is the output; the transcript is the method.