Author a Model Configuration

Experimenter Build a reusable model set-up, which decides which models answer and how they combine, so that any run can pick it at launch.

Screen Model-configurations `model-configurations.html` (models come from Model-catalog `model-catalog.html`)

Writes the set-up `model_config` its member models `model_config_model`

Key fields how they combine `combine_method`

self-improvement `max_iterations` · `score_target` · `min_score_gain` · `underperform_threshold`

vote weight / merge lead / routing tier `weight` what the router matches on `model_catalog.expertise`

1 · Given / When / Then

Human wording first; the exact table/field in `mono`.

GIVEN	At least one model is enabled in the **Model-catalog Screen** (the install's menu). The experimenter has the **experimenter** or **admin** role.
WHEN	On the **Model-configurations Screen** they name a reusable set-up and add one or more models from the catalog, optionally giving each a relative `weight`; every member is a generator. They pick a `combine_method` (`single`, `synthesize`, `vote`, `consensus`, `judge_best`, or `moe`) — when several models answer, the one trusted judge configured on the experiment settles on the winner, and under `moe` that same judge is also the tie-breaking router. They can also turn on self-improvement (`max_iterations`, `score_target`, `min_score_gain`, `underperform_threshold`). Then they Save.
THEN	A `model_config` and its `model_config_model` rows are written. Together they form a reusable "one logical model" that any run can pick at launch. The simplest set-up is one model with `combine_method = single`, which skips all the ensemble machinery. **writes** `model_config`, `model_config_model`

Variations and exceptions

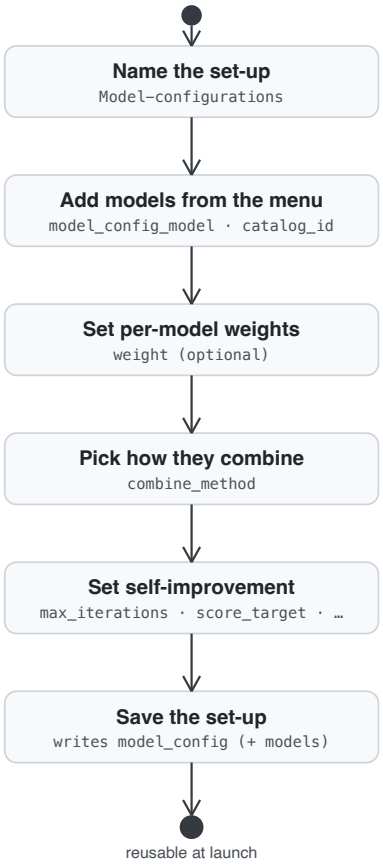
IF…	THEN…
`combine_method = single` with more than one member, or any other method with only one	Refused on save: single needs exactly one model and every other method at least two. `moe` on a one-model set-up is refused too — a router with nothing to choose between is `single` with extra machinery.
A `moe` set-up whose members have no `expertise` sentence in the catalog	Allowed, but warned about: with nothing to match, every question reads as a tie and every round pays for the judge's tie-break call.
A member model rejects a custom temperature	It is skipped by a capability list; the rest of the set-up runs as configured.
They try to delete a catalog model that a set-up uses	Refused; the model is retired with `enabled = false` instead, so existing set-ups still resolve (*Curate the Model Catalog*).

IF…	THEN…
Any run references the set-up	Its substantive fields and member models are read-only and it cannot be deleted (`RESTRICT`); only `notes` stays editable. To change it, clone it into a new, distinctly named configuration.

Acceptance test — *Given* an enabled catalog model, *When* the experimenter saves a set-up with one or more catalog models and a valid `combine_method` , *Then* a `model_config` + matching `model_config_model` rows exist and appear in the launch picker.

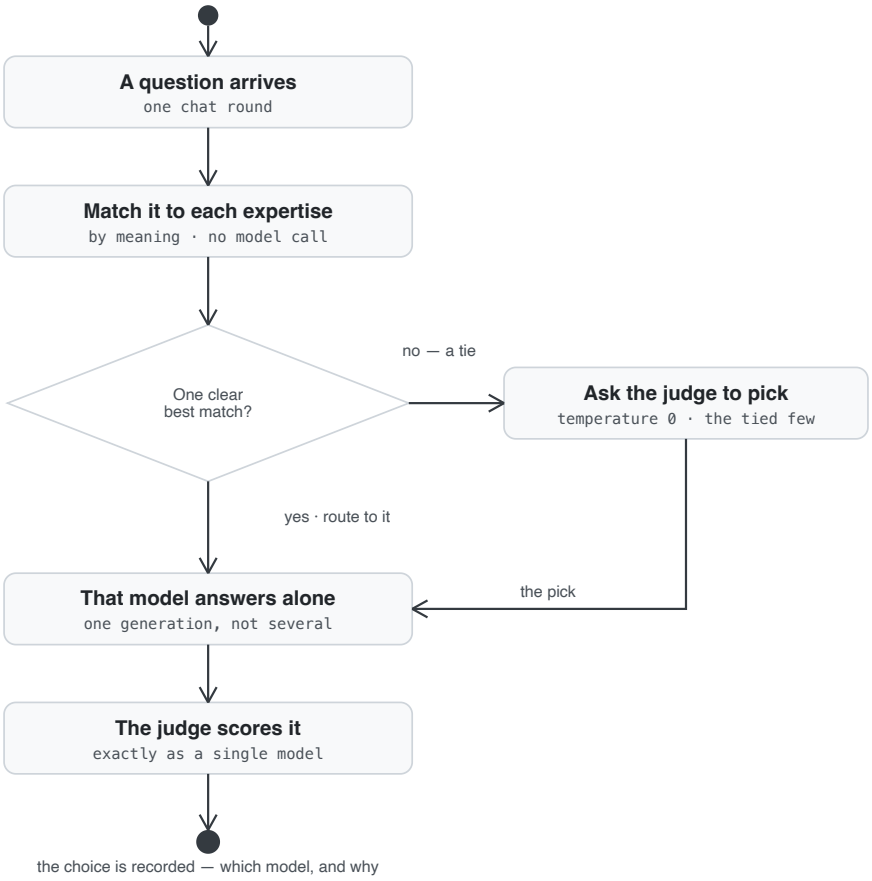
2 · The Journey, Screen by Screen

One screen; Save writes the set-up and its member models.



3 · How Mixture-of-Experts Routing Decides

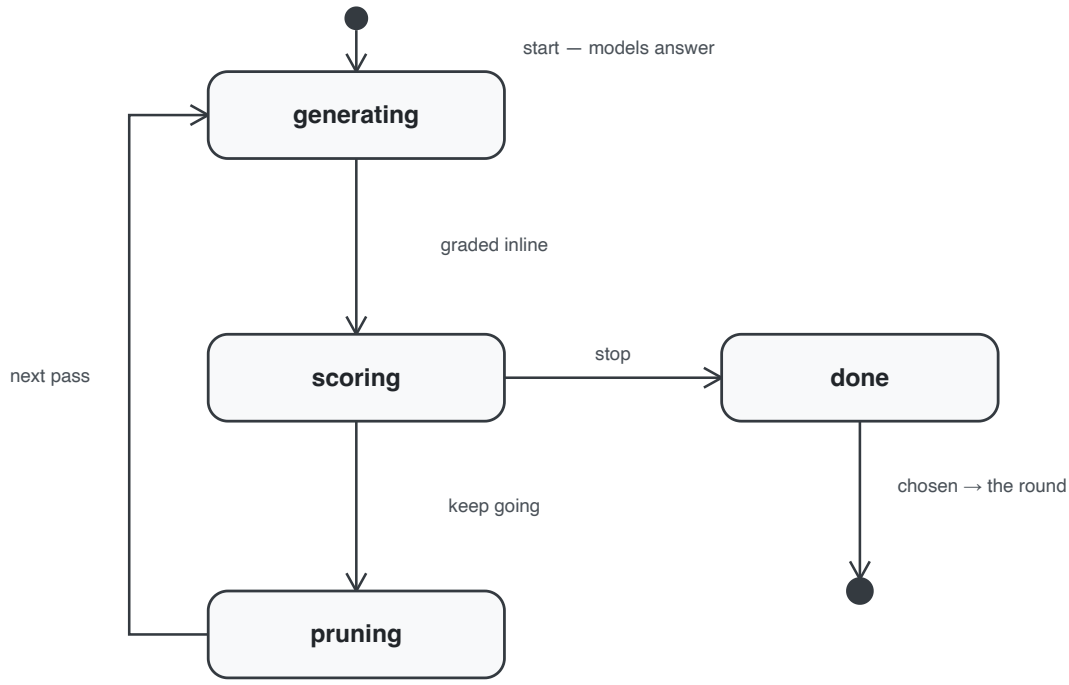
What happens at run time when the combine method is `moe` — match, and only ask the judge on a tie.



Routing has no screen of its own; it runs on the server, once per question, before anything is generated. The free step compares the question with each model's one-sentence `expertise` in the catalog. Only when two or more models come out too close to call does the study's trusted judge get asked — at temperature 0, so the same question routes the same way every time — and a tie the judge cannot settle falls to whichever model carries the highest `weight`, rather than to chance. The decision is written to `chat_round.moe_routing`, so a run can report which model answered what, and how often the tie-break fired.

4 · The Self-Improvement Loop

What happens at run time when self-improvement is on — improve → score → stop.



Stops when the score reaches score_target, OR a pass gains less than min_score_gain, OR it hits max_iterations (Mixture-of-Agents: the best answer is fed back each pass). Pruning drops any generator below underperform_threshold; each pass is saved as a chat_round_candidate, and the winner becomes chat_round.response.

This loop has no screen of its own; it runs on the server at chat time. Its settings are chosen on the **Model-configurations Screen** (`model-configurations.html`), and its passes surface on the **See-all Screen**. The loop stops on `score_target` , or on a gain below `min_score_gain` , or on `max_iterations` . Under `moe` the routed model refines its own answer; the router is not re-run between passes.

5 · In Plain English

A model set-up is a saved recipe for how the assistant answers: which AI model or models to use, and how to turn several answers into one. You build it once and reuse it across many studies and runs, just like a cohort.

The easy case is a single model: pick one, choose “just use it,” and you are done. If you want more, you can add several models and choose how to settle on one answer. You can merge them, take a weighted vote among them, pick the answer they most agree on, or let the study's one trusted judge pick the best. The judge lives on the study, not here, so the same impartial grader settles every set-up the same way.

There is one more choice, and it works the other way round. All of those run every model and then settle on one answer, which means you pay for every model on every question. **Mixture of experts** instead picks the right model *first* and asks only that one, so a set-up with five models still costs one answer. The next diagram shows how it decides.

You can also switch on self-improvement. The models take another pass at their own answers, keep the best, and stop when the answer is good enough, stops getting better, or hits a pass limit. Weak models get dropped along the way. The last diagram shows that loop.

Fits the schema cleanly. The set-up is `model_config`; each member is a `model_config_model`; the loop's stop-conditions are the `score_target` / `min_score_gain` / `max_iterations` fields. Routing adds no table of its own: it matches on `model_catalog.expertise`, tiers by the member's existing `weight`, and records what it did in `chat_round.moe_routing`.