Model Architecture Dispatch and Loader Boundaries

Deep Dive into SGLang — Chapter 13

Yin Yang

Foundation Books Project

Where this chapter sits

A served checkpoint never hands ModelRunner a Python class. It advertises architecture strings.

- In our running trace, `google/gemma-4-E4B-it` advertises `Gemma4ForCausalLM` in its text-tower config.
- Before any request row can execute, SGLang must resolve exactly *one* runtime class.
- The operational question is ownership: when a launch fails or surprises you, which subsystem should explain it?

Goal: treat the registry as a load-time dispatch boundary — external names enter, one resolved identity leaves, or the launch fails before tensors are blamed.

Roadmap

The dispatch problem

The four boundaries

Why it stays honest

The dispatch problem

Names in, one runtime class out

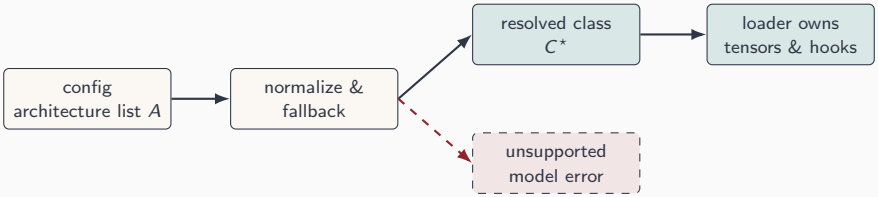
- A checkpoint's `architectures` list is *external evidence*, not an executable class.
- Resolution is the runtime decision that turns that evidence into one supported model class — or an early unsupported-model error.
- Other surfaces (logits, weight-loading hooks, capabilities) attach *after* the class is chosen.

The chapter's single handle is the resolved triple:

$$\rho(A, R_{\text{cls}}; I, F, M) = (a^\star, C^\star, I^\star).$$

$a^\star$: surviving name. $C^\star$: the class that owns forward and cache. $I^\star$: native, Transformers, or a forced wrapper.

The boundary, drawn once



Registry dispatch chooses the class. Weight loading, file formats, and quantization start only *after* a class contract exists — which is why a loader error and a dispatch error are different diagnoses.

The four boundaries

A diagnostic ladder, not a feature list

The same checkpoint directory can fail in four places. The *order* is fixed:

1. **Dispatch** — no supported architecture; no class was returned.
2. **Fallback policy** — a secondary or forced implementation was selected ($I^\star \neq$ native).
3. **Loader** — tensor, shard, or file-format interpretation failed *after* a class existed.
4. **Runner capability** — the loaded class advertises attention / modality / graph facts ModelRunner did not expect.

The useful question is never "did the checkpoint say Gemma?" — it is "which boundary first had enough evidence to reject the launch?"

The registry is process-local, not the file tree

- SGLang *imports* model modules and collects each module's `EntryClass` into `ModelRegistry.models`.
- A file on disk is only a *possible* support claim; an imported `EntryClass` is the surface resolution can use.
- `SGLANG_DISABLED_MODEL_ARCHS` can remove a class before startup — the checkpoint string is unchanged, but this process no longer supports it.

Two servers from the same source tree can legitimately support different architectures. Support is what imported here, now.

From concept to code: names in, one class out

The whole dispatch decision is one short function.

from `python/sglang/srt/model_loader/utils.py`

```
architectures = getattr(model_config.hf_config, "architectures", [])
...
supported_archs = ModelRegistry.get_supported_archs()
is_native_supported = any(arch in supported_archs for arch in architectures)
if model_config.model_impl == ModelImpl.MINDSPORE:
    architectures = ["MindSporeForCausalLM"]
elif not is_native_supported or model_config.model_impl == ModelImpl.TRANSFORMERS:
    architectures = resolve_transformers_arch(model_config, architectures)
model_cls, resolved_arch = ModelRegistry.resolve_model_cls(architectures)
setattr(model_config, "_resolved_model_arch", resolved_arch)
```

`get_model_architecture` reads the names, applies policy, resolves one class, and caches `_resolved_model_arch` — the operational form of $(a^\star, C^\star, I^\star)$.

Why it stays honest

The invariant the rest of the chapter checks

Architecture-dispatch alignment
Dispatch is valid only when $\rho(A, R_{\text{cls}}; I, F, M) = (a^\star, C^\star, I^\star)$ identifies *one* loaded model contract, and every surface below preserves it — never silently switching architecture, class, or implementation.

- The raw list A may be rewritten by fallback policy.
- The cached triple may *not* be relabeled by later code.
- The scheduler reads the resolved surface (via `get_resolved_model_impl`), not the loader policy that produced it.

Evidence: identity, not lookup latency

- Registry *import* is a process-start cost (one measured run: ≈ 1.59 s, 134 supported entries) — separate from any one checkpoint's weight load.
- Warmed resolution finishes at microsecond scale: lookup is *not* a launch bottleneck.
- A `dummy` loader and an `auto` loader can share a resolved identity while taking different load paths.

The actionable artifact is the identity carried across boundaries: resolved architecture, resolved implementation, selected loader, load format — not a stopwatch on the dictionary lookup.

What to carry forward

1. A checkpoint advertises **names**; the registry resolves exactly **one runtime class** or fails early.
2. **Construction precedes lookup**: support is the imported `EntryClass` surface in this process.
3. Diagnosis follows a fixed ladder: **dispatch, fallback, loader, runner**.
4. The durable object is the resolved triple $(a^\star, C^\star, I^\star)$, plus the **selected-class loader contract** that keeps it stable.
5. Every concept has a **home in the SGLang source** — that link is the spine of this book.

Next: Section 1 states the dispatch problem precisely — and why one hard-coded model class could never serve the model zoo.