A FIELD GUIDE TO MODEL CONTROL

Where can you *change* an AI?

Skills steer a model from above — through context, instructions and tools. Open weights let builders reach below the interface — into generation, parameters and, sometimes, training. A guide to what changes, what stays fixed, and when each approach makes sense. No maths degree required.

Published free to read and free to quote · theontology.ai/open-vs-closed-ai September 2026

CONTENTS

01 **The short version** — Working conditions vs machinery

02 **Fix the vocabulary** — Closed, open weights, open source — three labels, not two

03 **The control stack** — One AI system, four places to intervene

04 **Working above the model** — What a skill is and what it can — and cannot — change

05 **Working below the interface** — What open weights add: the generation layer, made concrete

06 **Choosing the layer** — A decision framework — intervention, not ideology

07 **The idea to keep** — Takeaway and primary references

WHO THIS IS FOR

Anyone deciding how much control over an AI system their organisation actually needs — and at what layer to buy it. It assumes no machine-learning background. The examples are deliberately small; the distinctions are the point.

01 The short version

A skill changes the model's *working conditions.*
Open weights let you change more of the *machinery.*

Picture an excellent chef in someone else's kitchen. You can hand them a recipe, point out the ingredients and let them use the oven. That is the role of a **skill**: it gives a capable model a method, relevant material and approved ways to act.

If you own the kitchen, you can also recalibrate the oven, replace the equipment or rebuild the room. That is the extra control that comes with **open weights** and a self-managed model stack.

HIGHER-LEVEL CHANGE

Skills change how the system works on a task. They add instructions, examples, scripts, references and access to tools. They are quick to edit and work with both open and closed models. The weights stay fixed.

LOWER-LEVEL CHANGE

Open weights expand how the model itself can be run. Builders can host the parameters, inspect intermediate scores, add decoding rules, fine-tune behaviour and choose the infrastructure around the model. More of the stack is yours.

A USEFUL NUANCE

"Closed" and "open" are not two neat boxes. Providers expose different controls, and downloadable weights can still carry restrictive licences. Think **spectrum**, not switch. And "lower" means closer to the machinery — not automatically better.

02 Fix the vocabulary

The everyday phrase "open model" hides an important difference between having the weights and having the whole recipe used to produce them. Three labels are routinely collapsed into two.

01

Closed model

You send input to a service and receive output. The provider owns the model runtime and chooses which controls the API exposes.

- Fast to adopt
- Little infrastructure to run
- Internals remain behind the service boundary

02

Open weights

The learned parameters can be downloaded and run. You control the inference stack, within the licence and the limits of the material released.

- Host on your infrastructure
- Control generation code
- Not necessarily open source

03

Open-source AI

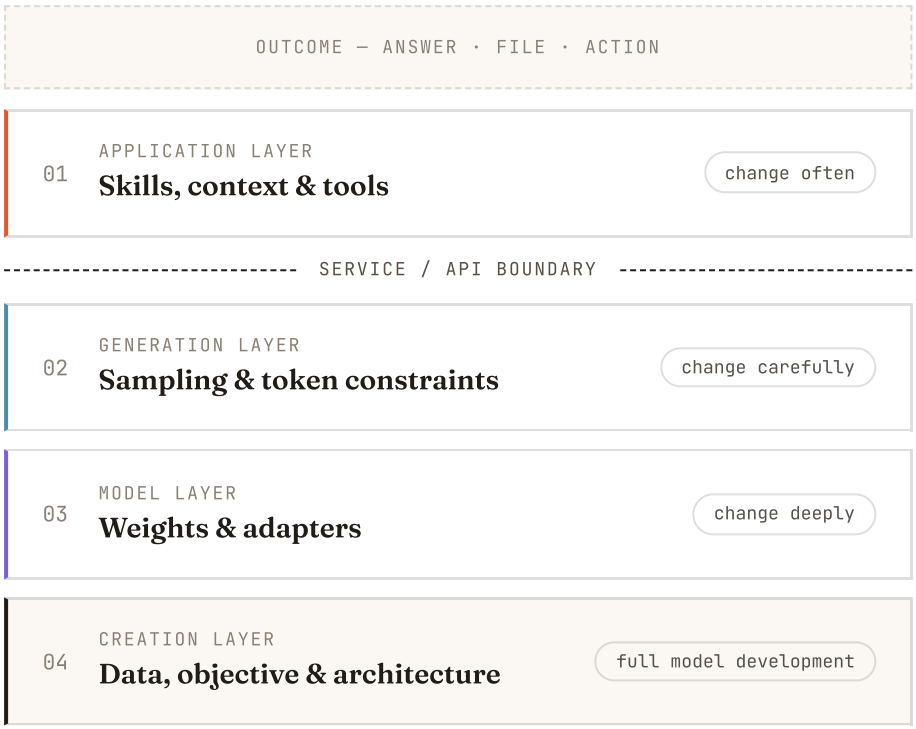
A stronger claim: freedoms to use, study, modify and share, plus access to the preferred form for modification — not just a checkpoint.

- Weights and model code
- Training and data information
- Terms that preserve the freedoms

The distinction follows the Open Source Initiative's *Open Source AI Definition 1.0* — reference 1, §07.

03 The control stack

One AI system, four places to intervene. The dashed line is where most closed services stop: everything above it can be changed by anyone building on the API; everything below it needs the provider's co-operation — or the weights.

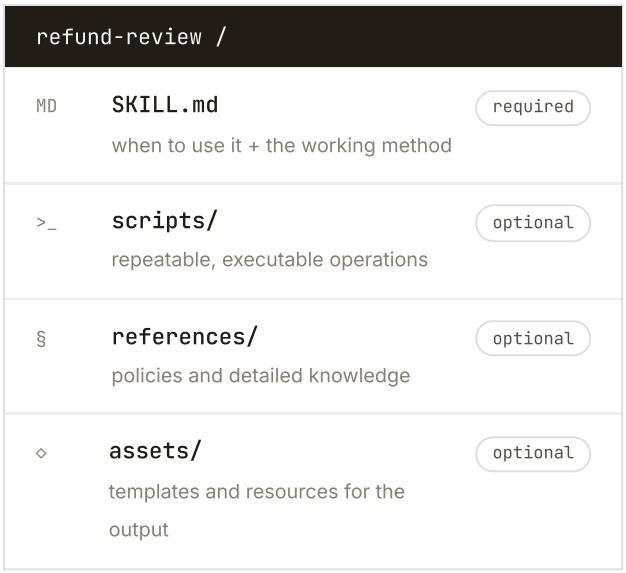


LAYER	WHAT A BUILDER CHANGES THERE	GOOD FIT	BOUNDARY
Application	Procedure, supporting material and approved tools for a task; the model reasons with that temporary context.	Procedures, current knowledge, integrations, repeatable work.	The model weights remain unchanged.
Generation	How the next token is chosen: sampling settings, biases on token scores, grammar constraints on output shape.	Hard output rules; formats that must always parse.	Needs self-hosting or an API that exposes these controls.
Model	The learned parameters themselves — fine-tuning or adapters that shift behaviour across all prompts.	Persistent behaviour change; domain style and tone.	Needs the weights, or a provider's tuning service.
Creation	Training data, objective and architecture — building or re-training the model.	Research and full model development.	Frontier-scale data, compute and expertise.

Open weights mostly expand the lower half of this picture. Skills live in the top half — and remain useful no matter which model sits underneath.

04 Working above the model

A skill is a reusable way of working. It packages the know-how an agent needs for a particular kind of task, then loads that know-how when it is relevant.



A SMALL PACKAGE, LOADED PROGRESSIVELY

The model does not need every manual in its head all the time.

An agent first sees a short catalogue of skill names and descriptions. When the task matches, it loads the relevant instructions and pulls in deeper references or scripts only as needed. What changes: context, procedure and available actions. What does not: the model's learned parameters.

01
Discover
The task matches a skill's description.

02
Load
The method and relevant references enter context.

03
Work
The model follows the method and calls approved tools.

04
Act
Files, systems or drafts change within permissions.

THE IMPORTANT BOUNDARY

A skill can change the world around the model. It may create a file, update a ticket or run a calculation, because the agent has tools and permission to do so. But the skill is not secretly retraining the model. It changes what the model sees, the method it follows and the actions it can request *for this run*.

The folder anatomy and progressive-loading pattern follow the Agent Skills specification — reference 2, §07.

05 Working below the interface

When you can run the model yourself, the score for every possible next token is available inside your own process. That creates intervention points that no prompt can reach. A worked toy example makes the difference concrete.

THE SETUP

A model completes the phrase `"The cat in the …"` over a toy four-token vocabulary. Every candidate token receives a score — a **logit** — and softmax turns those scores into probabilities. The three columns show the same model under three different interventions.

TOKEN	BASELINE — BASE WEIGHTS	SKILL CONTEXT — CHANGE THE INPUT	RUNTIME BIAS — CHANGE THE SCORES
hat	73.1%	31.0%	9.0%
mat	14.8%	55.4%	82.6%
box	8.1%	9.2%	5.9%
car	4.0%	4.4%	2.5%

Illustrative values, not a benchmark. Baseline: the learned association wins — "hat". Skill context: a rhyme guide loaded into context shifts the distribution toward "mat" by changing what the model reads. Runtime bias: a logits processor adds a bias to the scores directly — an intervention only possible when you control the generation code.

WHAT THIS BUYS IN PRACTICE

Hosting choice (privacy, residency, latency), inspectable intermediate scores, decoding constraints that make invalid output impossible rather than discouraged, fine-tuning and adapters, and freedom to swap any part of the serving stack.

KEEP THE CLAIM HONEST

Lower-level control is stronger, but it is not magic. Token constraints can guarantee a syntactically valid shape; they cannot guarantee the content is true or the decision wise. Sampling can remain nondeterministic. And some closed services expose structured-output or token-bias features of their own.

A concrete example of runtime-level control is a logits processor in Hugging Face Transformers, which receives token scores and returns processed scores — reference 3, §07.

06 Choosing the layer

Choose the intervention, not the ideology: use the highest layer that reliably solves the problem. Higher layers are faster to change and easier to inspect. Lower layers offer deeper control, but usually demand more data, evaluation, infrastructure and care.

NEED	BEST FIRST MOVE	WHY	MODEL ACCESS REQUIRED
Teach a repeatable procedure	Skill	Readable, versionable and quick to update	Open or closed
Use live systems or change files	Skill + tools	Puts action behind explicit permissions	Open or closed
Guarantee an output grammar	Decoding constraint	Prevents invalid tokens during generation	Self-hosted or supported API
Shift persistent model behaviour	Fine-tune / adapter	Changes learned behaviour across prompts	Weights or provider service
Own, inspect and replace the runtime	Open weights	Moves infrastructure and generation under your control	Downloadable parameters

Ask 01

Does this knowledge change often?

Keep changing policies and procedures in context, references or tools — not baked into weights.

Ask 02

Must the output obey a hard rule?

Use validation or constrained decoding. A beautifully written instruction is still an instruction.

Ask 03

Do you need to own the boundary?

Open weights matter when deployment, privacy, latency or deep inference control must be yours.

07 The idea to keep

Skills are editable *working methods.*
Open weights are editable *machinery.*

They are not rivals. A strong AI system often uses both: a controllable model underneath, and clear, auditable skills above it that connect intelligence to real work.

Primary references

01 Open Source Initiative — Open Source AI Definition 1.0
`opensource.org/ai/open-source-ai-definition`

02 Agent Skills — format specification
`agentskills.io/specification`

03 Hugging Face Transformers — generation utilities
`huggingface.co/docs/transformers/internal/generation_utils`

theontology.ai/open-vs-closed-ai · questions and arguments welcome: hello@theontology.ai September 2026