Cerebellum: Continuous Distillation of Frontier Models for Production Tasks

Sahil Mahendrakar Suhaas Katikaneni

September 2026

Abstract. Production applications increasingly route narrow, repetitive tasks through frontier language models, paying frontier-scale cost and latency for work that a far smaller model could perform equally well. Cerebellum is a system that continuously turns a customer's frontier model traffic into small, task-specific models that run at a fraction of the cost and latency while matching the frontier model's quality on that task. It consists of four parts. Data ingestion collects representative requests together with the frontier model's responses, either from an uploaded dataset or directly from production traffic. Distillation trains several candidate open-weight models of different sizes to reproduce the frontier model's outputs. Evaluation scores every candidate on a held-out set for agreement with the frontier model, cost, and latency, so the customer chooses a model on evidence. Deployment places the chosen model behind an endpoint that accepts the customer's existing API calls, with shadow mode, gradual rollout, and instant rollback. Once live, the model's own traffic flows back into the dataset, and candidates are retrained, re-evaluated, and redeployed as the task evolves. The result is a model that stays cheaper and faster than the frontier model at equal or better quality, without the customer managing training, evaluation, or serving.

1. Introduction

Cerebellum is a system that provides continuous evaluation, distillation, and deployment of custom models that run cheaper and faster than frontier models at similar or better performance. There are four main pieces to Cerebellum: data ingestion, distillation, evaluation, deployment. Figure 1 shows how the four parts connect. The remainder of this paper describes each part in turn: how data is collected and formatted, how candidate models are distilled, how they are evaluated against the frontier model, and how a chosen model is deployed, monitored, and improved over time.

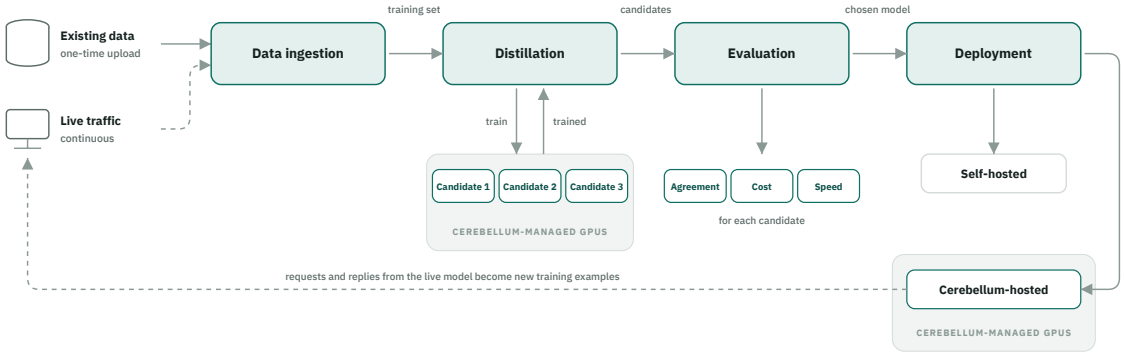


Figure 1: How the four parts of Cerebellum connect. Shaded boxes are the four stages Cerebellum runs; grey panels are Cerebellum-managed GPUs. Dashed arrows are the continuous path, which comes after the pilot: in the pilot, data arrives once as an upload.

2. Data ingestion

In order to distill custom models and run evaluations, Cerebellum needs a dataset of representative inputs and high-quality outputs for a specific task. Such a dataset can come from several sources: existing production traffic with PII redacted, a synthetically generated dataset, or a handcrafted and hand-labeled dataset. These sources differ only in who authored the inputs and outputs, but they share a single format described below. Production data is preferred because it reflects the true distribution of inputs the model will see. Synthetic data is useful for bootstrapping a task that has no production history yet.

2.1. Ingestion paths

There are two ways customers can provide data:

If a customer has historical data available, they can upload it directly to Cerebellum. If a customer does not have historical data available, Cerebellum can ingest data directly from their production workflows through a simple endpoint change at the model invocation site. We call this continuous ingestion. Under continuous ingestion, requests are forwarded to the frontier model, and both the request and the response are captured. The same endpoint later serves as the point at which a distilled model can be deployed and monitored. Continuous ingestion also allows PII redaction to run client-side, so that sensitive data never leaves the customer's environment.

For the pilot, we will do one-time, direct ingestion. The customer will upload a synthetic or existing production dataset directly.

2.2. Dataset format

Each example in the dataset is a single model invocation: the request as sent to the model API, and the response it returned. The request consists of the system prompt, the message history, and the tools made available to the model. The response is the model's output. This is the same shape as the customer's existing API calls, so ingestion requires no transformation, and it preserves everything the distilled model must be trained on in order to behave like the teacher in production.

The dataset is a JSONL file with one example per line. The input is the whole conversation up to that point. Output is the next assistant turn. For example, a support ticket triage task might contain examples such as:

```
{
  "input": {
    "system": "You are a support triage assistant. Classify the ticket and respond with JSON.",
    "messages": [
      {"role": "user", "content": "I was charged twice this month. Can you fix it?"}
    ],
    "tools": []
  },
  "output": {
    "content": "{\"category\": \"billing\", \"priority\": \"high\"}"
  }
}
```

2.3. Train and held-out split

The dataset is split into a training subset and a held-out subset using a shuffled 80/20 split. The training set is used to distill the custom models, and the held-out set is used to run evaluations.

A good dataset size depends on how narrow the task is and how complex its outputs are. As a rough guideline, a held-out set of at least 300 examples is needed to distinguish models reliably. With an 80/20 training split, a total dataset of roughly 1,500 to 2,000 examples should be sufficient. If a customer has fewer examples than this, synthetic generation can be used to increase the dataset size.

3. Distillation

Once a task has a dataset, Cerebellum distills the behavior of the frontier model into one or more small open weight models of varying sizes. The teacher's outputs in the training set already demonstrate how the task should be performed. Distillation teaches a small model to reproduce those outputs. The result is a set of candidate models that behave like the teacher on the customer's task, but are a fraction of its size, and therefore cheaper and faster to run.

3.1. How distillation works

Cerebellum uses supervised fine-tuning. A small model that has already been pre-trained is shown every example in the training set, and its weights are adjusted so that it becomes more likely to produce the teacher's output. Fine-tuning is performed with LoRA (low-rank adaptation). This leaves the base model's weights frozen and trains a small set of additional weights alongside them. For each task, Cerebellum trains multiple candidate models of different sizes, across one or many open-weight model families.

Cerebellum manages the training configuration. Learning rate, training length, and the other settings are chosen per model size, so the customer does not need to tune anything.

3.2. Where training runs

Training jobs run on Cerebellum-managed cloud GPUs. Each job is self-contained, and candidates are trained in parallel, so distillation typically finishes in less than a couple of hours for a pilot-sized dataset. The customer's data is used only for their own models, and the resulting weights belong to the customer.

For customers with stricter privacy or compliance requirements, the same training jobs can be run inside the customer's own cloud account, though this is beyond the scope of the pilot.

4. Evaluation

Once the candidate models have been trained, Cerebellum evaluates every one of them against the held-out set. Each candidate is sent the same input the teacher was sent, and its outputs are compared to the teacher's. The result for every candidate is a single quality score alongside its cost and latency, which is what the customer can use to choose their model.

4.1. What is measured?

The primary quality metric for the pilot is teacher agreement. Teacher agreement measures the percentage of held-out examples on which the candidate's output matches the teacher. This represents how faithfully a candidate reproduces the frontier model's behavior on the specific task.

What counts as a match depends on the shape of the teacher's outputs, which Cerebellum determines automatically by inspecting the held-out set. Cerebellum uses two evaluators: code-based evaluators and LLM-as-a-judge evaluators.

Code-based evaluators are used when the teacher's outputs are structured outputs or tool calls. The code-based evaluator checks that the candidate's output matches the teacher's exactly, field by field.

LLM-as-a-judge evaluators are used when the teacher's outputs are unstructured text. A frontier model acts as a judge and is shown the teacher's output and the candidate's output, and determines if they are equivalent along with a reason.

Many real outputs mix the two, for example a JSON object with a free-text field. In that case the structured fields are compared exactly and the judge is applied to the free-text field, and the example succeeds only if both agree. The customer can view examples where the distilled model disagreed with the frontier model to gain a better understanding of where and why the models disagreed.

A high teacher agreement score gives the customer confidence that their smaller, custom model will perform at a similar level of quality as their existing frontier model.

Cost and latency of each model are also measured during the same evaluation run. This allows the customer to compare teacher agreement score, cost, and latency across each candidate.

5. Deployment

After evaluation, the customer has a quality score, a cost estimate, and latency for every candidate model. Deployment is where the customer picks one of these custom models and uses it for their real traffic. Cerebellum is designed so that this step is small and reversible, and can be rolled back easily if performance degrades.

5.1. Self-hosting versus Cerebellum-hosted

The customer owns the weights of every candidate model Cerebellum trains for them, and can run a deployed model in either of two ways.

Self-hosted. At any point, the customer can export a model as a single open-weight model that runs on any common serving stack, including the customer's own cloud account. If this option is chosen, the customer is completely responsible for serving the model and monitoring its quality and health in production.

Cerebellum-hosted. Cerebellum can handle serving and monitoring the deployed model in production. Deploying a candidate produces an endpoint and an API key. The endpoint accepts requests in the same format the customer already uses to call the frontier model. Switching to the distilled model is simply a one-line endpoint change. Behind the endpoint, Cerebellum runs the model on Cerebellum-managed GPUs, and the customer never provisions or manages hardware. Cerebellum will also manage the rollout, monitoring, and

continuous evaluation of the model.

5.2. Rolling out safely

Cerebellum can perform a safe and gradual rollout to the distilled model so the customer can catch problems before they reach all of their users. The following features ensure a safe rollout:

Shadow mode. The frontier model answers every request as before. In the background, the distilled model also answers each one, and its output is stored and compared to the frontier model's rather than returned. No user is affected, and the customer sees how the candidate would have performed on live traffic.

Rollout. The distilled model answers a percentage of requests and the frontier model continues to answer the rest of traffic. The customer can monitor for any issues without their users being impacted.

Rollback. At any moment, all requests can be redirected to the frontier model, and nothing needs to be redeployed or restarted.

5.3. Monitoring

Once a model is serving traffic, Cerebellum continuously monitors the quality and health of the model.

Quality is measured as live agreement with the frontier model. A small random sample of live requests is also sent to the frontier model, and each pair is scored with the same evaluators used in evaluation. This allows the customer to ensure their distilled model is still performing well compared to the frontier model.

Health is tracked with automated checks that run on all traffic. Every model response is checked that it parses, that the fields the task requires are present, and that any tool call names a real tool. Alongside them, Cerebellum tracks latency, reported as the median and 95th percentile, and cost to date.

5.4. Continuous improvement

When live agreement with the frontier model starts to drop, or once enough new examples have come in, the candidate models are retrained on the larger dataset and evaluated again. The new candidates can be compared to the deployed model and frontier models, and the customer can move to one of them using the same rollout strategy described above. This ensures the customer always has the best performing model for their task.

Taken together, the four parts form a loop rather than a pipeline. Production traffic grows the dataset, new candidates are distilled from it, evaluation ranks them against the frontier model, and the best one takes over serving through the same rollout steps as before. This loop is what makes Cerebellum continuous. The dataset grows, the candidates improve, and the deployed model is replaced whenever a better one exists.