

Lab 6.3 Run gpt-oss-20b on AWS

Gavin · AI-assisted course design

Contents

1	Goal	3
2	Background	3
3	Prerequisites	3
4	Workflow	4
5	Tasks	4
5.1	Task 1 — Choose and price the machine	4
5.1.1	A concrete starting configuration	4
5.1.2	Quota and capacity are different checks	5
5.1.3	Price the whole session	5
5.2	Task 2 — Launch, connect, and inspect	6
5.3	Task 3 — Install the pinned runtime	8
5.4	Task 4 — Download exactly one model file	9
5.5	Task 5 — Load the GPU model and send a request	10
5.5.1	Record one warm-up and two observations	11
5.6	Task 6 — Read the evidence, then close the session	14
5.6.1	What do the measurements mean?	14
5.6.2	Continue immediately or finish	14
5.6.3	Export before termination	14
6	Troubleshooting without enlarging the experiment	15
7	Optional extension — Plan for gpt-oss-120b	16
8	What did we just do?	16
9	More Learning	17

1 Goal

Get a real OpenAI open-weight model running on one rented GPU. Send a small, bounded request, preserve what actually happened, and remove the resources when you finish.

2 Background

In Lab 4.1, a tiny model made inference mechanics inexpensive to inspect. Here the same ideas become operational decisions: how checkpoint format, runtime, memory, context length, and hardware affect whether a much larger model can run.

OpenAI describes gpt-oss-20b as approximately 21 billion total parameters with 3.6 billion active per token. Its MXFP4 MoE weights make a small-memory inference path possible. Active parameters describe computation; they do not mean that only the active experts need to be stored. OpenAI also requires the model's Harmony conversation format. We will use an inference engine that applies that format, rather than concatenate arbitrary role labels. See the OpenAI model card and reference implementations.

This lab uses **llama.cpp with its maintainers' GGUF conversion**. GGUF is the runtime's model container. This is a converted representation of OpenAI's checkpoint, not an OpenAI-hosted API call. The runtime maintainer's gpt-oss guide documents this route and its memory considerations.

! Inference fit is not instrumentation fit

OpenAI's "within 16GB" statement concerns supported quantized inference. It does not promise that every backend, context size, activation cache, or routing recorder fits there. This lab captures outputs and resource observations. It does not attach the small-model microscope's hooks or claim to capture expert routing. Activation or routing research needs a separate instrumented-runtime and memory plan; do not silently rent a larger instance when it fails.

3 Prerequisites

You need the AWS workbench record from Lab 1.4, Lesson 4.1: Inference, and Lab 4.1: Measure Inference and Sampling. Use the AWS account, identity, Region, billing checks, and connection method you established there. Account creation and payment setup are not repeated here.

Estimated time: 90–120 minutes, including setup, download, evidence export, and cleanup. Quota approval may take longer; resolve that before starting the paid session.

Deliverable: a verified model/runtime manifest, three recorded inference attempts, hardware and memory observations, a cost worksheet, and cleanup evidence. A failed or blocked attempt is a valid report, but it is not a completed model run.

i Validation status

The reference below was checked against publisher documentation and pinned source interfaces on 9 October 2026. It has not been run on AWS or validated with the 20B weights by the course author. The hardware choice is a source-supported candidate, not a measured fit or performance guarantee. Preserve failures and changes instead of treating these instructions as benchmark results.

4 Workflow

1. Choose a single GPU and price the complete session.
2. Launch the agreed machine and check its actual hardware.
3. Build a pinned runtime and verify one pinned model file.
4. Run three short requests through a loopback-only endpoint.
5. Export the evidence, then terminate and check remaining charges.

Before launching, predict: will the downloaded file size equal GPU memory use? Will the first request have the same latency as the next two? Can a reasoning model exhaust its output allowance before producing a final answer?

5 Tasks

5.1 Task 1 — Choose and price the machine

5.1.1 A concrete starting configuration

Component	Reference choice	Why it is here
EC2 purchase option	One Linux On-Demand instance, default/shared tenancy	One short session without a reservation or recurring commitment
Instance	g5.2xlarge: 8 vCPUs, 32 GiB host RAM, one NVIDIA A10G	More host-memory headroom than the 16 GiB g5.xlarge; only one GPU
GPU memory	AWS markets A10G as 24 GB; its instance specification lists approximately 22 GiB	GB and GiB differ; check the real free MiB with <code>nvidia-smi</code>
Image	AWS Deep Learning Base OSS Nvidia Driver GPU AMI (Ubuntu 24.04) , x86_64	Supplies an NVIDIA driver and CUDA toolkit; no framework installation is needed
Storage	One encrypted gp3 root volume, 100 GiB if the selected image permits it	OS/toolkit, build, 12.11 GB model, logs, and spare capacity

Component	Reference choice	Why it is here
Network	Existing public subnet, temporary public IPv4, SSH only from the Region's EC2 Instance Connect prefix list	Browser terminal access; no public model service

Hardware sources: AWS G5 and accelerated-instance specifications. The 20260828 Base OSS image release lists G5 support and CUDA 12.8 among its installed toolkits. Use an AWS-published, supported image; record its exact regional AMI ID, name, release, architecture, and any difference from this reference before approval. Do not select a lookalike community image or add a paid Marketplace software subscription.

The image's minimum root-volume size may exceed 100 GiB. Check that in the launch preview and price the actual size. Do not shrink below the image minimum or automatically add another volume. Reserve **at least 30 GiB free after the runtime build**, before downloading. The one model file is 12,109,566,624 bytes, about 11.28 GiB. Downloading directly to a .part file and renaming it avoids an extra model copy or Hub cache. The 30 GiB free-space floor is a lab planning margin, not a measured installation requirement.

A g6.2xlarge with one L4 and 32 GiB host RAM is a possible separately checked alternative. It needs its own current price, image compatibility, and GPU verification. It is not an automatic fallback. This lab does not require multi-GPU P-series instances.

5.1.2 Quota and capacity are different checks

In the selected Region:

- Open **Service Quotas** → **Amazon EC2** → **Running On-Demand G and VT instances**. This limit counts vCPUs. The proposed instance needs 8 vCPUs in addition to existing use. A new account can have a zero quota. Request only the increase you need and wait before launching anything.
- In **EC2** → **Instance Types**, confirm that the chosen type is offered in the Region and intended Availability Zone. An offering and a sufficient quota do not reserve physical capacity.
- If launch reports `InsufficientInstanceCapacity`, record it. Retry later or choose a separately priced and approved alternative. Do not create a Capacity Reservation, launch multiple machines, or escalate to a larger type to make the error disappear.

Sources: EC2 quotas, instance discovery, and capacity troubleshooting.

5.1.3 Price the whole session

Copy current rates from EC2 On-Demand pricing, EBS pricing, and public IPv4 pricing or the AWS Pricing Calculator. Save the Region, currency, pricing date, and estimate. Use the account's actual purchase eligibility; do not assume GPU compute is free or covered by credits.

Worksheet item	Enter before launch
Instance and image	Exact type, regional AMI ID, any software surcharge

Worksheet item	Enter before launch
Running time	Setup + build + download + inference + export + cleanup margin; maximum 2 hours for this session
Compute	Current hourly rate × planned running hours
EBS	Actual provisioned GiB and current storage rate, for the entire time the volume exists; include extra IOPS/throughput if selected
Networking	Public IPv4 allocation time, applicable outbound transfer, and any existing network charges attributable to this run
Other charges	Taxes, currency conversion, account-specific fees, retained snapshots or backups
Total and allowance	Estimated total, contingency, and the learner's explicit maximum spend
Finish	UTC deadline and who will export evidence and terminate the instance

For a rough monthly EBS-rate estimate, prorate storage by its lifetime, not just GPU-running time; use the calculator's current billing convention for the final estimate. Keep baseline gp3 performance, no new NAT gateway, no load balancer, no Elastic IP, and no snapshot for the reference path.

Write a short approval record: “One [type], [AMI], [Region], [volume size], for at most [duration], ending [UTC time], estimated [currency/amount], with [currency/amount] maximum spend and cleanup by [person].” If an assistant is helping, agree on those specific resources before it creates anything. Include Lab 6.4 in that same session only if you intend to do it immediately.

Warning

AWS Budgets notifications and a spending allowance are **not a hard billing cap**. Charges and alerts can lag. A stopped instance can still leave billable storage, addresses, or other resources. Keep the billing safeguards from Lab 1.4, set a personal deadline alarm, and actively finish the cleanup below.

5.2 Task 2 — Launch, connect, and inspect

Reuse the launch and access workflow from Lab 1.4. Before the paid launch, confirm the access policy permits `ec2-instance-connect:SendSSHPublicKey` for this GPU instance/tag with `ec2:osuser=ubuntu`, plus `ec2:DescribeInstances` for the console. A policy restricted to the earlier CPU instance or `ec2-user` is insufficient; have the account administrator adjust only the needed scope. See AWS EIC permissions.

The important launch changes and checks are:

- One approved GPU instance and AWS Ubuntu DLAMI, tagged `llmcourse-lab24`; verify type, image, quantity, and cost in the launch summary.
- The approved encrypted gp3 volume, with **Delete on termination**; instance-initiated shutdown behavior **Stop**; IMDSv2 required.
- The same restricted browser-SSH arrangement: TCP 22 from the Region's EC2 Instance Connect prefix list. Keep inference/Jupyter/web ports closed. Outbound package/model downloads are needed during setup.
- No administrator instance role or automatic replacement service. The model process needs no AWS access key, OpenAI API key, Hugging Face token, or private prompt.

Ubuntu uses username `ubuntu`, unlike the earlier Amazon Linux workbench. Confirm that the image supports EC2 Instance Connect. If it is not preinstalled, this optional **user data** installs the Ubuntu package. Its first line after the interpreter schedules a protective power-off; it contains no credentials.

```
#!/bin/bash
shutdown -P +110 "LLMCourse lab session deadline"
timeout 10m bash -c 'apt-get update && apt-get install -y ec2-instance-connect'
```

Use the same protective shutdown line even if installation is unnecessary. It is a best-effort stop approximately 110 minutes after boot, not a spending guarantee or evidence-export mechanism. A reboot or failed startup can invalidate assumptions about it. The manual deadline remains at most two hours from launch. If the approved window is shorter, shorten the timer too. AWS documents shutdown behavior.

After status checks pass, connect as `ubuntu` using the method from Lab 1.4. If you cannot connect within ten minutes, stop the instance from the console and investigate; do not leave it running while repairing access. Follow the EIC prerequisites rather than opening SSH to the world. Keep credentials and private keys out of chat, source files, and evidence.

In the instance terminal, create the working directories and reusable environment:

```
set -euo pipefail
# Fresh session only: preserve any earlier run instead of overwriting it.
test ! -e "$HOME/llmcourse-gpt-oss"
mkdir -p "$HOME/llmcourse-gpt-oss/evidence/lab24" "$HOME/llmcourse-gpt-oss/models"
cat > "$HOME/llmcourse-gpt-oss/env.sh" <<'SH'
export ROOT="$HOME/llmcourse-gpt-oss"
export RUN="$ROOT/evidence/lab24"
export MODEL="$ROOT/models/gpt-oss-20b-MXFP4.gguf"
export SERVER="$ROOT/llama.cpp/build/bin/llama-server"
export BASE_URL="http://127.0.0.1:8080"
SH
source "$HOME/llmcourse-gpt-oss/env.sh"
{
    date -u
```

```

uname -a
cat /etc/os-release
free -h
df -h "$ROOT"
nvidia-smi
/usr/local/cuda-12.8/bin/nvcc --version
} > "$RUN/hardware.txt" 2>&1
cat "$RUN/hardware.txt"
nvidia-smi --query-gpu=name,memory.total,memory.free,driver_version \
--format=csv > "$RUN/gpu-before.csv"
cat "$RUN/gpu-before.csv"

```

Check: one A10G, approximately 32 GiB host RAM, correct OS, functioning driver, and CUDA compiler. Require **at least 20,000 MiB actually free GPU memory** for this conservative starting plan. This floor leaves margin above the maintainer guide's example inference footprint, but does not prove fit. If another process occupies the GPU, do not kill an unfamiliar process. Investigate or end the session.

Record the instance ID, AMI ID, Region, volume IDs, launch time, deadline, and approved worksheet in your private lab record. Keep account identifiers out of a public course submission. If any hardware check fails, preserve the output and go to cleanup.

5.3 Task 3 — Install the pinned runtime

The source pin is llama.cpp v0.6.0, commit:

d81235049384534c167caea52b85a694f6103d14.

Use the pinned CUDA build instructions and server interface as the reference. The AMI and Ubuntu package versions are recorded alongside it; a source pin alone does not make every compiler and driver identical.

```

sudo apt-get update
sudo apt-get install -y --no-install-recommends \
    build-essential cmake git curl ca-certificates libssl-dev

git init "$ROOT/llama.cpp"
git -C "$ROOT/llama.cpp" remote add origin https://github.com/ggml-org/llama.cpp.
git
git -C "$ROOT/llama.cpp" fetch --depth 1 origin \
    d81235049384534c167caea52b85a694f6103d14
git -C "$ROOT/llama.cpp" checkout --detach FETCH_HEAD
test "$(git -C "$ROOT/llama.cpp" rev-parse HEAD)" = \
    d81235049384534c167caea52b85a694f6103d14

cmake -S "$ROOT/llama.cpp" -B "$ROOT/llama.cpp/build" \
    -DCMAKE_BUILD_TYPE=Release -DGGML_CUDA=ON \
    -DCMAKE_CUDA_COMPILER=/usr/local/cuda-12.8/bin/nvcc \
    -DLLAMA_BUILD_TESTS=OFF -DLLAMA_BUILD_EXAMPLES=OFF \
    -DLLAMA_BUILD_APP=OFF -DLLAMA_BUILD_UI=OFF -DLLAMA_USE_PREBUILT_UI=OFF \

```

```

-DLLAMA_BUILD_IS_DEV=OFF \
2>&1 | tee "$RUN/configure.log"
timeout --signal=TERM --kill-after=30s 20m \
  cmake --build "$ROOT/llama.cpp/build" --config Release -j 4 \
  --target llama-server 2>&1 | tee "$RUN/build.log"

"$SERVER" --version > "$RUN/runtime-version.txt" 2>&1
"$SERVER" --list-devices > "$RUN/runtime-devices.txt" 2>&1
"$SERVER" --help > "$RUN/runtime-help.txt" 2>&1
git -C "$ROOT/llama.cpp" rev-parse HEAD > "$RUN/runtime-commit.txt"
dpkg-query -W > "$RUN/os-packages.tsv"
cp "$ROOT/llama.cpp/LICENSE" "$RUN/runtime-LICENSE.txt"
cat "$RUN/runtime-devices.txt"

```

The CUDA device must appear. Stop on a build timeout, missing CUDA backend, or compiler error. Do not quietly run a CPU-only binary or switch to an unpinned installer. Check the remaining session time before spending it on debugging.

5.4 Task 4 — Download exactly one model file

Identity	Pinned value
Original model	openai/gpt-oss-20b, Apache 2.0
Conversion publisher	ggml-org/gpt-oss-20b-GGUF, the llama.cpp maintainers
Conversion repository revision	ef9b12f2ff56c69cf32153a02784e7a3c88bf524
Selected file	gpt-oss-20b-MXFP4.gguf
File size	12,109,566,624 bytes
SHA-256	27cd6c432c7672cb812a92f611cf3ba7 bbc35928262bb1e1253ff4ee6ae35901
Conversion's recorded primary source revision	6cee5e81ee83917806bbde320786a8fb61efebec

The pinned file pointer supplies the byte count and SHA-256. The conversion commit records provenance. Save the conversion README, source record, and log. This archive also contains speculative draft models; we will not download or use them.

```

python3 - <<'PY'
import os, shutil
free = shutil.disk_usage(os.environ["ROOT"]).free
assert free >= 30 * 1024**3, f"Only {free / 1024**3:.1f} GiB free; stop and review storage"
PY

export MODEL_REV=ef9b12f2ff56c69cf32153a02784e7a3c88bf524
export MODEL_BASE="https://huggingface.co/ggml-org/gpt-oss-20b-GGUF/resolve/$MODEL_REV"

```

```

for item in README.md .src_sha convert.log; do
    curl --fail --location --connect-timeout 30 --max-time 60 \
        --max-filesize 1048576 "$MODEL_BASE/$item" \
        --output "$RUN/model-${item#.}"
done

# A new download only. Do not overwrite evidence from a previous attempt.
test ! -e "$MODEL" && test ! -e "$MODEL.part"
curl --fail --location --connect-timeout 30 --max-time 1200 \
    --max-filesize 13000000000 \
    "$MODEL_BASE/gpt-oss-20b-MXFP4.gguf" --output "$MODEL.part"
test "$(stat -c %s "$MODEL.part")" = 12109566624
printf '%s %s\n' \
    27cd6c432c7672cb812a92f611cf3ba7bbc35928262bb1e1253ff4ee6ae35901 \
    "$MODEL.part" | sha256sum --check
mv "$MODEL.part" "$MODEL"
sha256sum "$MODEL" > "$RUN/model-sha256.txt"
printf '%s\n' "$MODEL_REV" > "$RUN/model-revision.txt"
stat -c '%n %s bytes' "$MODEL" > "$RUN/model-size.txt"

```

Do not use an unpinned main, pull the whole repository, convert a second copy, or add the 120B model. A failed size/hash check blocks loading. Keep the error and partial-download size, then remove the incomplete file only when you have decided whether to retry within the same allowance. Public download requires no Hugging Face login.

5.5 Task 5 — Load the GPU model and send a request

Prepare before starting the server clock. Read the request procedure below and save `smoke.py` first; do not execute it yet. If you plan to continue directly into Lab 6.4, read that Lab now, save its predictions and client, and check that its full 15-minute request window will fit within this same 30-minute server deadline after startup and the three smoke attempts. Keep evidence export and resource cleanup within the separate AWS session deadline. If the reprise will not fit, use its later-session route rather than extending either clock.

Start a 30-minute runtime window, which must fit inside the remaining approved EC2 session. This timer stops the model process; it does **not** stop EC2 billing. The explicit context, one slot, fixed batches, disabled prompt-cache reuse, and disabled auto-fitting make the starting workload inspectable.

```

source "$HOME/llmcourse-gpt-oss/env.sh"
if ss -ltn | grep -qE ":\:8080[[:space:]]"; then
    echo "Port 8080 is already in use; stop and inspect the existing process."
    exit 1
fi
nohup timeout --signal=TERM --kill-after=30s 30m \
    "$SERVER" --model "$MODEL" --alias gpt-oss-20b \
    --host 127.0.0.1 --port 8080 --offline \
    --ctx-size 4096 --parallel 1 --n-gpu-layers 99 --fit off \
    --batch-size 256 --ubatch-size 256 --flash-attn on \

```

```

--cache-type-k f16 --cache-type-v f16 \
--cache-ram 0 --no-cache-prompt --no-context-shift \
--jinja --reasoning-format deepseek --reasoning-effort low \
--n-predict 256 --no-warmup --perf \
> "$RUN/server.log" 2>&1 < /dev/null &
echo "$!" > "$RUN/server-supervisor.pid"

# Five-minute readiness deadline; at most five extra seconds to force termination.
if ! timeout --signal=TERM --kill-after=5s 5m bash -c '
    while kill -0 "$(cat "$RUN/server-supervisor.pid")" 2>/dev/null; do
        if curl --fail --silent --max-time 2 "$BASE_URL/health" > "$RUN/health.json";
        then
            exit 0
        fi
        sleep 5
    done
    exit 1
'; then
    kill -TERM "$(cat "$RUN/server-supervisor.pid")" 2>/dev/null || true
    tail -n 60 "$RUN/server.log"
    echo "Model did not become ready; preserve evidence and clean up."
    exit 1
fi
pgrep -P "$(cat "$RUN/server-supervisor.pid")" -x llama-server > "$RUN/server.pid"
ss -ltnp | grep ':8080' | tee "$RUN/listening.txt"
grep -Ei 'CUDA|offload|buffer|context|n_ctx' "$RUN/server.log" \
| tee "$RUN/placement.txt"
curl --fail --silent "$BASE_URL/props" > "$RUN/server-props.json"

```

Inspect the full log and placement.txt before generating. Confirm that the model's transformer layers and output layer were offloaded to CUDA, rather than interpreting "CUDA detected" as proof of GPU execution. Small CPU-side buffers and host memory mapping can still appear. Unexpected partial layer offload, CPU expert placement, insufficient-memory warnings, or OOM are failed preconditions for this reference run. Do not reduce GPU offload, add swap, or increase the instance automatically.

Check listening.txt: the endpoint must bind only to **127.0.0.1:8080**. It is used from inside the instance terminal. Leave the security group closed to port 8080; do not bind to 0.0.0.0. No agent tools, shell execution, external MCP servers, or browser tools are enabled.

5.5.1 Record one warm-up and two observations

Use one harmless prompt: In two sentences, explain why a GPU can speed up matrix multiplication. The following Python client uses the standard library; it requires no SDK key or pip install. Save it as \$ROOT/smoke.py:

```

import json
import os
from pathlib import Path

```

```

import signal
import time
import urllib.error
import urllib.request

run = Path(os.environ["RUN"])
base = os.environ["BASE_URL"]
assert base == "http://127.0.0.1:8080"
request = {
    "model": "gpt-oss-20b",
    "messages": [{"role": "user", "content":
        "In two sentences, explain why a GPU can speed up matrix
multiplication."}],
    "max_tokens": 128,
    "temperature": 0,
    "seed": 17,
    "reasoning_effort": "low",
    "cache_prompt": False,
    "repeat_penalty": 1.0,
    "stream": False,
}
(run / "request.json").write_text(json.dumps(request, indent=2) + "\n")

def deadline(_signum, _frame):
    raise TimeoutError("120-second request deadline")

signal.signal(signal.SIGALRM, deadline)
for label in ("warmup", "trial-1", "trial-2"):
    record = {"label": label, "status": "started", "request": request}
    (run / f"{label}.record.json").write_text(json.dumps(record, indent=2) + "\n")
    start = time.perf_counter()
    signal.alarm(120)
    try:
        req = urllib.request.Request(
            base + "/v1/chat/completions",
            data=json.dumps(request).encode(),
            headers={"Content-Type": "application/json"},
        )
        with urllib.request.urlopen(req, timeout=120) as response:
            raw = response.read(1_048_577)
            elapsed = time.perf_counter() - start
            signal.alarm(0)
            (run / f"{label}.response.json").write_bytes(raw)
            if len(raw) > 1_048_576:
                raise ValueError("Response exceeded the 1 MiB recording limit")
            result = json.loads(raw)
            choice = result["choices"][0]
            usage = result["usage"]

```

```

assert isinstance(usage["completion_tokens"], int)
record.update(status="received", request_wall_seconds=elapsed,
              usage=usage, finish_reason=choice.get("finish_reason"),
              timings=result.get("timings"))
message = choice["message"]
print(label, f"{elapsed:.3f} s", usage, choice.get("finish_reason"))
print("Final content:", message.get("content"))
except Exception as exc:
    signal.alarm(0)
    record.update(status="failed", elapsed_to_failure=time.perf_counter() -
start,
                error=f"{type(exc).__name__}: {exc}")
    if isinstance(exc, urllib.error.HTTPError):
        (run / f"{label}.error-body.txt").write_bytes(exc.read(65_536))
    # A client timeout alone need not cancel server computation.
    try:
        os.kill(int((run / "server-supervisor.pid").read_text()),
signal.SIGTERM)
    except ProcessLookupError:
        pass
    raise
finally:
    (run / f"{label}.record.json").write_text(json.dumps(record, indent=2) +
"\n")

```

Run a lightweight GPU sampler alongside those three calls:

```

nohup timeout 8m nvidia-smi \
  --query-gpu=timestamp,memory.used,memory.free,utilization.gpu \
  --format=csv --loop-ms=500 \
  > "$RUN/gpu-samples.csv" 2>&1 < /dev/null &
echo "$!" > "$RUN/gpu-sampler.pid"
ps -p "$(cat "$RUN/server.pid")" -o pid,rss,vsz,etime,comm > "$RUN/process-
before.txt"
if timeout --signal=TERM --kill-after=10s 7m python3 "$ROOT/smoke.py" \
  > "$RUN/client.log" 2>&1; then
  cat "$RUN/client.log"
else
  kill -TERM "$(cat "$RUN/server-supervisor.pid")" 2>/dev/null || true
  cat "$RUN/client.log"
  echo "Attempt failed; do not add more requests. Export what exists and clean
up."
fi
ps -p "$(cat "$RUN/server.pid")" -o pid,rss,vsz,etime,comm \
  > "$RUN/process-after.txt" || true
kill -TERM "$(cat "$RUN/gpu-sampler.pid")" 2>/dev/null || true
cp "$ROOT/smoke.py" "$RUN/smoke.py"

```

The request allowance is **three calls × 128 requested output tokens**, including reasoning and final-channel output. Retain actual counts: the pinned server documents that a partial multibyte character can slightly exceed a requested token limit. Normal end-of-sequence stopping remains enabled. A length finish reason or empty final content may mean the reasoning used the allowance; preserve that outcome instead of silently raising the limit. Raw responses retain any separately returned `reasoning_content`, but those traces are generated text, not a validated explanation of the computation.

5.6 Task 6 — Read the evidence, then close the session

5.6.1 What do the measurements mean?

For the two non-warm-up observations, record actual prompt/completion token counts, finish reason, final content, and request wall time. This wall time starts before the local HTTP request and ends when its complete response has been read. It includes local request handling and generation; it is **not time to first token** or isolated prefill time. Keep runtime-supplied timing fields with their names instead of relabelling them.

Report the range of observed GPU memory .used values and the largest sampled value. A 500 ms sampler can miss a short peak, and whole-device memory is not solely model-weight storage. Process RSS/VSZ from ps are in KiB on this Linux image; they are host-process observations, not GPU memory. Relate them to the accounting from Lesson 4.1: weights, KV cache, compute buffers, runtime, and temporary allocations. Do not substitute a file-size calculation for those observations.

With two observations, make no speed or quality ranking. Check whether both responses completed, whether their counts/content agree, and whether the warm-up differs. Predictions can be wrong without the experiment being broken.

5.6.2 Continue immediately or finish

The cloud inference reprise in Lab 6.4 reuses this exact model and loopback runtime. Continue only when it was included in the approved session and sufficient time remains for its requests **and export/cleanup**. It does not authorize leaving the GPU running overnight or extending a timer. Otherwise, finish now and plan a separate later session.

5.6.3 Export before termination

Stop the model and sampler using the recorded supervisor PIDs, and retain logs:

```
kill -TERM "$(cat "$RUN/server-supervisor.pid")" 2>/dev/null || true
kill -TERM "$(cat "$RUN/gpu-sampler.pid")" 2>/dev/null || true
date -u > "$RUN/runtime-stopped.txt"
# Include Lab 6.4 evidence too, if it was run in this session.
tar -czf "$ROOT/lab24-evidence.tgz" -C "$ROOT" evidence env.sh
sha256sum "$ROOT/lab24-evidence.tgz"
ls -lh "$ROOT/lab24-evidence.tgz"
```

Copy the archive to your computer through an already approved file-transfer method. For a browser-only terminal and a small archive, `base64 -w 0 "$ROOT/lab24-evidence.tgz"; printf '\n'`

produces text you can copy into a local file named `lab24-evidence.b64`. Decode on your computer with:

```
python3 - <<'PY'
import base64, hashlib
from pathlib import Path
text = "".join(Path("lab24-evidence.b64").read_text().split())
data = base64.b64decode(text, validate=True)
Path("lab24-evidence.tgz").write_bytes(data)
print(hashlib.sha256(data).hexdigest())
PY
tar -tzf lab24-evidence.tgz
```

Match the hash to the instance's hash and open the request, response, and record files locally. Copy only the encoded text, not terminal prompts. If the archive is too large to copy reliably, use an approved transfer route; do not expose a public download server or create an unplanned storage service. A file left on the instance is not exported. Do not include weights, credentials, or the entire home directory in the archive.

Complete the termination workflow from Lab 1.4, checking these GPU-session resources:

1. **Terminate the exact lab instance.** This permanently removes its instance storage and any EBS volumes marked for deletion; export first. Wait for the terminated state.
2. Inspect the recorded EBS volume IDs. Confirm deletion, or record any retained volume and its ongoing cost. Also check applicable backup/Recycle Bin retention; do not override organization retention policy.
3. Check **Elastic IP addresses** and public IPv4 inventory. An automatically assigned address should be released with the instance; release any separately allocated lab address only after checking ownership and dependencies.
4. Check for lab snapshots, unintended extra instances, volumes, endpoints, or paid network resources. Remove only resources created for this exercise and safe to remove. A security group itself is not a running GPU, and unrelated account resources are not this lab's cleanup target.
5. Record termination time, residual resources, the latest reported cost and billing-data timestamp. Revisit billing after usage has appeared; "not yet reported" does not mean zero cost.

If you only **stop** an instance to recover evidence, give its retained storage a named owner, a removal deadline, and an estimate. Stopping is an intermediate state, not final cleanup. See instance termination, EBS lifecycle, and EBS retention considerations.

6 Troubleshooting without enlarging the experiment

Symptom	Next check
Zero G-instance quota	Resolve quota before paying for anything; keep this attempt <code>not_run</code>
Insufficient regional/AZ capacity	Record the error; retry later or approve a newly priced alternative

Symptom	Next check
Cannot connect	Check username, image support, EIC prefix list, subnet route, and IAM permissions; stop the instance while investigating
CUDA missing or build fails	Verify image, <code>/usr/local/cuda-12.8</code> , driver, and build log; do not accept CPU fallback as the GPU run
Weight size or SHA-256 differs	Do not load it; check exact revision/path and incomplete download
OOM or partial GPU offload	Preserve placement/error evidence and clean up; revise the resource/runtime plan separately
Request times out	End the sequence and terminate the runtime; a client timeout alone is not cleanup
No final answer within 128 tokens	Record <code>finish_reason</code> , reasoning/final fields, and actual counts; distinguish truncation from a load failure
Session deadline arrives	Stop generating, export what exists, and terminate; retain a partial outcome

7 Optional extension — Plan for gpt-oss-120b

The 120B model is not required to complete this lab. OpenAI describes roughly 117B total and 5.1B active parameters, with a supported quantized path on an 80 GB GPU. That is an implementation-dependent starting point, not permission to launch an expensive multi-GPU instance. See OpenAI's 120B card.

Write a proposal rather than downloading it now: which question needs the larger model, which pinned representation/runtime supports the intended hardware, available GPU and host memory, download/storage peak, current regional quota/capacity, complete cost, and cleanup owner. Explain why the 20B result is insufficient. Obtain a separate resource and spending decision before any 120B run. The 20B evidence remains valuable even when this extension is declined.

8 What did we just do?

We turned an open-weight model into a bounded, inspectable inference session: selected hardware, checked a runtime and artifact identity, applied the right conversation format, measured a defined request, and closed the cloud resources.

Submit a short account of:

- **Identity:** model, conversion, hashes, runtime commit, image, hardware, and effective server settings.
- **Outcome:** `not_run`, `failed`, `partial`, or `completed`; all three attempts, including warm-up and any truncation/errors.

- **Observation:** actual token counts, defined timings, memory samples, and the limits of those measurements.
- **Interpretation:** which predictions held, and what belongs to the model versus the runtime or cloud setup.
- **Cost and closure:** approved estimate, elapsed paid session, exported archive verification, termination, retained-resource status, and any billing follow-up still due.

A live endpoint alone is not the deliverable. Neither a price estimate nor a successful small-model exercise demonstrates that these 20B weights ran. Your saved evidence should let another learner tell those cases apart.

9 More Learning

- OpenAI gpt-oss model card — architecture, evaluation, and model limitations.
- OpenAI gpt-oss repository — model formats and reference-implementation tradeoffs.
- OpenAI Harmony format — why chat serialization matters.
- llama.cpp gpt-oss guide — supported inference configurations and memory categories.
- Pinned llama.cpp server reference — exact request fields and server controls used here.
- AWS Deep Learning AMI setup — image discovery, launch, connection, and lifecycle.
- AWS EC2 instance specifications — hardware facts to verify before pricing a replacement.