Featured

Getting started

Hello, world

Simple web scraper

Serving web endpoints

Large language models (LLMs)

Serverless TensorRT-LLM (LLaMA 3 8B)

View on GitHub

In this example, we demonstrate how to use the TensorRT-LLM framework to serve Meta's LLaMA 3 8B model at a total throughput of roughly 4,500 output tokens per second on a single NVIDIA A100 40GB GPU. At Modal's on-demand rate of ~$4/hr, that's under $0.20 per million tokens — on auto-scaling infrastructure and served via a customizable API.

Additional optimizations like speculative sampling and FP8 quantization can further improve throughput. For more on the throughput levels that are possible with TensorRT-LLM for different combinations of model, hardware, and workload, see the official benchmarks.

Overview

This guide is intended to document two things: the general process for building TensorRT-LLM on Modal and a specific configuration for serving the LLaMA 3 8B model.

Build process

Any given TensorRT-LLM service requires a multi-stage build process, starting from model weights and ending with a compiled engine. Because that process touches many sharp-edged high-performance components across the stack, it can easily go wrong in subtle and hard-to-debug ways that are idiosyncratic to specific systems. And debugging GPU workloads is expensive!

This example builds an entire service from scratch, from downloading weight tensors to responding to requests, and so serves as living, interactive documentation of a TensorRT-LLM build process that works on Modal.

Engine configuration

TensorRT-LLM is the Lamborghini of inference engines: it achieves seriously impressive performance, but only if you tune it carefully. We carefully document the choices we made here and point to additional resources so you know where and how you might adjust the parameters for your use case.

Installing TensorRT-LLM

To run TensorRT-LLM, we must first install it. Easier said than done!

In Modal, we define container images that run our serverless workloads. All Modal containers have access to GPU drivers via the underlying host environment, but we still need to install the software stack on top of the drivers, from the CUDA runtime up.

We start from the official `nvidia/cuda:12.1.1-devel-ubuntu22.04` image, which includes the CUDA runtime & development libraries and the environment configuration necessary to run them.

```
import modal

tensorrt_image = modal.Image.from_registry(
    "nvidia/cuda:12.1.1-devel-ubuntu22.04", add_python="3.10"
)
```

On top of that, we add some system dependencies of TensorRT-LLM, including OpenMPI for distributed communication, some core software like `git`, and the `tensorrt_llm` package itself.

```
tensorrt_image = tensorrt_image.apt_install(
    "openmpi-bin", "libopenmpi-dev", "git", "git-lfs", "wget"
).pip_install(
    "tensorrt_llm==0.10.0.dev2024042300",
    pre=True,
    extra_index_url="https://pypi.nvidia.com",
)
```

Note that we're doing this by method-chaining a number of calls to methods on the `modal.Image`. If you're familiar with Dockerfiles, you can think of this as a Pythonic interface to instructions like `RUN` and `CMD`.

End-to-end, this step takes five minutes. If you're reading this from top to bottom, you might want to stop here and execute the example with `modal run trtllm_llama.py` so that it runs in the background while you read the rest.

Downloading the Model

Next, we download the model we want to serve. In this case, we're using the instruction-tuned version of Meta's LLaMA 3 8B model. We use the function below to download the model from the Hugging Face Hub.

```
MODEL_DIR = "/root/model/model_input"
MODEL_ID = "NousResearch/Meta-Llama-3-8B"
MODEL_REVISION = "315b20096dc791d381d514deb5f8bd9c8d6d3061"  # pin model revisions to prev


def download_model():
    import os

    from huggingface_hub import snapshot_download
    from transformers.utils import move_cache

    os.makedirs(MODEL_DIR, exist_ok=True)
    snapshot_download(
        MODEL_ID,
        local_dir=MODEL_DIR,
        ignore_patterns=["*.pt", "*.bin"],  # using safetensors
        revision=MODEL_REVISION,
    )
    move_cache()
```

Just defining that function doesn't actually download the model, though. We can run it by adding it to the image's build process with `run_function`. The download process has its own dependencies, which we add here.

```
MINUTES = 60  # seconds
tensorrt_image = (  # update the image by downloading the model we're using
    tensorrt_image.pip_install(  # add utilities for downloading the model
        "hf-transfer==0.1.6",
        "huggingface_hub==0.22.2",
        "requests~=2.31.0",
    )
    .env(  # hf-transfer: faster downloads, but fewer comforts
        {"HF_HUB_ENABLE_HF_TRANSFER": "1"}
    )
    .run_function(  # download the model
        download_model,
```

```
        timeout=20 * MINUTES,
    )
)
```

Configuring the model

Now that we have the model downloaded, we need to convert it to a format that TensorRT-LLM can use. We use a convenience script provided by the TensorRT-LLM team. This script takes a few minutes to run.

```
GIT_HASH = "71d8d4d3dc655671f32535d6d2b60cab87f36e87"
CHECKPOINT_SCRIPT_URL = f"https://raw.githubusercontent.com/NVIDIA/TensorRT-LLM/{GIT_HASH}
```

TensorRT-LLM requires that a GPU be present to load the model, even though it isn't used directly during this conversion process. We'll use a single A100-40GB GPU for this example, but we have also tested it successfully with A10G, A100-80GB, and H100 GPUs.

The most important feature to track when selecting hardware to run on is GPU RAM: larger models, longer sequences, and bigger batches all require more memory, We tuned all three to maximize throughput on this example.

The amount of GPU RAM on a single card is a tight constraint for most LLMs: RAM is measured in tens of gigabytes and models have billions of floating point parameters, each consuming one to four bytes of memory. The performance cliff if you need to spill to CPU memory is steep, so the only solution is to split the model across multiple GPUs. This is particularly important when serving larger models (e.g. 70B or 8×22B).

```
N_GPUS = 1  # Heads up: this example has not yet been tested with multiple GPUs
GPU_CONFIG = modal.gpu.A100(count=N_GPUS)
```

This is also the point where we specify the data type for this model. We use IEEE 754-compliant half-precision floats, (`float16`), because we found that it resulted in marginally higher throughput, but the model is provided in Google's `bfloat16` format. On the latest Ada Lovelace chips, you might use `float8` to reduce GPU RAM usage and speed up inference, but note that the FP8 format is very new, so expect rough edges.

```
DTYPE = "float16"
```

We put that all together with another invocation of `.run_commands` .

```
CKPT_DIR = "/root/model/model_ckpt"
tensorrt_image = (  # update the image by converting the model to TensorRT format
    tensorrt_image.run_commands(  # takes ~5 minutes
        [
            f"wget {CHECKPOINT_SCRIPT_URL} -O /root/convert_checkpoint.py",
            f"python /root/convert_checkpoint.py --model_dir={MODEL_DIR} --output_dir={CKP
            + f" --tp_size={N_GPUS} --dtype={DTYPE}",
        ],
        gpu=GPU_CONFIG,  # GPU must be present to load tensorrt_llm
    )
)
```

Compiling the engine

TensorRT-LLM achieves its high throughput primarily by compiling the model: making concrete choices of CUDA kernels to execute for each operation. These kernels are much more specific than `matrix_multiply` or `softmax` — they have names like `maxwell_scudnn_winograd_128x128_ldg1_ldg4_tile148t_nt`. They are optimized for the specific types and shapes of tensors that the model uses and for the specific hardware that the model runs on.

That means we need to know all of that information a priori — more like the original TensorFlow, which defined static graphs, than like PyTorch, which builds up a graph of kernels dynamically at runtime.

This extra layer of constraint on our LLM service is precisely what allows TensorRT-LLM to achieve its high throughput.

So we need to specify things like the maximum batch size and the lengths of inputs and outputs. The closer these are to the actual values we'll use in production, the better the throughput we'll get.

```
MAX_INPUT_LEN, MAX_OUTPUT_LEN = 256, 256
MAX_BATCH_SIZE = (
    128  # better throughput at larger batch sizes, limited by GPU RAM
)
ENGINE_DIR = "/root/model/model_output"

SIZE_ARGS = f"--max_batch_size={MAX_BATCH_SIZE} --max_input_len={MAX_INPUT_LEN} --max_outp
```

There are many additional options you can pass to `trtllm-build` to tune the engine for your specific workload. You can find the document we used for LLaMA here, which you can use to adjust the arguments to fit your workloads, e.g. adjusting rotary embeddings and block sizes for longer contexts.

We selected plugins that accelerate two core components of the model: dense matrix multiplication and attention. You can read more about the plugin options here.

```
PLUGIN_ARGS = f"--gemm_plugin={DTYPE} --gpt_attention_plugin={DTYPE}"
```

We put all of this together with another invocation of `.run_commands`.

```
tensorrt_image = (  # update the image by building the TensorRT engine
    tensorrt_image.run_commands(  # takes ~5 minutes
        [
            f"trtllm-build --checkpoint_dir {CKPT_DIR} --output_dir {ENGINE_DIR}"
            + f" --tp_size={N_GPUS} --workers={N_GPUS}"
            + f" {SIZE_ARGS}"
            + f" {PLUGIN_ARGS}"
        ],
        gpu=GPU_CONFIG,  # TRT-LLM compilation is GPU-specific, so make sure this matches
    ).env(  # show more log information from the inference engine
        {"TLLM_LOG_LEVEL": "INFO"}
    )
)
```

Serving inference at thousands of tokens per second

Now that we have the engine compiled, we can serve it with Modal by creating an `App`.

```
app = modal.App(f"example-trtllm-{MODEL_ID}", image=tensorrt_image)
```

Thanks to our custom container runtime system, even this large, many gigabyte container boots in seconds.

At container start time, we boot up the engine, which completes in under 30 seconds. Container starts are triggered when Modal scales up your infrastructure, like the first time you run this code or the first time a request comes in after a period of inactivity.

Container lifecycles in Modal are managed via our `Cls` interface, so we define one below to manage the engine and run inference. For details, see this guide.

```
@app.cls(
    gpu=GPU_CONFIG,
    container_idle_timeout=10 * MINUTES,
)
class Model:
    @modal.enter()
    def load(self):
        """Loads the TRT-LLM engine and configures our tokenizer.
```

```
        The @enter decorator ensures that it runs only once per container, when it starts.
        import time

        print(
            f"{COLOR['HEADER']}🥶 Cold boot: spinning up TRT-LLM engine{COLOR['ENDC']}"
        )
        self.init_start = time.monotonic_ns()

        import tensorrt_llm
        from tensorrt_llm.runtime import ModelRunner
        from transformers import AutoTokenizer

        self.tokenizer = AutoTokenizer.from_pretrained(MODEL_ID)
        # LLaMA models do not have a padding token, so we use the EOS token
        self.tokenizer.add_special_tokens(
            {"pad_token": self.tokenizer.eos_token}
        )
        # and then we add it from the left, to minimize impact on the output
        self.tokenizer.padding_side = "left"
        self.pad_id = self.tokenizer.pad_token_id
        self.end_id = self.tokenizer.eos_token_id

        runner_kwargs = dict(
            engine_dir=f"{ENGINE_DIR}",
            lora_dir=None,
            rank=tensorrt_llm.mpi_rank(),  # this will need to be adjusted to use multiple
        )

        self.model = ModelRunner.from_dir(**runner_kwargs)

        self.init_duration_s = (time.monotonic_ns() - self.init_start) / 1e9
        print(
            f"{COLOR['HEADER']}🚀 Cold boot finished in {self.init_duration_s}s{COLOR['END
        )

    @modal.method()
    def generate(self, prompts: list[str], settings=None):
        """Generate responses to a batch of prompts, optionally with custom inference sett
        import time

        if settings is None:
            settings = dict(
                temperature=0.1,  # temperature 0 not allowed, so we set top_k to 1 to get
                top_k=1,
                stop_words_list=None,
                repetition_penalty=1.1,
            )

        settings[
            "max_new_tokens"
        ] = MAX_OUTPUT_LEN  # exceeding this will raise an error
        settings["end_id"] = self.end_id
```

```
settings["pad_id"] = self.pad_id

num_prompts = len(prompts)

if num_prompts > MAX_BATCH_SIZE:
    raise ValueError(
        f"Batch size {num_prompts} exceeds maximum of {MAX_BATCH_SIZE}"
    )

print(
    f"{COLOR['HEADER']}🚀 Generating completions for batch of size {num_prompts}..
)
start = time.monotonic_ns()

parsed_prompts = [
    self.tokenizer.apply_chat_template(
        [{"role": "user", "content": prompt}],
        add_generation_prompt=True,
        tokenize=False,
    )
    for prompt in prompts
]

print(
    f"{COLOR['HEADER']}Parsed prompts:{COLOR['ENDC']}",
    *parsed_prompts,
    sep="\n\t",
)

inputs_t = self.tokenizer(
    parsed_prompts, return_tensors="pt", padding=True, truncation=False
)["input_ids"]

print(
    f"{COLOR['HEADER']}Input tensors:{COLOR['ENDC']}", inputs_t[:, :8]
)

outputs_t = self.model.generate(inputs_t, **settings)

outputs_text = self.tokenizer.batch_decode(
    outputs_t[:, 0]
)  # only one output per input, so we index with 0

responses = [
    extract_assistant_response(output_text)
    for output_text in outputs_text
]
duration_s = (time.monotonic_ns() - start) / 1e9

num_tokens = sum(
    map(lambda r: len(self.tokenizer.encode(r)), responses)
)
```

```
        for prompt, response in zip(prompts, responses):
            print(
                f"{COLOR['HEADER']}{COLOR['GREEN']}{prompt}",
                f"\n{COLOR['BLUE']}{response}",
                "\n\n",
                sep=COLOR["ENDC"],
            )
            time.sleep(0.01)  # to avoid log truncation

        print(
            f"{COLOR['HEADER']}{COLOR['GREEN']}Generated {num_tokens} tokens from {MODEL_I
            f" throughput = {num_tokens / duration_s:.0f} tokens/second for batch of size
        )

        return responses
```

Calling our inference function

Now, how do we actually run the model?

There are two basic methods: from Python via our SDK or from anywhere, by setting up an API.

Calling inference from Python

To run our `Model`'s `.generate` method from Python, we just need to call it — with `.remote` appended to run it on Modal.

We wrap that logic in a `local_entrypoint` so you can run it from the command line with

```
modal run trtllm_llama.py
```

For simplicity, we hard-code a batch of 128 questions to ask the model.

```
@app.local_entrypoint()
def main():
    questions = [
        # Generic assistant questions
        "What are you?",
        "What can you do?",
        # Coding
        "Implement a Python function to compute the Fibonacci numbers.",
        "Write a Rust function that performs binary exponentiation.",
        "How do I allocate memory in C?",
        "What are the differences between Javascript and Python?",
```

```
"How do I find invalid indices in Postgres?",
"How can you implement a LRU (Least Recently Used) cache in Python?",
"What approach would you use to detect and prevent race conditions in a multithrea
"Can you explain how a decision tree algorithm works in machine learning?",
"How would you design a simple key-value store database from scratch?",
"How do you handle deadlock situations in concurrent programming?",
"What is the logic behind the A* search algorithm, and where is it used?",
"How can you design an efficient autocomplete system?",
"What approach would you take to design a secure session management system in a we
"How would you handle collision in a hash table?",
"How can you implement a load balancer for a distributed system?",
"Implement a Python class for a doubly linked list.",
"Write a Haskell function that generates prime numbers using the Sieve of Eratosth
"Develop a simple HTTP server in Rust.",
# Literate and creative writing
"What is the fable involving a fox and grapes?",
"Who does Harry turn into a balloon?",
"Write a story in the style of James Joyce about a trip to the Australian outback
"Write a tale about a time-traveling historian who's determined to witness the mos
"Describe a day in the life of a secret agent who's also a full-time parent.",
"Create a story about a detective who can communicate with animals.",
"What is the most unusual thing about living in a city floating in the clouds?",
"In a world where dreams are shared, what happens when a nightmare invades a peace
"Describe the adventure of a lifetime for a group of friends who found a map leadi
"Tell a story about a musician who discovers that their music has magical powers."
"In a world where people age backwards, describe the life of a 5-year-old man.",
"Create a tale about a painter whose artwork comes to life every night.",
"What happens when a poet's verses start to predict future events?",
"Imagine a world where books can talk. How does a librarian handle them?",
"Tell a story about an astronaut who discovered a planet populated by plants.",
"Describe the journey of a letter traveling through the most sophisticated postal
"Write a tale about a chef whose food can evoke memories from the eater's past.",
"Write a poem in the style of Walt Whitman about the modern digital world.",
"Create a short story about a society where people can only speak in metaphors.",
"What are the main themes in Dostoevsky's 'Crime and Punishment'?",
# History and Philosophy
"What were the major contributing factors to the fall of the Roman Empire?",
"How did the invention of the printing press revolutionize European society?",
"What are the effects of quantitative easing?",
"How did the Greek philosophers influence economic thought in the ancient world?",
"What were the economic and philosophical factors that led to the fall of the Sovi
"How did decolonization in the 20th century change the geopolitical map?",
"What was the influence of the Khmer Empire on Southeast Asia's history and cultur
"What led to the rise and fall of the Mongol Empire?",
"Discuss the effects of the Industrial Revolution on urban development in 19th cen
"How did the Treaty of Versailles contribute to the outbreak of World War II?",
"What led to the rise and fall of the Mongol Empire?",
"Discuss the effects of the Industrial Revolution on urban development in 19th cen
"How did the Treaty of Versailles contribute to the outbreak of World War II?",
"Explain the concept of 'tabula rasa' in John Locke's philosophy.",
"What does Nietzsche mean by 'ressentiment'?",
"Compare and contrast the early and late works of Ludwig Wittgenstein. Which do yo
```

```
"How does the trolley problem explore the ethics of decision-making in critical si
# Thoughtfulness
"Describe the city of the future, considering advances in technology, environmenta
"In a dystopian future where water is the most valuable commodity, how would socie
"If a scientist discovers immortality, how could this impact society, economy, and
"What could be the potential implications of contact with an advanced alien civili
"Describe how you would mediate a conflict between two roommates about doing the d
"If you could design a school curriculum for the future, what subjects would you i
"How would society change if teleportation was invented and widely accessible?",
"Consider a future where artificial intelligence governs countries. What are the p
# Math
"What is the product of 9 and 8?",
"If a train travels 120 kilometers in 2 hours, what is its average speed?",
"Think through this step by step. If the sequence a_n is defined by a_1 = 3, a_2 =
"Think through this step by step. Calculate the sum of an arithmetic series with f
"Think through this step by step. What is the area of a triangle with vertices at
"Think through this step by step. Solve the following system of linear equations:
# Facts
"Who was Emperor Norton I, and what was his significance in San Francisco's histor
"What is the Voynich manuscript, and why has it perplexed scholars for centuries?"
"What was Project A119 and what were its objectives?",
"What is the 'Dyatlov Pass incident' and why does it remain a mystery?",
"What is the 'Emu War' that took place in Australia in the 1930s?",
"What is the 'Phantom Time Hypothesis' proposed by Heribert Illig?",
"Who was the 'Green Children of Woolpit' as per 12th-century English legend?",
"What are 'zombie stars' in the context of astronomy?",
"Who were the 'Dog-Headed Saint' and the 'Lion-Faced Saint' in medieval Christian
"What is the story of the 'Globsters', unidentified organic masses washed up on th
"Which countries in the European Union use currencies other than the Euro, and wha
# Multilingual
"战国时期最重要的人物是谁?",
"Tuende hatua kwa hatua. Hesabu jumla ya mfululizo wa kihesabu wenye neno la kwanz
"Kannst du die wichtigsten Eigenschaften und Funktionen des NMDA-Rezeptors beschre
"¿Cuáles son los principales impactos ambientales de la deforestación en la Amazon
"Décris la structure et le rôle de la mitochondrie dans une cellule.",
"Какие были социальные последствия Перестройки в Советском Союзе?",
# Economics and Business
"What are the principles of behavioral economics and how do they influence consume
"Discuss the impact of blockchain technology on traditional banking systems.",
"What are the long-term effects of trade wars on global economic stability?",
"What is the law of supply and demand?",
"Explain the concept of inflation and its typical causes.",
"What is a trade deficit, and why does it matter?",
"How do interest rates affect consumer spending and saving?",
"What is GDP and why is it important for measuring economic health?",
"What is the difference between revenue and profit?",
"Describe the role of a business plan in startup success.",
"How does market segmentation benefit a company?",
"Explain the concept of brand equity.",
"What are the advantages of franchising a business?",
"What are Michael Porter's five forces and how do they impact strategy for tech st
# Science and Technology
```

```
        "Discuss the potential impacts of quantum computing on data security.",
        "How could CRISPR technology change the future of medical treatments?",
        "Explain the significance of graphene in the development of future electronics.",
        "How do renewable energy sources compare to fossil fuels in terms of environmental
        "What are the most promising technologies for carbon capture and storage?",
        "Explain why the sky is blue.",
        "What is the principle behind the operation of a microwave oven?",
        "How does Newton's third law apply to rocket propulsion?",
        "What causes iron to rust?",
        "Describe the process of photosynthesis in simple terms.",
        "What is the role of a catalyst in a chemical reaction?",
        "What is the basic structure of a DNA molecule?",
        "How do vaccines work to protect the body from disease?",
        "Explain the significance of mitosis in cellular reproduction.",
        "What are tectonic plates and how do they affect earthquakes?",
        "How does the greenhouse effect contribute to global warming?",
        "Describe the water cycle and its importance to Earth's climate.",
        "What causes the phases of the Moon?",
        "How do black holes form?",
        "Explain the significance of the Big Bang theory.",
        "What is the function of the CPU in a computer system?",
        "Explain the difference between RAM and ROM.",
        "How does a solid-state drive (SSD) differ from a hard disk drive (HDD)?",
        "What role does the motherboard play in a computer system?",
        "Describe the purpose and function of a GPU.",
        "What is TensorRT? What role does it play in neural network inference?",
    ]

    model = Model()
    model.generate.remote(questions)
    # if you're calling this service from another Python project,
    # use [`Model.lookup`](https://modal.com/docs/reference/modal.Cls#lookup)
```

Calling inference via an API

We can use `modal.web_endpoint` and `app.function` to turn any Python function into a web API.

This API wrapper doesn't need all the dependencies of the core inference service, so we switch images here to a basic Linux image, `debian_slim`, which has everything we need.

```
web_image = modal.Image.debian_slim(python_version="3.10")
```

From there, we can take the same remote generation logic we used in `main` and serve it with only a few more lines of code.

```
@app.function(image=web_image)
@modal.web_endpoint(method="POST")
```

```
def generate_web(data: dict):
    return Model.generate.remote(data["prompts"], settings=None)
```

To set our function up as a web endpoint, we need to run this file — with `modal serve` to create a hot-reloading development server or `modal deploy` to deploy it to production.

```
modal serve trtllm_llama.py
```

You can test the endpoint by sending a POST request with `curl` from another terminal:

```
curl -X POST url-from-output-of-modal-serve-here \
-H "Content-Type: application/json" \
-d '{
    "prompts": ["Tell me a joke", "Describe a dream you had recently", "Share your favorit
}' | python -m json.tool # python for pretty-printing, optional
```

And now you have a high-throughput, low-latency, autoscaling API for serving LLaMA 3 8B completions!

Footer

The rest of the code in this example is utility code.

```
COLOR = {
    "HEADER": "\033[95m",
    "BLUE": "\033[94m",
    "GREEN": "\033[92m",
    "RED": "\033[91m",
    "ENDC": "\033[0m",
}
```

```
def extract_assistant_response(output_text):
    """Model-specific code to extract model responses.

    See this doc for LLaMA 3: https://llama.meta.com/docs/model-cards-and-prompt-formats/m
    # Split the output text by the assistant header token
    parts = output_text.split("<|start_header_id|>assistant<|end_header_id|>")

    if len(parts) > 1:
        # Join the parts after the first occurrence of the assistant header token
        response = parts[1].split("<|eot_id|>")[0].strip()

        # Remove any remaining special tokens and whitespace
```

```
        response = response.replace("<|eot_id|>", "").strip()

        return response
    else:
        return output_text
```