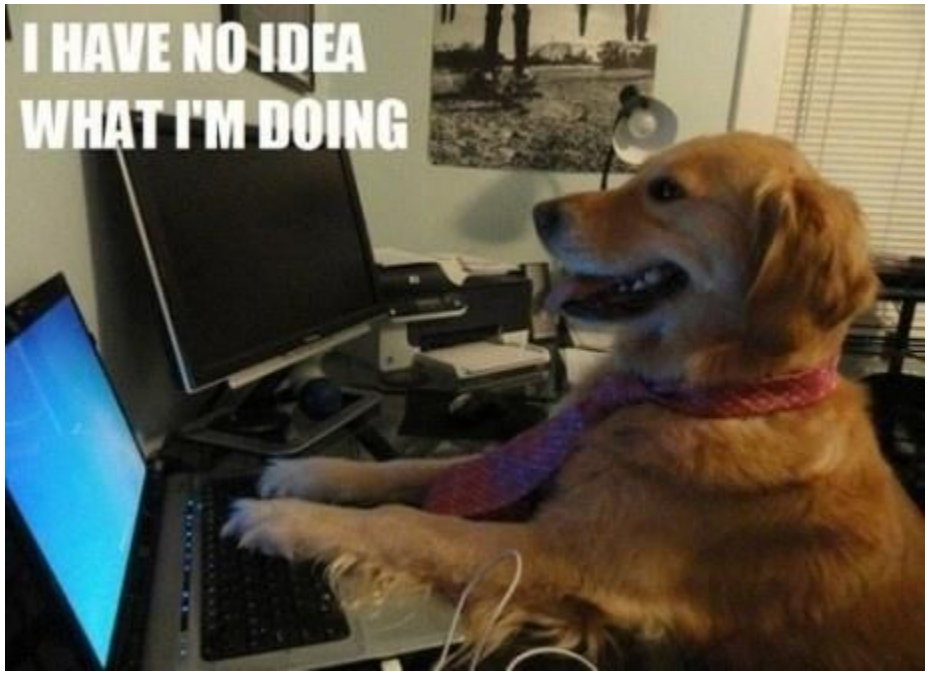
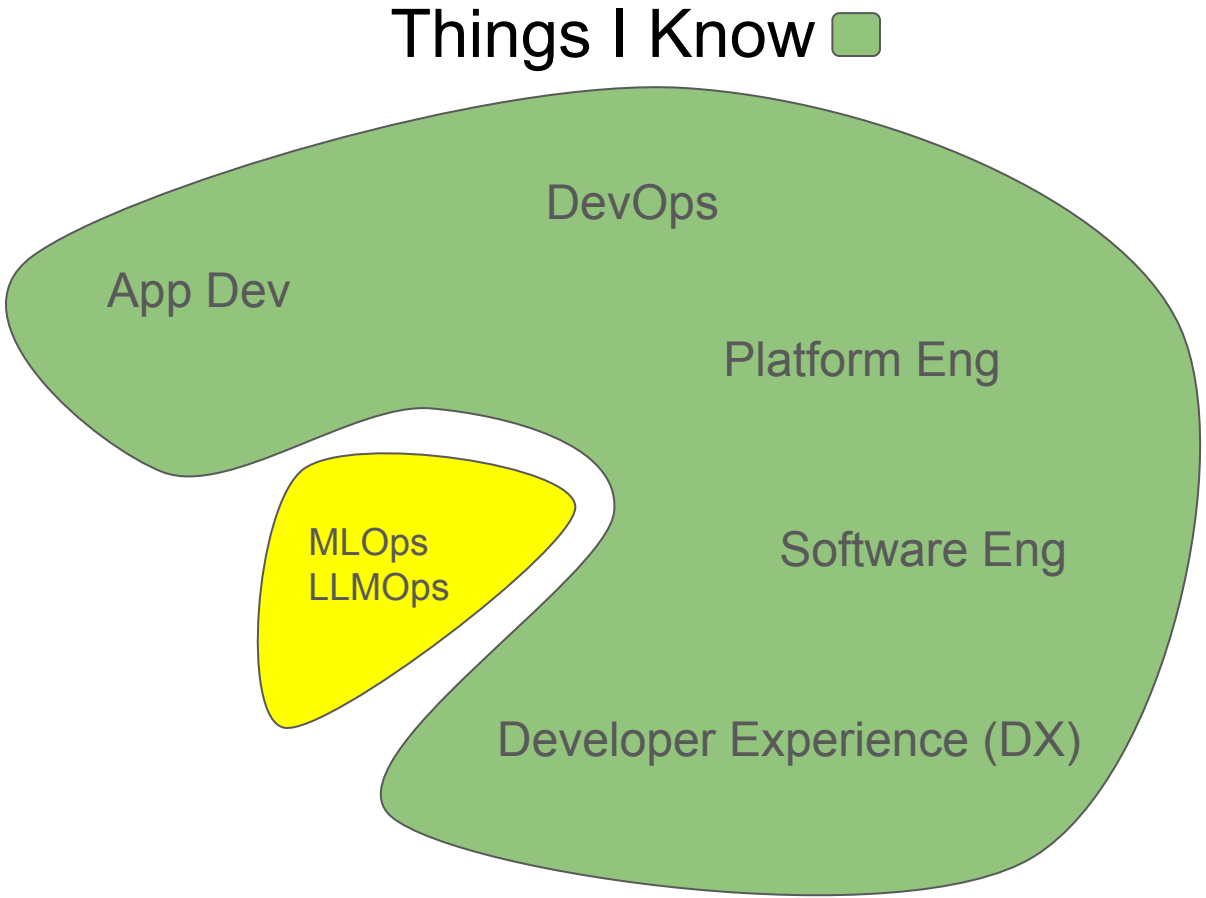
Self Hosting (Large Language) Models

About CD

- Senior Principal Engineer @ PMG
- Mostly platform engineering now
- Product engineering background
- Classically trained guitarist
  - https://www.classicalguitar.org/
- @chrisguitarguy nearly everywhere

Just a Quick Note Before We Start




Things I Know
DevOps
App Dev
Platform Eng
MLOps
LLMOps
Software Eng
Developer Experience (DX)

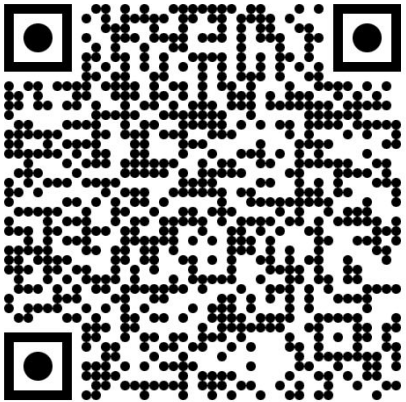
Goals

1. Do not dynamically download models at runtime
    a. This is fine for dev/test
    b. But a model file is basically remote code execution (download the o
    and execute it)
    c. Want more control and don't want models changing underneath us
    d. https://vickiboykis.com/2024/02/28/gguf-the-long-way-around/
2. Do not embed models directly into containers
    a. Models are *big* – even small LLMs are large
    b. Can take a long time to cold start just due to container size and download time
3. Hosted in $infraProviderOfChoice
    a. At PMG this is AWS with Elastic Container Service

Two Use Cases

1. Embedding a model into application code
2. Hosting an inference endpoint

Code/notes here:
https://github.com/AgencyPMG/self-hosting-large-language-models



Goal #1: Use Local Models

Saving Models Locally

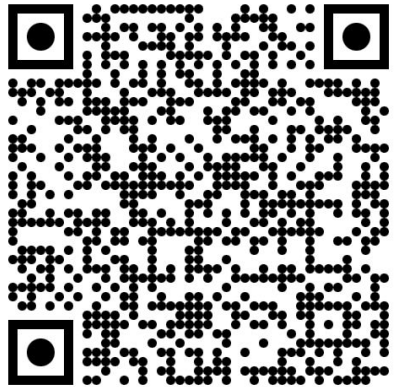
https://huggingface.co/docs/transformers/main/en/installation#offline-mode

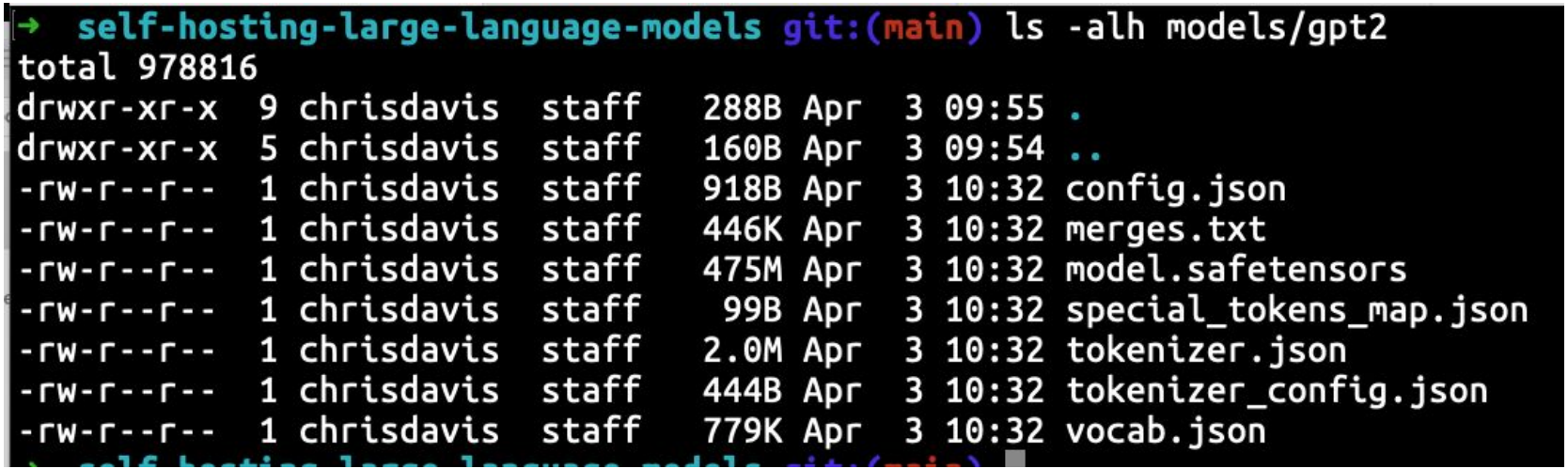
```python
import pathlib
from transformers import AutoTokenizer, AutoModel

def download_model(model_name: str, output_path: pathlib.Path):
    """
    Could drop this into a notebook to download the model in a specific
    environment (eg AWS sagemaker with inferentia)
    """
    if not output_path.exists():
        output_path.mkdir(parents=True)

    tokenizer = AutoTokenizer.from_pretrained(model_name)
    tokenizer.save_pretrained(str(output_path))

    model = AutoModel.from_pretrained(model_name)
    model.save_pretrained(str(output_path))
```




➜ self-hosting-large-language-models git:(main) ls -alh models/gpt2
total 978816
drwxr-xr-x 9 chrisdavis staff 288B Apr 3 09:55 .
drwxr-xr-x 5 chrisdavis staff 160B Apr 3 09:54 ..
-rw-r--r-- 1 chrisdavis staff 918B Apr 3 10:32 config.json
-rw-r--r-- 1 chrisdavis staff 446K Apr 3 10:32 merges.txt
-rw-r--r-- 1 chrisdavis staff 475M Apr 3 10:32 model.safetensors
-rw-r--r-- 1 chrisdavis staff 99B Apr 3 10:32 special_tokens_map.json
-rw-r--r-- 1 chrisdavis staff 2.0M Apr 3 10:32 tokenizer.json
-rw-r--r-- 1 chrisdavis staff 444B Apr 3 10:32 tokenizer_config.json
-rw-r--r-- 1 chrisdavis staff 779K Apr 3 10:32 vocab.json

Use a Local Model

Basically just point transformers to a local path :tada:

```python
import typing
import pathlib
from transformers import AutoTokenizer, AutoModelForCausalLM, pipeline

ModelResult = typing.Tuple[AutoModelForCausalLM, AutoTokenizer]

def load_model(model_path: pathlib.Path):
    """
    Could drop this into a notebook to download the model in a specific
    environment (eg AWS sagemaker with inferentia)
    """
    assert model_path.exists()

    tokenizer = AutoTokenizer.from_pretrained(str(model_path))
    model = AutoModelForCausalLM.from_pretrained(str(model_path))

    return (tokenizer, model,)

def prompt(model_path: pathlib.Path, prompt: str):
    tokenizer, model = load_model(model_path)

    p = pipeline('text-generation', model=model, tokenizer=tokenizer)
    return p(prompt)
```

An App with a Local Model

- Small FastAPI app
- Load the model and tokenizer at boot time
- Validate the path of the model exists at boot time
- We need this as an example to tackle Goal #2
- https://github.com/AgencyPMG/self-hosting-large-language-models/blob/main/app_example

Quick Demo

Goal #2: Don’t Embed Models in Containers

Containerize the App Example

- But don’t embed the model
- Just add app code, deps, etc
- MODEL_ROOT_PATH is not set

```
FROM python:3.12-slim-bookworm

ADD requirements.txt app_example/main.py /app/

WORKDIR /app

RUN apt-get update \
    && apt-get -y install gcc python3-dev \
    && pip install -r requirements.txt

EXPOSE 8000

CMD ["uvicorn",  "main:app", "--host", "0.0.0.0", "--port", "8000"]
```

Run the Container with a Volume & Env Var

Quick Demo

```
exec docker run --rm \
    -p 8000:8000 \
    --volume "$(pwd)/models:/models" \
    --env MODEL_ROOT_PATH=/models \
    --env MODEL_VERSION=gpt2 \
    app_example
```

This is the Proof of Concept for Goal #2

- Can we run a model without embedding the pretrained model in the container?
- Yes!
- Now we get to the infra portions: how do we mount a volume with the container in a prod-like environment?

Everything You Need to Know About AWS

Why Containers vs “Serverless”

- Loading a model takes a minute
- More practical do that in a long running process
- With a lambda, the cold start time might be very impacted by model load
    - This is more important for something serving an HTTP request than a background process

To Run a Workload in AWS…

You'll need…

- A virtual private cloud (VPC)
- IAM Roles
- Security Groups
- ECS/EKS cluster
- Load Balancer
- HTTPS Certificates
- Etc.

We’re Gonna Ignore All That

And cheat a little bit…

The Interesting Bit

```
exec docker run --rm \
    -p 8000:8000 \
    --volume "$(pwd)/models:/models" \
    --env MODEL_ROOT_PATH=/models \
    app_example
```

Distributed File Systems

Host and manage models in a distributed file system. Attach the filesystem to tasks or servers.

AWS has a few options:

- **EFS** https://aws.amazon.com/efs/
  - Network Attached Storage (basically)
- FSx https://aws.amazon.com/fsx/lustre/
  - High performance, FSx for Lustre is probably more relevant for training than inference
  - Inference we're loading the model once vs reading/writing data at high velocity

EFS Concepts

- A *Filesystem*
  - What it sounds like, where the files are stored
- An *Access Point*
  - Combo of unix-like user and group IDs and root directories, etc
  - Can be used with filesystem *policies* to enforce access
  - See https://chrisguitarguy.com/2017/12/19/aws-service-policies-vs-iam-permissions/
- A *Mount Target*
  - An Elastic Network Interface (ENI) for the filesystem in an availability zone and subnet in a VPC

Solution: Mount an EFS File System in a Container

We'll use ECS to do this, but EKS can do it too.

EFS Configuration
https://github.com/AgencyPMG/self-hosting-large-language-models/blob/main/infrastructure/efs.tf

App Container Configuration
https://github.com/AgencyPMG/self-hosting-large-language-models/blob/main/infrastructure/ecs_app_example.tf

Remember we're hand waving away a bunch of supporting infrastructure here.

This is One Half of Solving This:

```
exec docker run --rm \
    -p 8000:8000 \
    --volume "$(pwd)/models:/models" \
    --env MODEL_ROOT_PATH=/models \
    app_example
```

Now we have the volume mounted in the container, and we need to get models in it.

Enter DataSync (and S3)

EFS needs to be mounted to be used. That's not very fun to automate.

Instead we can make a model repository in S3 and setup AWS DataSync to ship those models to EFS for use.

- A *location* is somewhere data sync is going to pull files from or push files to
- A *task* is the config to execute a data sync from one destination to another
- A *task execution* is an actual "run" of the task to sync

https://github.com/AgencyPMG/self-hosting-large-language-models/blob/main/infrastructure/datasync.tf

Treating Models as “Attached Resources”

A combo of two 12 factor app concepts:

1. Config in Environment Variables https://12factor.net/config
2. Backing Services https://12factor.net/backing-services

Model root path is a config example. And the model we use is the *backing service* (sorta).

Deploying a Model Version

https://github.com/AgencyPMG/self-hosting-large-language-models/blob/main/bin/infra/deploy_app_model

https://github.com/AgencyPMG/self-hosting-large-language-models/blob/main/bin/infra/change_app_model_version

We’ve Deployed an “App” with a Model

But What About Inference Endpoints?

Could Do Basically the Same Thing as the “App”

1. Containerize inference server software
    a. https://developer.nvidia.com/triton-inference-server
    b. https://github.com/bentoml/OpenLLM
    c. https://docs.vllm.ai/en/latest/
    d. https://kserve.github.io/website/latest/
    e. Lots of options here
2. Mount model code via EFS
3. Run on ECS/EKS/Whatever
4. ???
5. Profit

But Let’s Do Something Different

AWS Sagemaker

- Serverless and provisioned inference endpoints
- Model “registry”
- Other MLOps pipeline things to play with, but we’ll focus on serverless inference endpoints

Sagemaker Endpoints Core Concepts

- A *model* is an immutable, hosted model
  - Includes what container image it uses
  - As well as model data (eg pre-trained parameters, etc)
- A *endpoint configuration* defines how a model will run on an endpoint
  - This includes serverless vs hosted
  - Concurrency limits
  - And, most importantly, what model is running
- An *endpoint* is the actual inference endpoint

https://github.com/AgencyPMG/self-hosting-large-language-models/blob/main/infrastructure/sagemaker.tf

Deployment Process

Very close to the earlier docker + ECS app:

1. Build and bundle pre-trained model
2. Upload to S3
3. Create a new Sagemaker model with the pre-trained model
4. Create a new Sagemaker endpoint config pointing to the model
5. Update the inference endpoint with the new endpoint config

https://github.com/AgencyPMG/self-hosting-large-language-models/blob/main/bin/infra/deploy_inference_model

Summary

Remember the Goals:

1. Don’t download models at runtime
2. Don’t embed models with source code

If you get one thing from this talk:

A *model repository* somewhere (eg S3) can act as a handoff point between teams.

Questions?