# Serving JAX Models with vLLM & SGLang

Leveraging Popular OSS Servers for JAX Model Deployment

# Why Serve JAX Models with OSS Servers?

- Many enterprises already have established infrastructure for training and serving ML models, often centered around PyTorch.
- Requiring new, JAX-specific infrastructure creates adoption barriers.
- Supporting popular OSS servers like vLLM and SGLang allows JAX models to fit into existing GenAI workflows.
- This facilitates a smoother, incremental transition to JAX, allowing businesses to prototype and eventually replace PyTorch models in production.

# Your JAX Model

This flow assumes that you have a model implemented in JAX

- In this case, we're going to load pre-trained weights from Hugging Face into an equivalent JAX model implementation
- You could also have pre-trained your own model
- Once you have a model and weights in JAX, you work with it in JAX until you're ready to serve it
- When you want to serve it, you convert it to a serveable format for your target servers - in this case vLLM or SGLang

# Basic Workflow: From JAX to Served Model

**Load Model** → **(Optional) Modify Model** → **Convert & Save Weights** → **Serve**

### Load Model

Obtain model weights, often from Hugging Face, in JAX/Flax format using safetensors.

### (Optional) Modify Model

Perform fine-tuning or other alterations using JAX.

(Note: Specific JAX modifications are outside the scope of this presentation).

### Convert & Save Weights

Convert JAX weights to a compatible format (like a flattened dictionary of tensors) and save them back into safetensors format.

### Serve

Load the prepared safetensors model into the chosen OSS server (vLLM or SGLang) and run inference.

**Note**: Some layer types will require weights to be transposed or permuted for conversion to JAX.

# Code Example: Loading Safetensors into JAX/Flax

```python
import jax
from pathlib import Path
from safetensors import safe_open

def load_safetensors(path_to_model_weights):
    weights = {}
    # Use pathlib to find all .safetensors files
    safetensors_files = Path(path_to_model_weights).glob('*.safetensors')
    for file in safetensors_files:
        # Open each file, specifying 'flax' framework
        with safe_open(file, framework="flax") as f:
            for key in f.keys():
                weights[key] = f.get_tensor(key)
    return weights
```

# Map the PyTorch weights to Flax NNX

Here's a quick summary:

- **Linear (FC)**: Transpose
- **Convolutions**: Transpose from `[outC, inC, kH, kW]` to `[kH, kW, inC, outC]`

```
# [outC, inC, kH, kW] -> [kH, kW, inC, outC]
kernel = jnp.transpose(kernel, (2, 3, 1, 0))
```

- **BatchNorm**: No change

Google

# Map the PyTorch weights to Flax NNX

- **Convolutions and FC Layers**:
  - **PyTorch**: The activations will have shape [N, C, H, W] after the convolutions and are then reshaped to [N, C * H * W] before being fed to the fc layers.
  - **Flax**: The activations after the convolutions will be of shape [N, H, W, C] in Flax. Before we reshape the activations for the fc layers, we have to transpose them to [N, C, H, W].

# Working on your model with JAX

Now your model is ready for post-training

- You could be:
  - Fine-tuning
  - Aligning
  - Merging
  - ?
- We'll see an example of Llama 3.2 in the notebook

Next Step: Preparing your model for serving

# Code Example: JAX -> PyTorch & Flattening

```
# Flatten the nested dictionary structure required by servers
def flatten_weight_dict(torch_params, prefix=""):
    flat_params = {}
    for key, value in torch_params.items():
        new_key = f"{prefix}{key}" if prefix else key
        if isinstance(value, dict):
            flat_params.update(flatten_weight_dict(value, new_key + "."))
        else:
            flat_params[new_key] = value
    return flat_params
```

# Code Example: JAX -> PyTorch & Flattening

```
from safetensors.flax import save_file

# Usage:
jax_weights = load_safetensors(...) # From previous step
servable_weights = flatten_weight_dict(jax_weights)
save_file(servable_weights, path_to_model_weights + '/model.safetensors')
```

Google

# Which models can you serve with vLLM?

- While safetensors is a required format for the model's weights, vLLM has two other critical requirements that determine compatibility.
- `config.json` and `tokenizer.json` files
  - We reuse the files we downloaded from Hugging Face in our example
- **Supported models list**: If your model is not in the vLLM supported models list, it needs to meet vLLMs requirements as a **custom model**.

https://docs.vllm.ai/en/latest/models/supported_models.html#custom-models

# Example: Serving with vLLM

- Uses the vLLM library.
- Requires converted/flattened weights saved in safetensors format.
- Initialize LLM object pointing to the model directory containing the safetensors file.
- Use `llm.generate()` method for inference.

# Example: Serving with vLLM

```python
import os
from vllm import LLM, SamplingParams

# Define model path (must contain the converted safetensors)
model_id = "meta-llama/Llama-3.2-1B"
path_to_model_weights = os.path.join('/content', model_id)

# Load the model using vLLM
# Specify safetensors format and desired dtype
llm = LLM(model=path_to_model_weights, load_format="safetensors", dtype="half")
```

Google

# Example: Serving with vLLM

```python
# Define prompts and sampling parameters
prompts = ["The capital of France is"]
sampling_params = SamplingParams(temperature=0.8, top_p=0.95)

# Generate text
outputs = llm.generate(prompts, sampling_params)

# Print results
for output in outputs:
    prompt = output.prompt
    generated_text = output.outputs[0].text
    print("-" * 10)
    print(f"Prompt: {prompt}\nGenerated text: {generated_text}")
```

# Which models can you serve with SGLang?

- Similar to vLLM, SGLang requires a `config.json` and a `tokenizer.json`
  - It may also require a `tokenizer_config.json` and a `special_tokens_map.json` depending on the model.
- You can also register an external model implementation.

https://docs.sglang.ai/supported_models/support_new_models.html

# Example: Serving with SGLang

- Uses the SGLang library.
- Also requires converted/flattened weights saved in safetensors format.
- Initialize `sgl.Engine` object pointing to the model directory.
- Use `llm.generate()` method for inference, passing sampling parameters as a dictionary
- Currently requires CUDA 12.4

# Example: Serving with SGLang

```python
import os
import sglang as sgl

# Define model path (must contain the converted safetensors)
model_id = "meta-llama/Llama-3.2-1B"
path_to_model_weights = os.path.join('/content', model_id)

# Launch the SGLang engine
llm = sgl.Engine(model_path=path_to_model_weights)
```

# Example: Serving with SGLang

```python
# Define prompts and sampling parameters
prompts = ["The capital of France is"]
sampling_params = {"temperature": 0.8, "top_p": 0.95}

# Generate text
outputs = llm.generate(prompts, sampling_params)

# Print results
for prompt, output in zip(prompts, outputs):
    print("-" * 10)
    print(f"Prompt: {prompt}\nGenerated text: {output['text']}")
```

# Summary and References

- **Capability**: Proof-of-concepts demonstrate it's feasible to serve JAX models (after conversion) using popular OSS engines like vLLM and SGLang on GPUs.
- **Benefit**: Enables enterprises to integrate JAX into existing PyTorch-based serving infrastructure, facilitating adoption.
- JAX: https://jax.dev
- vLLM: https://docs.vllm.ai/
- SGLang: https://docs.sglang.ai/

# Learning Resources

Code Exercises, Quick References, and Slides

- https://goo.gle/learning-jax

Learn JAX

# Community and Docs

Community:

- https://goo.gle/jax-community

Docs

- JAX AI Stack: https://jaxstack.ai
- JAX: https://jax.dev
- Flax NNX: https://flax.readthedocs.io