JAX Serving with vLLM & SGLang

Quick Reference

This cheat sheet provides a quick reference for the functions and utilities used to convert a JAX model and serve it with the popular open-source servers vLLM and SGLang.

End-to-End Workflow

The process involves preparing the JAX model weights and then loading them into a serving engine.

Step	Action	Description
Load	Load model weights into JAX/Flax.	Use the `load_safetensors` utility to read `.safetensors` files from a directory into a nested dictionary.
Transpose	Transpose weights for specific layers.	**Crucial Step:** Before flattening, transpose weights for layers like Dense/Linear to match the PyTorch format.
Convert	Flatten the weight dictionary.	Use the `flatten_weight_dict` utility to convert the (now transposed) nested JAX weights into a flat key-value structure.
Save	Save the flattened weights.	Use `safetensors.flax.save_file` to write the flattened dictionary to a single `model.safetensors` file.
Serve	Load the model into a server.	Point either vLLM or SGLang to the directory containing the new `model.safetensors` file to start the inference engine.

Key Functions & Code Examples

1. Loading Safetensors into JAX/Flax

This function aggregates all `.safetensors` files in a directory into a single JAX/Flax compatible weight dictionary.

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

def load_safetensors(path_to_model_weights):
    weights = {}
    safetensors_files =
Path(path_to_model_weights).glob('*.safetensors')
    for file in safetensors_files:
        with safe_open(file, framework="flax") as f:
            for key in f.keys():
                weights[key] = f.get_tensor(key)
    return weights
```

2. Handling Weight Transposition (Important!)

Before flattening, certain JAX layer weights **must be transposed** to match the dimensional format expected by PyTorch-based servers. Failure to do so will result in errors or incorrect model outputs.

- **Dense/Linear Layers:** The `kernel` weights must be transposed. JAX typically uses `(input_features, output_features)`, while PyTorch expects `(output_features, input_features)`.
- **Attention Projections (e.g., `q_proj`, `k_proj`, `v_proj`, `o_proj`):** These are linear layers, and their `kernel` weights must also be transposed.
- **Embedding Layers:** These weights are generally compatible and do **not** need to be transposed.

You must implement logic to iterate through your loaded `jax_weights` dictionary and apply `.T` to the appropriate tensors before flattening. The exact implementation will depend on your model's specific structure.

3. Flattening & Saving Weights for Serving

This function converts the nested (and now correctly transposed) JAX weight dictionary into a flat format and saves it.

```python
from safetensors.flax import save_file

# Utility to flatten the nested dictionary
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

# --- Usage ---
# jax_weights = load_safetensors(...)
# # ... (Apply transpositions to jax_weights here) ...
# servable_weights = flatten_weight_dict(jax_weights)
# save_file(servable_weights, 'path/to/model/model.safetensors')
```

4. Serving with vLLM

Initialize the `LLM` class pointing to your model directory and use `.generate()`.

```python
from vllm import LLM, SamplingParams

# Point to the directory with the converted model.safetensors
llm = LLM(model="/path/to/your/model", load_format="safetensors",
dtype="half")
```

```
# 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)
```

5. Serving with SGLang

Initialize the `sgl.Engine` and use its `.generate()` method. **Note:** Requires CUDA 12.4.

```python
import sglang as sgl

# Point to the directory with the converted model.safetensors
llm = sgl.Engine(model_path="/path/to/your/model")

# 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)
```