Link to the Drive with Checkpoints:
https://drive.google.com/drive/folders/1ir8xxywEh3fWsWhd66ARkIDFHPQzCoBU?usp=sharing

Link to the Github Repo:
https://github.com/Sou0602/commavq_nanogpt

	Config 1 config/ train_commavq.py	**Config 2** config/ train_commavq_1.py	**Config 3** config/ train_commavq_2.py
Block Size	2580	2580	2580
Batch Size	12	12	12
Embedding Size	512	512	516
N-Heads	8	8	6
N-Layers	8	8	6
Vocab_Size/ Output Logits	1048	1048	1026
Learning Rate	1e-4	8e-5	1e-4
Out Dir	out-commavq	out-commavq_1	out-commavq_2
Dropout	0.2	0.2	0.15
Gradient Accumulation	1	4	1
FLOPs	~41 TFLOPs	~41TFLOPs	~25TFLOPs
Number of Parameters	25.71 M	25.71M	19.71M
Tokens per Iteration	30,960	123,840	30,960
Trained Until	~ 500,000 iterations	~ 163,000 iterations	~ 480,000 iterations
Train Time per iteration	21ms	80ms	35ms
Inference Loss	2.3778	2.4017	2.442

Inference Time Taken(in mins)	2:26	2:28	2:06

I started training with 6 heads and 6 layers, the configuration from the Shakespeare character model, which I used for debugging Karpathy's nanogpt model. (https://github.com/karpathy/nanoGPT/tree/master)

Once the data is loaded, follow the instructions from commaai/commavq github repo for the nanogpt (https://github.com/commaai/commavq/blob/master/nanogpt/prepare.py)

The dataset has a size of 1.54801 * 10^10 tokens, with 30,960 tokens per iterations, one pass over the entire training data would be ~ 500,000 iterations. Although the loss does not vary much from 250,000 iterations, I rant the models a little longer to complete a pass over the training data.

I trained simple models with 6 layers and 6 heads as a starting point. The vocab_size/ output_size that we need is 1026(for the 1024 tokens in the dataset, 2 tokens for separating the frames, and a token to separate the segments). The block size parameter is fixed from the dataset, given 20 frames and 129 frames per token. I chose the embeddings to be close to 512, but a factor of 512//6 is not an integer, so I went with 516. With a batch size of 12, I was using the full memory of the GPU (one node of RTX 6000 from a shared cluster), but the flops utilization was not high. (7% of A100 from Karpathy's MFU calculations, which is about 25 TFLOPs, and up to 60% of the RTX max FLOPs). [Config 3]

I then updated the model to 8 layers and 8 heads. A batch size of 12 was working for the forward pass. Adjusted the embeddings to 512. There were some compiler errors for 1026 as the output shape, so I used 1048 tokens as an output shape. (1040/1032 might work, but I have not tested). This configuration uses the FLOPs effectively up to 100% of the GPU Usage and gives better performance. [Config 1]

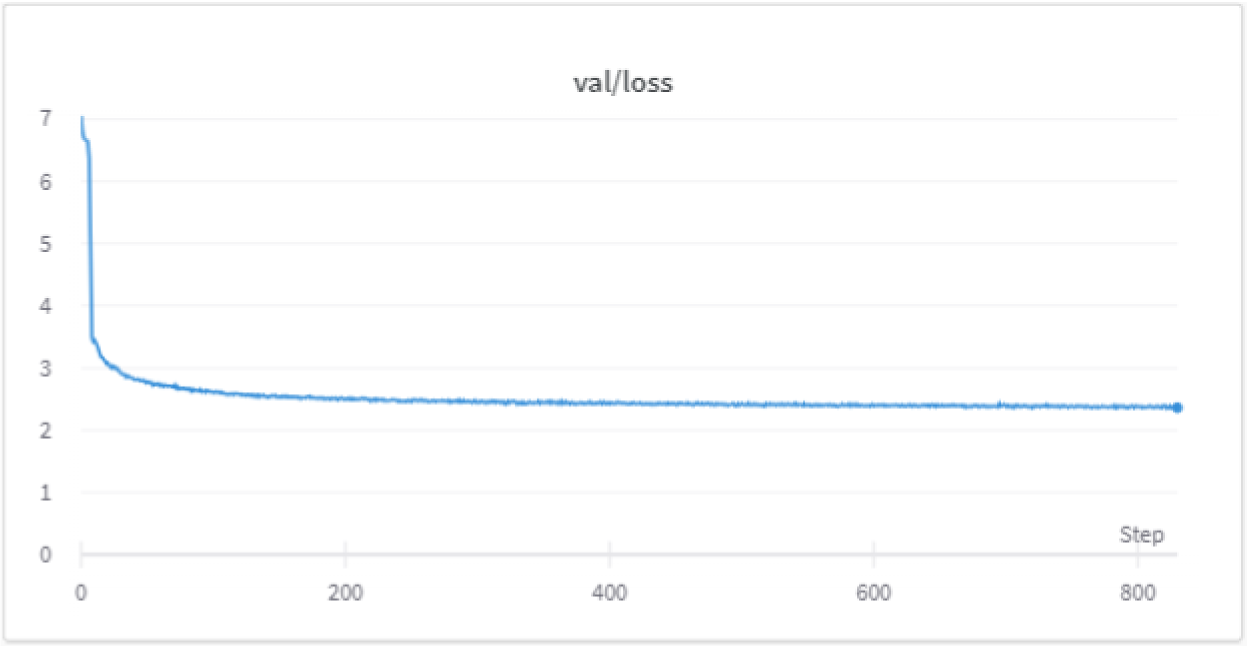
I also wanted to check if I can get any added benefit with gradient accumulation, increasing the effective batch. Gradient Accumulation is more effective with DDP, but with a single node, it is almost similar to the performance with gradient accumulation 1 but can be useful in scenarios where large batch updates are needed. [Config 2]

The evaluation script(eval.py) is updated from the eval notebook on the commaai/commavq github repo:
https://github.com/commaai/commavq/blob/master/notebooks/eval.ipynb

The performances with different configurations are mentioned in the table above.

Although the repo includes a script for single node training, for DDP training, copy the data/commavq folder to Karpathy's nanogpt data folder and copy the config/train_commavq.py file to the nanogpt config folder. (Refer to https://github.com/karpathy/nanoGPT/tree/master on how to run the DDP training)

The model and train loop are derived from Karpathy's nanogpt repo but are more structured in this version.



The X-Axis denotes the step, where validation loss is logged. The validation loss is logged every 500 iterations.

ONNX Considerations:

For faster inference, I was experimenting with converting torch model checkpoint to ONNX and TensorRT. I did not have access to a docker with GPUs (to use the Nvidia containers) to experiment with TensorRT but converted the checkpoint to ONNX Models, which can be used with OnnxRuntime and possibly to TensorRT in the future.

The current version of torch does not have support for the scaled_dot_product_attention operator. (https://github.com/pytorch/pytorch/issues/97262)
One of the possible fixes was to update to torch nightly, but the training was unstable, and the onnxruntime crashes with a segmentation fault.
The other fix was to add a custom operator in onnx. I followed this approach from:
https://github.com/pytorch/pytorch/pull/99658/files?diff=unified&w=0#diff-244955d820ec138d5ddffb20ee6f517cc4c5d281f19ccb53d8db47043b5ac46f

Made a change to the scale, ##(works with scale: parameter None for scaled_dot_product_attention)

The updated symbolic_opset14.py file is present in onnx-utils w/ instructions on how to update it in the torch/onnx repo.

I found the ONNX Inference times to be slower than Pytorch forward models, and followed some optimizations from https://onnxruntime.ai/docs/performance/ .

Evaluation with ONNX Runtime Inference Session has a high variance depending on the GPU Usage in the cluster, from 4:30 to 8:10(slower by 2 to 4 times, when compared to the torch inference models)

It still needs some work on profiling to understand the bottlenecks, optimizing them if possible. We can also directly convert the model to tensorRT and test the performance with it.

Some improvements and future directions here would be:

- Experimenting with various model architectures and hyperparameters and multi-node training
- Profiling onnx and further optimizing the inference session if possible
- Experiment with further converting the onnx models to tensorRT to check the model performance and speedups, if any.

How to run the code:(from the ReadME in the GitHub link)

# commavq-nano

## Creating the Environment
```
$ conda create --name <env_name> --file env_requirements.txt
$ conda activate <env_name>
```

## Load Data in the data/commavq directory

We need to create the data in the right format of train.bin and val.bin.

The file below is a modified version of prepare.py from the comma-vq dataset:
https://github.com/commaai/commavq/blob/master/nanogpt/prepare.py

```
$ python data/commavq prepare.py
```

The configuration files have 3 model configurations.
* train_commavq.py (8 heads, vocab_size = 1048)
* train_commavq_1.py (8 heads, vocab_size = 1048)
* train_commavq_2.py (6 heads, vocab_size = 1026)

In my models, vocab_size/output dimension is different for the model with 6 heads and with 8 heads.
The model with 6 heads has a vocab_size of 1026, and the model with 8 heads.

To update the vocab size, change the n_heads parameter. The default file has 1048 for 8 heads.
```
$ python data/commavq create_meta_pkl.py --n_heads=8
```

## Run Tests to check the data location, dataloader, and the model.
```
$ chmod +x run_tests.sh
```

```
$ ./run_tests.sh
```

## Training the configurations
```
$ python train.py config/train_commavq.py
```

To run different configurations, change the config file. Alternatively, create new configurations or update the parameters for different models in the config files.
For the model sizes, the training iterations can be stopped earlier, change the parameter max_iters for shorter training times.

## Evaluating the Models
This file is updated from the eval.ipynb notebook from the commavq repo, https://github.com/commaai/commavq/blob/master/notebooks/eval.ipynb

```
$ python eval.py
```

In the config/eval_commavq.py, change the out_dir parameter to evaluate different models.
Alternatively, the eval file also takes an argument --ckpt_path for evaluating the model at the input checkpoint path.

```
$ python eval.py --ckpt_path MODEL_PATH
```
## Exporting the Models to ONNX
The scaled_dot_product_attention operator is not included in the torch package. So, I updated the symbolic_opset14.py file in the onnx directory.(More details about this in the Training Log)

```
$ cp -f onnx-utils/symbolic_opset14.py $(pip show torch | grep "Location:" | awk '{print $2}')/torch/onnx
```
Make changes to the symbolic_opset.py to support scaled dot product attention.
In the directory, onnx_utils has an updated version for symbolic_opset14.py, which can be replaced with the file in the onnx directory.

Change the models by changing the Out_dir parameter in config/eval_commavq.py file.
The file converts a ckpt.pt checkpoint to ckpt.onnx Onnx file in the same directory and terms with a dummy input with OnnxRuntime.
```
```

```
$ python export_to_onnx.py
```

## Evaluating the Models with Onnx Files
Change the models by changing the Out_dir parameter in config/eval_commavq.py file.
```
$ python eval_commavq_ort.py
```

## Evaluating from checkpoints
Link to the checkpoints:
https://drive.google.com/drive/folders/1ir8xxywEh3fWsWhd66ARkIDFHPQzCoBU?usp=sharing

The drive link has three folders names, out-commavq, out-commavq_1,out-commavq_2. Each folder has a ckpt.pt and ckpt.onnx. To evaluate with these checkpoints, download the folder and copy it to the root directory of the repo.
In config/eval_commavq.py, change the out_dir accordingly.