Kraken: Inherently Parallel Transformers For Efficient Multi-Device Inference

Rohan Baskar Prabhakar
Princeton University
rohanbp@princeton.edu

Hengrui Zhang
Princeton University
hengrui.zhang@princeton.edu

David Wentzlaff
Princeton University
wentzlaf@princeton.edu

Abstract

Large Transformer networks are increasingly used in settings where low inference latency is necessary to enable new applications and improve the end-user experience. However, autoregressive inference is resource intensive and requires parallelism for efficiency. Parallelism introduces collective communication that is both expensive and represents a phase when hardware resources are underutilized. Towards mitigating this, Kraken is an evolution of the standard Transformer architecture that is designed to complement existing tensor parallelism schemes for efficient inference on multi-device systems. By introducing a fixed degree of intra-layer model parallelism, the architecture allows collective operations to be overlapped with compute, decreasing latency and increasing hardware utilization. When trained on OpenWebText, Kraken models reach a similar perplexity as standard Transformers while also preserving their language modeling capabilities as evaluated on the SuperGLUE benchmark. Importantly, when tested on multi-GPU systems using TensorRT-LLM engines, Kraken speeds up Time To First Token by a mean of 35.6% across a range of model sizes, context lengths, and degrees of tensor parallelism.

1 Introduction

Deep neural networks based on the Transformer architecture (51) have become the prevalent choice for a variety of tasks involving sequences, especially in natural language processing and computer vision (44) (7) (19). Their capabilities, particularly in language modeling, have been driven by a rapid increase in parameter count (7). Today's largest language models have up to a trillion parameters (21) and consequently demand more efficiency from the systems used to train and serve them. This has necessitated the need for many techniques and optimizations that focus on improving the performance of both algorithms and systems (47) (34) (17) (37) (12).

Large models are often used in interactive applications where latency is an important metric that dictates the quality of the end-user experience (28). A typical web search takes about 0.2 seconds but the Time To First Token (TTFT) for large models can be up to a few seconds (depending on context length, model size, and available hardware) (42). Additionally, because it is not always feasible to run models on local hardware, they are served to users via datacenters that use multi-device compute nodes, adding to latency constraints. Increasingly, language models are also used as intermediate steps in longer processes such as augmenting web searches or presenting the results of database queries (28). The rising prevalence of such multi-step applications makes reducing inference latency even more critical.

38th Conference on Neural Information Processing Systems (NeurIPS 2024).

Continuing this theme, this work focuses on reducing the latency cost of the collective operations introduced by tensor parallelism (49) in the forward pass. In particular, it introduces Kraken, a variation of the standard Transformer architecture (44)(7) that reduces the amount of inter-device communication and allows remaining collective operators to be overlapped with compute. Kraken models have a fixed degree of innate model parallelism that allows computational graphs on each device to run independently without having to wait for the results of collective operations. The architecture is designed to complement the topology of multi-device setups such as nodes in typical datacenters and DGX (13) systems. By designing the model architecture to account for characteristics of the hardware, our approach increases compute utilization and allows more efficient inference.

We evaluate the improvements Kraken offers over standard Transformers in two key aspects: model quality and inference latency. For the former, we train a series of Kraken models with varying degrees of parallelism and parameter count on OpenWebText (23) and compare them with the GPT-2 (44) family of models on the SuperGLUE suite of benchmarks (53). We then implement Kraken using the TensorRT-LLM library (15) and measure the Time To First Token (TTFT) given various model sizes and context lengths to illustrate the efficiency gains when collective operators are no longer on the critical path. We find that while maintaining the language modeling capabilities of standard Transformers, Kraken models speedup the Time To First Token (TTFT) by a geomean of 35.6% when tested across a range of model sizes, context lengths, and degrees of parallelism.

2 Background

2.1 Decoder-Only Transformer models

We will briefly discuss the forward pass of decoder-only Transformer (DTransformer) models that use self-attention mechanisms to perform language modeling (39) in order to motivate our approach. Given an input sequence x consisting of tokens belonging to a vocabulary V, such models return a probability distribution over the vocabulary that describes what the next token in x could be i.e., the model is trained to estimate $P(x[\ell + 1] \mid x[1:\ell])$ where ℓ is the initial length of x. To compute the output logits in the forward pass, x is converted to a sequence of embeddings that incorporate information about each token and its position in the sequence. These embeddings are used as input to a stack of Transformer layers one of which is depicted in Figure 1.

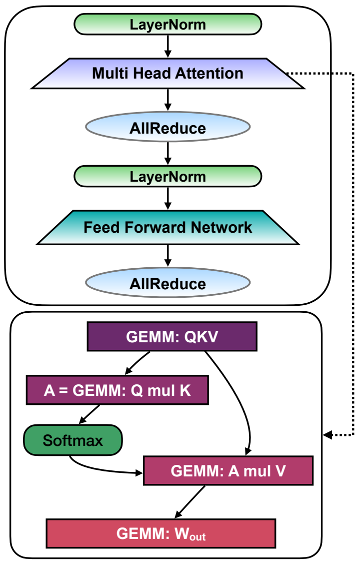

Figure 1: **One layer of a standard Transformer** consisting of Multi-Head Attention (also shown) followed by a FeedForward Network. Residual connections have been omitted.

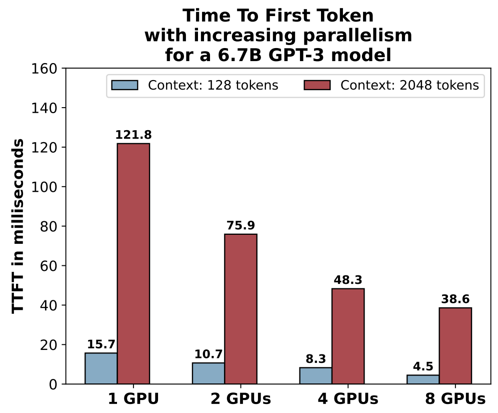

Figure 2: **Increasing the degree of tensor parallelism decreases the Time To First Token.** Even when weights and KV cache fit on device memory, parallelism can be worthwhile. These results are for a 6.7B parameter GPT-3 like model and were collected using TensorRT-LLM engines on our evaluation platform: an HGX A100 40GB system.

Such layers comprise the bulk of the compute in the model and consist of a Multi-Head Attention (MHA) block followed by a FeedForward Network (FFN or Multi-Layer Perceptron). The FFN typically consists of two linear transformations with a non-linear activation function in between. On the other hand, Multi-Head Attention (Figure 1) implements scaled dot-product attention i.e., each head computes $\texttt{Softmax}(\frac{QK^T}{\sqrt{h}})V$ where $Q, K, V \in \mathbb{R}^{h \times l}$. Here, h is the head dimension and l is the sequence length. The General Matrix Multiply (GEMM) W_{out} combines the outputs of the different heads. The activations of the last layer are used as input to the unembedding operation. Henceforth, we will refer to the GPT-2,3 (44)(7) like construction as the standard Transformer architecture. This variant uses MHA, has sequential Attention and FFN blocks, expands the embedding dimension from d to $4d$ in the hidden layer of the FFN, uses Gaussian Error Linear Unit (GELU) non-linearities, and places Layer Norm operators before the MHA and FFN i.e., is a Pre-LN Transformer.

A widely used optimization during inference involves caching the Key and Value matrices (from GEMM QKV in Figure 1) of each token in a KV cache that is stored in memory. This memoization has the effect of breaking up autoregressive inference (38)(58) into two distinct steps: 1) Prefill (when the first token and the KV cache are generated) and 2) Decode(for all subsequent tokens). Both these steps have distinct runtime characteristics with Prefill being more compute bound and Decode being more memory bandwidth bound (58). The KV cache entry for the next generated token is appended during each Decode step. Prefill, measured by TTFT, typically takes much longer than a single Decode step and will be the focus of this work.

2.2 Multi-Device systems

Given the extensive amount of compute and memory capacity required to efficiently serve large models, most widely used systems are node-based configurations where each node has a small number (between 4 and 16) of devices. These devices usually take the form of Graphics Processing Units (GPUs) or Tensor Processing Units (TPUs). The discussion in this work will focus on GPUs but we expect that our findings will also be of relevance to other choices of accelerators such as TPUs. Accessing data at levels of the system physically closer to compute cores such as scratchpad memory or caches is typically much faster and more efficient compared to accessing off-chip memory. Techniques like tiled matrix multiplication (20) and FlashAttention (17) account for this characteristic, considerably improving the runtime performance of implementations.

Devices within a node are configured in a topology and linked by interconnects such as Peripheral Component Interconnect Express (PCIe), NVLink, and NVSwitch. The different standards balance versatility and cost with performance. For instance, a topology that uses PCIe switches across some connections will have less overall bandwidth than a system that uses solely NVLink/NVSwitch. Inter-device communication primitives are provided to other software by libraries like NCCL (14) and RCCL (11). Communication is comparatively expensive and represents a phase in the forward pass where compute cores are mostly idle. This work strives to extend the IO-aware approach used by techniques like FlashAttention (17) towards the multi-device setting.

2.3 Tensor parallelism

Serving large models in multi-device settings requires parallelization schemes and strategies that partition the input sequence, model weights, activations, and/or incoming inference requests (49) (56) (35) (1) (60). In particular, model parallel (intra-operator) schemes fall into two broad categories: tensor parallelism and pipeline parallelism. Since Kraken models are designed to improve inference latency, this discussion will center around tensor parallelism.

There are several possible strategies (42) (55) to achieve distributed tensor parallelism in Transformer models but we will focus on the widely used scheme introduced by Shoeybi et al. (49) which is well-suited for multi-GPU settings. This scheme introduces two AllReduce operations per layer and takes advantage of the implicitly parallel nature of Multi-Head Attention with optimal partitioning of the FeedForward Network. In each layer, contiguous groups of Attention heads are placed across different devices and the W_{out} matrix that is used to combine the output of the different heads is partitioned across columns. The output of the MHA block is retrieved by reducing the local output of all devices; this introduces one AllReduce as shown in Figure 1. Similarly, in the FFN block, the W_1 weight matrix is partitioned across rows (using the notation where W_1 is multiplied by a column-vector of activations) and the W_2 weight matrix is partitioned across columns. The output

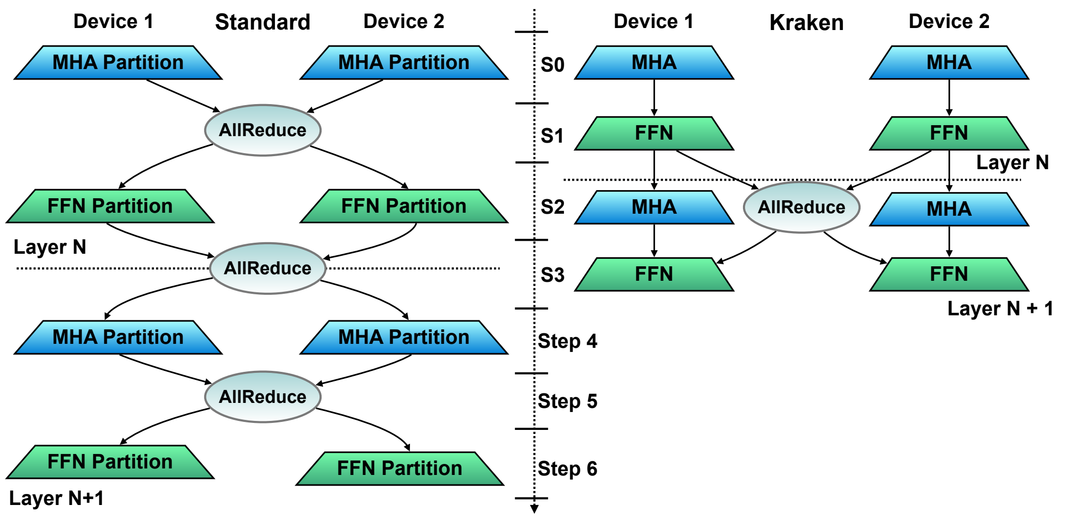

Figure 3: **Parallelizing two standard Transformer layers compared to executing two layers of a Kraken Transformer with 2-way parallelism**. Kraken Transformers have fewer AllReduce operations and these can be run concurrently with the Multi-Head Attention in the next layer. Step lengths are illustrative and not indicative of how much wall-clock time a particular operation might require.

of the FFN is computed by reducing the local output of all devices, thereby introducing another AllReduce operation. The objective of this work is to reduce and hide the runtime impact of these collectives introduced by tensor parallelism.

3 Kraken: Model architecture

3.1 Layer construction

The primary objective of Kraken is to preserve as much of the standard Transformer (GPT-2,3 like)(44)(7) architecture as possible while reducing the latency cost of the collective communication introduced by tensor parallelism. To achieve this, we allow each of the individual shards of a parallelized Transformer layer to behave as independent, smaller layers. More precisely, as depicted in Figure 3, instead of sharding the Multi-Head Attention and FeedForward Network blocks, each MHA and FFN block is replaced with a smaller, independent block. This introduces *a fixed degree of parallelism* that is chosen at the outset of training and accounts for the most common hardware deployment target. For example, if a model will be mostly served on nodes with eight GPUs each, a reasonable choice would be to use either 4-way or 8-way parallelism (depending on the size of the model). The former would be suitable for smaller models, allowing each node to serve two different inference requests at the same time.

Consequently, the only inter-device dependency is an AllReduce operation at the end of every layer. This collective represents the only interchange of activations between groups of sub-layers. Its output is used in the FFN block of the next layer and not in the MHA block. As shown in Figure 3, this allows for overlapping the compute in the MHA block with the AllReduce, effectively removing most inter-device communication from the critical path. Much like how positional embeddings are added to the token embeddings prior to the first layer (39), we chose to use element-wise addition to combine the outputs of the various sub-layers from the previous layer. This occurs prior to the Layer Norm before the FFN. The forward pass of each sub-layer, including residual connections, is also described in Algorithm 1. All pretrained models in our evaluation use the GELU activation function in the FFN. In initial experiments that explored different constructions, we scaled the weights of the residual layer by $\frac{1}{\sqrt{L*N}}$ using a similar line of reasoning as that used by Radford et al. (44). Here, L is the number of layers, and N is the degree of parallelism. We did not remove this initialization scheme in subsequent experiments even though accumulation along residual connections was limited to groups of L sub-layers.

Algorithm 1: Kraken Sub-Layer: Forward Pass

```
Input: $x \in \mathbb{R}^{l \times d}$
Output: $y \in \mathbb{R}^{l \times d}$
1 $residual = x$
  /* The first layer replaces the AllReduce with the identity operator */
2 $y =$ AllReduce$(x)$
3 $x =$ LayerNorm$(x)$
4 $x = residual +$ MultiHeadAttention$(x)$
5 $residual = x$
  /* The output of the AllReduce is used only here allowing it to be overlapped
     with Attention */
6 $x =$ LayerNorm$(x + y)$
7 $y = residual +$ FeedForwardNetwork$(x)$
8 return $y$
```

The token and positional embeddings are shared across all sub-layers of the first layer i.e., there is still one set of embeddings to maintain compatibility with weight tying (27) (43). After the last layer, we combine the outputs of the different sub-layers using a linear transformation with weights $W_{concat} \in \mathbb{R}^{d*N \times d}$ where d is the embedding dimension and N is the degree of parallelism. The output of this transformation is used as input to the unembedding. Depending on the implementation, this linear layer introduces the only blocking collective in the computational graph.

3.2 Deriving model configurations for a fixed parameter budget

Increasing the degree of parallelism while keeping other hyperparameters like the embedding dimension constant will increase the parameter count. Instead, using a configuration of a standard Transformer as the basis, we make the following two modifications to derive a Kraken configuration that has approximately the same number of parameters:

- First, the hidden state expansion in the FFN is reduced from $4d$ to $2d$ where d is the embedding/model dimension.
- Given the number of parameters P, degree of parallelism N, number of layers L, and vocabulary size V, we derive a closed form expression for P and solve for d i.e., $P = V * d + \Sigma_{i=1}^{L} N * ((3*(d \times d) + (d \times d)) + 2*(d \times 2d))$ where the term $((3*(d \times d) + (d \times d))$ comes from MHA and $2 * (d \times 2d))$ comes from the FFN.

4 Evaluation

4.1 Model configurations and perplexity

To evaluate language modeling performance, we train a series of models up to 761 million parameters large and with varying degrees of parallelism on OpenWebText (23). This allows us to compare the performance of the Kraken architecture with the GPT-2 (44) family of models. Because of limited access to compute, we do not exhaustively search for hyperparameters and stop training at 150 billion tokens in contrast to the about 300 billion tokens that language models of such sizes are typically trained for (44)(24). Table 1 details the embedding dimensions, number of layers, and other hyperparameters for each configuration. For Kraken configurations, the number of Attention heads in each layer is summed across all sub-layers. The context length was set at 1024 tokens. Models similar in size were trained using the same learning rate schedule and for the same number of gradient steps. More details about the training setup, required compute, and how each configuration was derived can be found in Appendix A.1. Pertinent code including the TensorRT-LLM implementation is available at `https://github.com/rohan-bp/kraken`.

Perplexity measurements for GPT-2 models are provided to add context to the results with the caveat that all Kraken models were trained on OpenWebText but GPT-2 models were trained on the closed-source WebText dataset. We would expect that all things equal, Kraken models will have lower perplexity unless the GPT-2 models were subsequently fine-tuned on OpenWebText. Nonetheless,

Table 1: **Model configurations and perplexity on OpenWebText for Kraken models compared to similarly sized GPT-2 models.** Lower perplexity is better.

Model	Layers	Embedding Dimension	Attention Heads	Total Params.	Validation Perplexity
GPT-2	12	768	12	117M	20.64
Kraken 2-way	12	678	12	124M	18.89
Kraken 4-way	12	504	12	124.5M	18.56
Kraken 6-way	12	418	12	123.2M	19.22
GPT-2 Medium	24	1024	16	345M	14.87
Kraken 2-way	24	888	16	350M	14.40
Kraken 4-way	24	644	16	353.4M	14.71
GPT-2 Large	24	1280	20	762M	13.69
Kraken 4-way	24	960	16	761M	13.09

Table 2: **Zero-Shot performance on SuperGLUE.** ReCoRD uses the F1 score as the evaluation metric. All other benchmarks use accuracy.

Model	BoolQ	RTE	CB	COPA	ReCoRD	WIC	WSC	MultiRC	Average
GPT-2	48.38	51.99	41.07	62.0	71.07	49.53	43.27	53.47	52.6
Kraken 124M (2-way)	53.85	54.15	44.64	68.0	72.42	49.53	36.54	56.89	54.5
Kraken 124M (4-way)	50.15	53.79	41.07	67.0	71.47	50.31	37.5	57.16	53.56
Kraken 124M (6-way)	47.92	56.68	8.93	69.0	70.69	50.16	53.85	53.82	51.38
GPT-2 Medium	58.53	53.07	42.86	68.0	79.43	50.0	41.35	52.58	55.73
Kraken 355M (2-way)	60.06	51.62	41.07	69.0	80.04	50.0	35.58	56.91	55.54
Kraken 355M (4-way)	61.68	55.23	35.71	72.0	79.01	50.0	36.54	57.2	55.92
GPT-2 Large	60.55	52.71	41.07	72.0	81.95	49.69	44.23	48.56	56.34
Kraken 760M (4-way)	60.58	49.1	10.71	73.0	82.04	50.0	36.54	51.65	51.7

even when trained over a smaller number of tokens, Kraken models reach a similarly low perplexity as standard Transformers.

4.2 Performance On SuperGLUE

We used the SuperGLUE benchmark suite (53) to evaluate performance on language tasks. All benchmarks were scored on accuracy except for ReCoRD which uses the F1 Score instead. No finetuning or training was performed for any combination of model and benchmark. Table 2 contains results for Zero-Shot performance and Table 3 presents performance in the Three-Shot setting. Scores were calculated using the Language Model Evaluation Harness (2) with the default choice of prompts and scoring metrics for option `lm-eval-SuperGLUE v1`. As conveyed by these results, Kraken largely preserves the language modeling capabilities of the standard Transformer architecture. We expect that the gap between standard Transformers on language tasks will close further if we train the models on higher quality data and using optimal choices for the various hyperparameters.

4.3 Evaluation platform

To measure improvements in inference latency, we used the TensorRT-LLM (15) library to create engines and compare Kraken models with other widely used dense model architectures. The library provides an interface to define popular Transformer models and bundles a collection of kernels,

Table 3: **Three-Shot performance on SuperGLUE.** ReCoRD uses the F1 score as the evaluation metric. All other benchmarks use accuracy.

Model	BoolQ	RTE	CB	COPA	ReCoRD	WIC	WSC	MultiRC	Average
GPT-2	53.76	47.65	42.86	60.0	70.12	50.0	51.92	50.62	53.37
Kraken 124M (2-way)	55.14	48.74	44.64	61.0	68.83	49.69	47.12	50.56	53.22
Kraken 124M (4-way)	59.05	44.04	48.21	64.0	69.94	47.49	51.92	54.17	54.85
Kraken 124M (6-way)	54.28	46.93	42.86	63.0	69.17	49.37	44.23	51.96	52.72
GPT-2 Medium	60.58	47.65	44.64	68.0	78.35	47.65	50.0	53.22	56.26
Kraken 355M (2-way)	50.73	53.07	37.5	68.0	79.01	49.06	66.35	51.34	56.88
Kraken 355M (4-way)	61.8	51.99	50.0	73.0	77.97	47.49	44.23	54.17	57.58
GPT-2 Large	60.58	51.99	39.29	70.0	81.05	47.81	60.58	51.53	57.85
Kraken 760M (4-way)	55.75	53.79	42.86	68.0	81.03	45.77	62.5	53.47	57.9

Table 4: **Configurations for the different model engines used to compare TTFT.** For models of similar sizes, hyperparameters are shared for the GPT-like and Parallel Attention + FeedForward variants. 4-way denotes Kraken configurations used when evaluating tensor parallelism across 4 devices and likewise for 8-way.

Model Size	Layers	dModel	Params. Per Layer	Attention Heads	Kraken 4-way dModel	4-way Params. Per Layer	4-way Attention Heads	Kraken 8-way dModel	8-way Params. Per Layer	8-way Attention Heads
1.3B	24	2048	50.3M	16	1248	49.9M	12	960	59.0M	10
6.7B	32	4096	201.3M	32	2496	199.4M	24	1920	235.9M	20
13B	40	5140	317.0M	40	3120	311.5M	30	2304	339.7M	32
65B	80	8192	805.3M	64	4992	797.4M	39	3648	851.7M	38
175B	96	12288	1.81B	96	7424	1.76B	58	5472	1.92B	57

plugins, and other optimizations that can be used to create efficient TensorRT engines (containing model weights) and serve them on systems with GPUs. All experiments were conducted on a 8 x A100 GPU machine with NVSwitch and 40GB of HBM memory per GPU.

4.4 Speedup in Time To First Token

For comparisons with other model architectures, we build engines for standard, GPT-like configurations and GPT-J (54) like configurations as detailed in Table 4. The latter serves to contrast our approach with one that runs the FFN in parallel with the Attention (10) (54). Such parallel layers require only one AllReduce in a layer but unlike Kraken, the collective cannot be overlapped with compute. For similarly sized models, the only difference is the embedding dimension for the Kraken models. All configurations follow the convention where `dModel` is divided by the number of Attention heads to calculate the size of each head. Other configuration parameters such as the number of layers, maximum context length, and vocabulary size were the same. The Attention heads are per sub-layer for Kraken models and parameter counts do not include biases and layer normalization.

There are two sets of engines, one for 4-way tensor parallelism and another for 8-way parallelism. We also did not follow the earlier convention 3.2 of solving for the embedding dimension of a Kraken model precisely. This is because the available optimizations in TensorRT-LLM are compatible only with specific Attention head dimensions. To account for this, we handpicked embedding dimensions that, given an equivalent GPT-like configuration, have about the same number of parameters while still being compatible with the available kernels. This was necessary for a fair evaluation but in practice, the chosen embedding dimension should also account for the performance

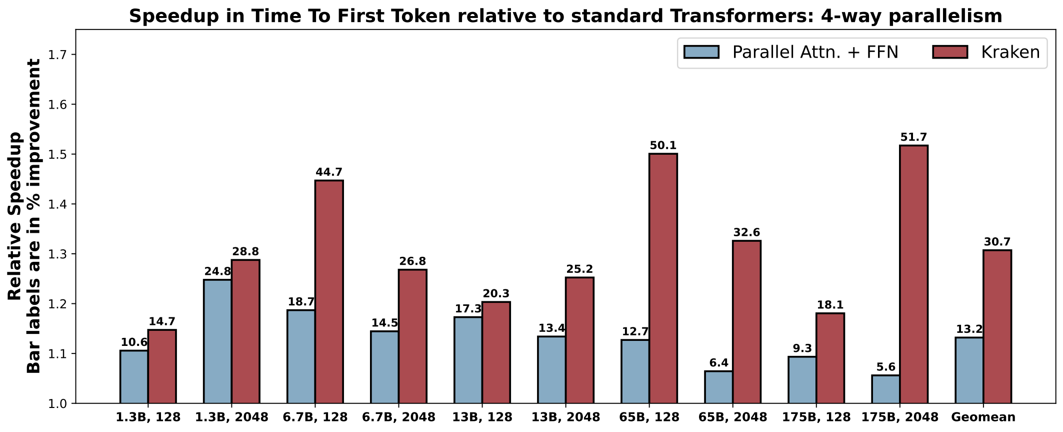

Figure 4: **Speedup in Time To First Token over standard Transformers on a system that uses NVSwitch and with 4-way parallelism**. x-axis labels denote the size of the model followed by the context length. Bar labels are in percentage improvement.

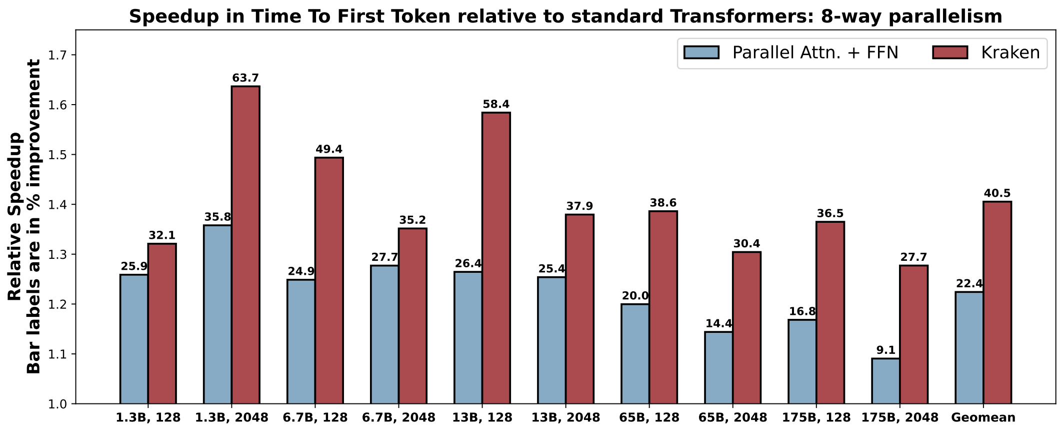

Figure 5: **Speedup in Time To First Token over standard Transformers on a system that uses NVSwitch and with 8-way parallelism**. x-axis labels denote the size of the model followed by the context length. Bar labels are in percentage improvement.

of the computational kernels it would map to. Anthony et al. (4) discuss this, showing how model hyperparameters can affect GEMM performance and consequently training efficiency.

Figure 4 shows the speedup in latency compared to GPT-like models on our evaluation platform and with 4-way parallelism while Figure 5 depicts improvements for 8-way parallelism. Across a range of model sizes and context lengths of 128 and 2048 tokens, using Kraken models can improve inference latency by anywhere from 10.9% to 63.7%. These results are end-to-end and include the extra fully-connected layer required by Kraken models. All results are normalized to the equivalent latency for a similar GPT-3 like model and engines were generated with random weights and *fp16* precision. Latency in terms of milliseconds is reported in Appendix A.3. Importantly, our evaluation platform uses NVSwitch and consequently has considerable inter-device bandwidth (600 GB/s) which means communication can be relatively inexpensive. We expect that these gains will be more pronounced on systems with less performant interconnects.

A more detailed evaluation on precisely which operators contribute to overall runtime is presented in Section 4.5. In order to take advantage of the concurrent communication and compute that Kraken allows, we extended TensorRT-LLM with the Overlap plugin that is described in Appendix A.2.

Each configuration was evaluated using the *gptSessionBenchmark* that is available as part of TensorRT-LLM. We enabled the default set of optimizations which include GPT Attention plugin, Remove Input Padding, and GEMM plugin. Experiments were run with a batch size of 1 but engines were built with a maximum batch size of 4, vocabulary size of 51,200 tokens, maximum input context length of 2048, and maximum output length of 4096.

4.5 Runtime characterization

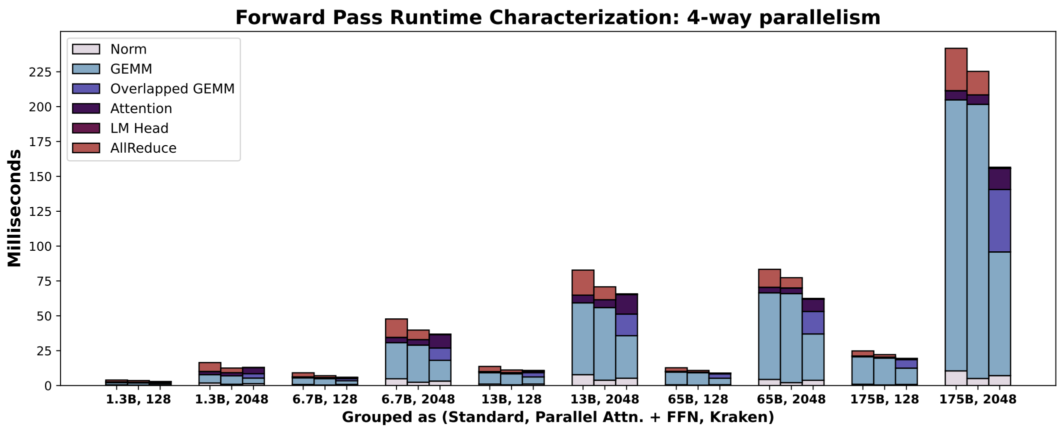

Figure 6: **Runtime characterization: 4-way parallelism.** For each cluster on the x-axis, labels denote the size of model followed by the context length.

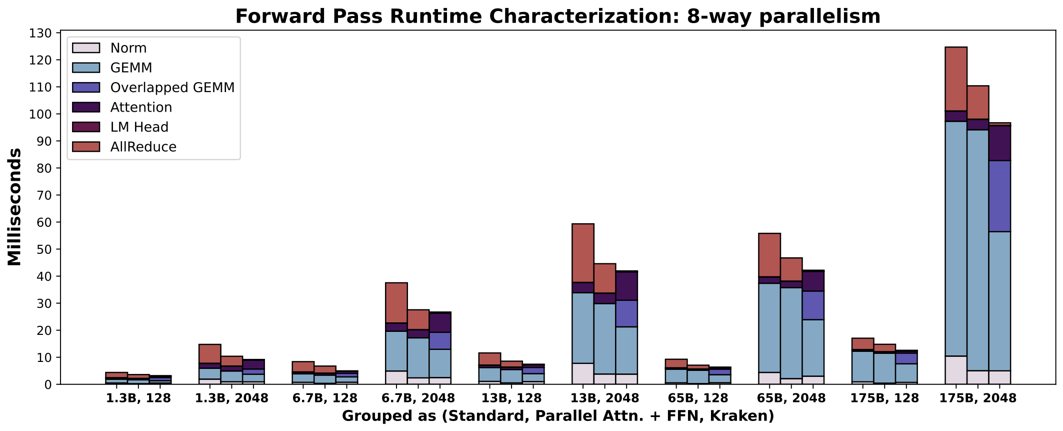

Figure 7: **Runtime characterization: 8-way parallelism.** For each cluster on the x-axis, labels denote the size of model followed by the context length.

To provide more context to the performance gains presented in Figures 4 and 5, we characterize the entire forward pass using the same experimental setting as Section 4.4. The runtime profiles presented in Figures 6 and 7 were obtained using the profiler built into *gptSessionBenchmark*. Across all model sizes and both context lengths, we find that a significant proportion of time is spent in the `AllReduce`. As expected, this proportion increases for 8-way parallelism compared to 4-way parallelism since each device works on a smaller fraction of the compute.

In some cases, such as for 175B models at context length 2048, some of the performance gains come from the GEMMs requiring significantly more time for the GPT and GPT-J like configurations. Despite these performance anomalies, in general, Kraken models spend a much smaller fraction of runtime in inter-device communication leading to considerable performance and efficiency improvements. Note that the cost of the memory copies and synchronization necessary for the Overlap plugin (Appendix A.2) are implicitly included in the `Overlapped GEMM` fraction. More precisely, `Overlapped GEMM` is the time spent computing GEMM W_{out}, GEMM QKV, and other operations in the Overlap plugin.

5 Discussion and Limitations

Need for pretraining: Currently, our approach requires training Kraken models from scratch which can be very resource intensive. Consequently, developing techniques to distill learned weights from existing models possibly as part of the weight initialization scheme is a promising area of future work. We also do not compare with more state-of-the-art Transformer training recipes or alternatives because of limited access to GPU compute. Training larger, compute-optimal (25) Kraken models on higher-quality datasets will permit evaluations on newer, more complex language modeling benchmarks. The fixed degree of model parallelism also places restrictions on the optimal choice of hardware to run a model on. For example, if we tried to deploy a model with 4-way parallelism on a system with two or six devices, we would either need to run groups of two sub-layers together or introduce more collectives. Either approach might negate most of the latency gains offered in the first place.

Compatibility with standard Transformer improvements: Since Kraken models replicate the Attention block by the degree of tensor parallelism, the size of the KV cache will also be larger than that of an equivalent Transformer model. One way to mitigate the increased memory consumption would be to replace Multi-Head Attention with either Multi-Query Attention (47) or Grouped Query Attention (3). Similarly, many other improvements to the standard Transformer architecture are also applicable to Kraken models such as RoPE embeddings (50) and RMSNorm (57). The architecture is also compatible with "drop-in" replacements for Attention that are more efficient because of time-complexity and/or sparsity (31) (9). Furthermore, we expect that the notion of parallelizing individual layers of the model will also prove useful in large deep learning models that use constructions other than Attention or MLPs (24). Incorporating a communication-aware approach to Neural Architecture Search (8) for Transformers is another promising area of future work.

Applicability of existing optimizations: We expect that Kraken will be readily compatible with many existing system-aware techniques used in deploying standard Transformers (56) (32) (1) (35). For example, it provides an extra degree of freedom to the various partitioning strategies proposed by Pope et al. (42). It is also compatible with techniques like FlashAttention (17), Speculative Decoding (34), fusing Attention with the FFN (36), and PagedAttention (32).

Mixture-of-Experts and hybrid models: By virtue of its architecture, Kraken evokes a comparison to Mixture-of-Experts (MoE) architectures such as the SwitchTransformer (21)(48). Layers in current MoE models have a single Attention block and use a learned Router to direct each token to one of a set of FeedForward Networks. Because the forward pass activates only a fixed fraction of the parameters, MoE models can be much larger than dense Transformers while maintaining the computational profile of inference. Nonetheless, serving them efficiently in multi-device settings is a challenge: MoEs require dynamic routing and suffer from load balancing issues. For instance, Huang et al. (26) find that the required All-to-All collectives can comprise a significant fraction of inference latency. An interesting direction of future work would be to incorporate an inter-device IO-aware approach in the construction of MoE models. In addition, recent work (52) has found that hybrid architectures that combine Multi-Head Attention, State Space Models (SSMs) (24), and FFNs in the same model perform better than models that use solely SSMs or MHA for the sequence-to-sequence transformation. Similarly, Kraken layers can be combined with standard Transformer layers and pure SSM layers to develop high-quality language models that are also efficient to run on hardware.

6 Acknowledgements

This material is based on research sponsored by the Air Force Research Laboratory (AFRL) and the Defense Advanced Research Projects Agency (DARPA) under agreement No. FA8650-18-2-7862. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright notation thereon. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the Air Force Research Laboratory (AFRL), the Defense Advanced Research Projects Agency (DARPA), or the U.S. Government. We thank Princeton Research Computing for their assistance in setting up and maintaining the necessary compute infrastructure. We thank Ashwinee Panda for feedback and aid in brainstorming the title. We also thank Jens Tuyls for the helpful discussions.

References

[1] Amey Agrawal, Nitin Kedia, Ashish Panwar, Jayashree Mohan, Nipun Kwatra, Bhargav Gulavani, Alexey Tumanov, and Ramachandran Ramjee. Taming Throughput-Latency Tradeoff in LLM Inference with Sarathi-Serve. In *18th USENIX Symposium on Operating Systems Design and Implementation (OSDI 24)*, pages 117–134, Santa Clara, CA, 2024. USENIX Association.

[2] Eleuther AI. Language Model Evaluation Harness. `https://github.com/EleutherAI/lm-evaluation-harness/`, 2022.

[3] Joshua Ainslie, James Lee-Thorp, Michiel de Jong, Yury Zemlyanskiy, Federico Lebron, and Sumit Sanghai. GQA: Training Generalized Multi-Query Transformer Models from Multi-Head Checkpoints. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 4895–4901. Association for Computational Linguistics, December 2023.

[4] Quentin Anthony, Jacob Hatef, Deepak Narayanan, Stella Biderman, Stas Bekman, Junqi Yin, Aamir Shafi, Hari Subramoni, and Dhabaleswar Panda. The Case for Co-Designing Model Architectures with Hardware. In *Proceedings of the 53rd International Conference on Parallel Processing*, ICPP '24, page 84–96, New York, NY, USA, 2024. Association for Computing Machinery.

[5] Roy Bar Haim, Ido Dagan, Bill Dolan, Lisa Ferro, Danilo Giampiccolo, Bernardo Magnini, and Idan Szpektor. The second PASCAL recognising textual entailment challenge. In *Proceedings of the Second PASCAL Challenges Workshop on Recognising Textual Entailment*, 2006.

[6] Luisa Bentivogli, Ido Dagan, Hoa Trang Dang, Danilo Giampiccolo, and Bernardo Magnini. The fifth PASCAL recognizing textual entailment challenge. In *Proceedings of the Text Analysis Conference*, 2009.

[7] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel Ziegler, Jeffrey Wu, Clemens Winter, Chris Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language Models are Few-Shot Learners. In *Advances in Neural Information Processing Systems*, volume 33, pages 1877–1901. Curran Associates, Inc., 2020.

[8] Krishna Teja Chitty-Venkata, Murali Emani, Venkatram Vishwanath, and Arun K. Somani. Neural Architecture Search for Transformers: A Survey. *IEEE Access*, 10:108374–108412, 2022.

[9] Krzysztof Marcin Choromanski, Valerii Likhosherstov, David Dohan, Xingyou Song, Andreea Gane, Tamas Sarlos, Peter Hawkins, Jared Quincy Davis, Afroz Mohiuddin, Lukasz Kaiser, David Benjamin Belanger, Lucy J Colwell, and Adrian Weller. Rethinking Attention with Performers. In *International Conference on Learning Representations*, 2021.

[10] Aakanksha Chowdhery, Sharan Narang, Jacob Devlin, Maarten Bosma, Gaurav Mishra, Adam Roberts, Paul Barham, Hyung Won Chung, Charles Sutton, Sebastian Gehrmann, Parker Schuh, Kensen Shi, Sasha Tsvyashchenko, Joshua Maynez, Abhishek Rao, Parker Barnes, Yi Tay, Noam Shazeer, Vinodkumar Prabhakaran, Emily Reif, Nan Du, Ben Hutchinson, Reiner Pope, James Bradbury, Jacob Austin, Michael Isard, Guy Gur-Ari, Pengcheng Yin, Toju Duke, Anselm Levskaya, Sanjay Ghemawat, Sunipa Dev, Henryk Michalewski, Xavier Garcia, Vedant Misra, Kevin Robinson, Liam Fedus, Denny Zhou, Daphne Ippolito, David Luan, Hyeontaek Lim, Barret Zoph, Alexander Spiridonov, Ryan Sepassi, David Dohan, Shivani Agrawal, Mark Omernick, Andrew M. Dai, Thanumalayan Sankaranarayana Pillai, Marie Pellat, Aitor Lewkowycz, Erica Moreira, Rewon Child, Oleksandr Polozov, Katherine Lee, Zongwei Zhou, Xuezhi Wang, Brennan Saeta, Mark Diaz, Orhan Firat, Michele Catasta, Jason Wei, Kathy Meier-Hellstern, Douglas Eck, Jeff Dean, Slav Petrov, and Noah Fiedel. PaLM: Scaling Language Modeling with Pathways. *Journal of Machine Learning Research*, 24(240):1–113, 2023.

[11] AMD Corp. ROCm Collective Communication Library. https://rocm.docs.amd.com/projects/rccl/en/latest/, 2024.

[12] NVIDIA Corp. H100 Transformer Engine Supercharges AI Training. https://blogs.nvidia.com/blog/h100-transformer-engine/, 2022.

[13] NVIDIA Corp. NVIDIA DGX H100. https://resources.nvidia.com/en-us-dgx-systems/ai-enterprise-dgx, 2023.

[14] NVIDIA Corp. NVIDIA Collective Communications Library. https://developer.nvidia.com/nccl, 2024.

[15] NVIDIA Corp. TensorRT-LLM. https://github.com/NVIDIA/TensorRT-LLM, 2024.

[16] Ido Dagan, Oren Glickman, and Bernardo Magnini. The PASCAL recognising textual entailment challenge. In *Proceedings of the First International Conference on Machine Learning Challenges: Evaluating Predictive Uncertainty Visual Object Classification, and Recognizing Textual Entailment*, MLCW'05, page 177–190, Berlin, Heidelberg, 2005. Springer-Verlag.

[17] Dao, Tri and Fu, Dan and Ermon, Stefano and Rudra, Atri and Ré, Christopher. FlashAttention: Fast and Memory-Efficient Exact Attention with IO-Awareness. In *Advances in Neural Information Processing Systems*, volume 35, pages 16344–16359. Curran Associates, Inc., 2022.

[18] Marie-Catherine De Marneffe, Mandy Simons, and Judith Tonhauser. The CommitmentBank: Investigating projection in naturally occurring discourse. In *Proceedings of Sinn und Bedeutung 23*, 2019.

[19] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale. In *International Conference on Learning Representations*, 2021.

[20] Kayvon Fatahalian, Jeremy Sugerman, and Pat Hanrahan. Understanding the Efficiency of GPU Algorithms for Matrix-Matrix Multiplication. In *Proceedings of the ACM SIGGRAPH/EUROGRAPHICS conference on Graphics hardware*, pages 133–137, 2004.

[21] William Fedus, Barret Zoph, and Noam Shazeer. Switch Transformers: Scaling to Trillion Parameter Models with Simple and Efficient Sparsity. *Journal of Machine Learning Research*, 23(120):1–39, 2022.

[22] Danilo Giampiccolo, Bernardo Magnini, Ido Dagan, and Bill Dolan. The third PASCAL recognizing textual entailment challenge. In *Proceedings of the ACL-PASCAL workshop on textual entailment and paraphrasing*, pages 1–9. Association for Computational Linguistics, 2007.

[23] Aaron Gokaslan, Vanya Cohen, Ellie Pavlick, and Stefanie Tellex. OpenWebText Corpus. http://Skylion007.github.io/OpenWebTextCorpus, 2019.

[24] Albert Gu and Tri Dao. Mamba: Linear-Time Sequence Modeling with Selective State Spaces. In *First Conference on Language Modeling*, 2024.

[25] Jordan Hoffmann, Sebastian Borgeaud, Arthur Mensch, Elena Buchatskaya, Trevor Cai, Eliza Rutherford, Diego de Las Casas, Lisa Anne Hendricks, Johannes Welbl, Aidan Clark, Tom Hennigan, Eric Noland, Katie Millican, George van den Driessche, Bogdan Damoc, Aurelia Guy, Simon Osindero, Karen Simonyan, Erich Elsen, Oriol Vinyals, Jack W. Rae, and Laurent Sifre. Training Compute-Optimal Large Language Models. In *Proceedings of the 36th International Conference on Neural Information Processing Systems*, NIPS '22, Red Hook, NY, USA, 2024. Curran Associates Inc.

[26] Haiyang Huang, Newsha Ardalani, Anna Sun, Liu Ke, Hsien-Hsin S. Lee, Anjali Sridhar, Shruti Bhosale, Carole-Jean Wu, and Benjamin Lee. Towards MoE Deployment: Mitigating Inefficiencies in Mixture-of-Expert (MoE) Inference. *arXiv preprint arXiv:2303.06182*, 2023.

[27] Hakan Inan, Khashayar Khosravi, and Richard Socher. Tying Word Vectors and Word Classifiers: A Loss Framework for Language Modeling. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*, 2017.

[28] Jean Kaddour, Joshua Harris, Maximilian Mozes, Herbie Bradley, Roberta Raileanu, and Robert McHardy. Challenges and Applications of Large Language Models. *arXiv preprint arXiv:2307.10169*, 2023.

[29] Andrej Karpathy. nanoGPT. `https://github.com/karpathy/nanoGPT`, 2022.

[30] Daniel Khashabi, Snigdha Chaturvedi, Michael Roth, Shyam Upadhyay, and Dan Roth. Looking beyond the surface: A challenge set for reading comprehension over multiple sentences. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pages 252–262, 2018.

[31] Nikita Kitaev, Lukasz Kaiser, and Anselm Levskaya. Reformer: The Efficient Transformer. In *International Conference on Learning Representations*, 2020.

[32] Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph Gonzalez, Hao Zhang, and Ion Stoica. Efficient Memory Management for Large Language Model Serving with PagedAttention. In *Proceedings of the 29th Symposium on Operating Systems Principles*, SOSP '23, page 611–626, New York, NY, USA, 2023. Association for Computing Machinery.

[33] Hector J Levesque, Ernest Davis, and Leora Morgenstern. The Winograd schema challenge. In *AAAI Spring Symposium: Logical Formalizations of Commonsense Reasoning*, volume 46, page 47, 2011.

[34] Yaniv Leviathan, Matan Kalman, and Yossi Matias. Fast Inference from Transformers via Speculative Decoding. In *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pages 19274–19286. PMLR, 23–29 Jul 2023.

[35] Shenggui Li, Fuzhao Xue, Chaitanya Baranwal, Yongbin Li, and Yang You. Sequence Parallelism: Long Sequence Training from System Perspective. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 2391–2404, Toronto, Canada, July 2023. Association for Computational Linguistics.

[36] Hao Liu and Pieter Abbeel. Blockwise Parallel Transformers for Large Context Models. In *Advances in Neural Information Processing Systems*, volume 36, pages 8828–8844. Curran Associates, Inc., 2023.

[37] Hao Liu, Matei Zaharia, and Pieter Abbeel. RingAttention with Blockwise Transformers for Near-Infinite Context. In *The Twelfth International Conference on Learning Representations*, 2024.

[38] Pratyush Patel, Esha Choukse, Chaojie Zhang, Aashaka Shah, Íñigo Goiri, Saeed Maleki, and Ricardo Bianchini. Splitwise: Efficient Generative LLM Inference Using Phase Splitting. In *2024 ACM/IEEE 51st Annual International Symposium on Computer Architecture (ISCA)*, pages 118–132, 2024.

[39] Mary Phuong and Marcus Hutter. Formal Algorithms for Transformers. *arXiv preprint arXiv:2207.09238*, 2022.

[40] Mohammad Taher Pilehvar and Jose Camacho-Collados. WiC: The Word-in-Context Dataset for Evaluating Context-Sensitive Meaning Representations. In *Proceedings of the North American Chapter of the Association for Computational Linguistics: Human Language Technology*, 2019.

[41] Adam Poliak, Aparajita Haldar, Rachel Rudinger, J. Edward Hu, Ellie Pavlick, Aaron Steven White, and Benjamin Van Durme. Collecting Diverse Natural Language Inference Problems for Sentence Representation Evaluation. In *Proceedings of Empirical Methods in Natural Language Processing*, 2018.

[42] Reiner Pope, Sholto Douglas, Aakanksha Chowdhery, Jacob Devlin, James Bradbury, Jonathan Heek, Kefan Xiao, Shivani Agrawal, and Jeff Dean. Efficiently Scaling Transformer Inference. In *Proceedings of Machine Learning and Systems*, volume 5, pages 606–624, 2023.

[43] Ofir Press and Lior Wolf. Using the Output Embedding to Improve Language Models. In *Conference of the European Chapter of the Association for Computational Linguistics*, 2016.

[44] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019.

[45] Melissa Roemmele, Cosmin Adrian Bejan, and Andrew S. Gordon. Choice of plausible alternatives: An evaluation of commonsense causal reasoning. In *2011 AAAI Spring Symposium Series*, 2011.

[46] Rachel Rudinger, Jason Naradowsky, Brian Leonard, and Benjamin Van Durme. Gender Bias in Coreference Resolution. In *Proceedings of the North American Chapter of the Association for Computational Linguistics: Human Language Technology*, 2018.

[47] Noam Shazeer. Fast Transformer Decoding: One Write-Head is All You Need. *arXiv preprint arXiv:1911.02150*, 2019.

[48] Noam Shazeer, *Azalia Mirhoseini, *Krzysztof Maziarz, Andy Davis, Quoc Le, Geoffrey Hinton, and Jeff Dean. Outrageously Large Neural Networks: The Sparsely-Gated Mixture-of-Experts Layer. In *International Conference on Learning Representations*, 2017.

[49] Mohammad Shoeybi, Mostofa Patwary, Raul Puri, Patrick LeGresley, Jared Casper, and Bryan Catanzaro. Megatron-LM: Training Multi-Billion Parameter Language Models Using Model Parallelism. *arXiv preprint arXiv:1909.08053*, 2020.

[50] Jianlin Su, Murtadha Ahmed, Yu Lu, Shengfeng Pan, Wen Bo, and Yunfeng Liu. Roformer: Enhanced transformer with rotary position embedding. *Neurocomputing*, 568:127063, 2024.

[51] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. Attention is All you Need. In *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017.

[52] Roger Waleffe, Wonmin Byeon, Duncan Riach, Brandon Norick, Vijay Korthikanti, Tri Dao, Albert Gu, Ali Hatamizadeh, Sudhakar Singh, Deepak Narayanan, Garvit Kulshreshtha, Vartika Singh, Jared Casper, Jan Kautz, Mohammad Shoeybi, and Bryan Catanzaro. An Empirical Study of Mamba-based Language Models. *arXiv preprint arXiv:2406.07887*, 2024.

[53] Alex Wang, Yada Pruksachatkun, Nikita Nangia, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R. Bowman. SuperGLUE: A Stickier Benchmark for General-Purpose Language Understanding Systems. *CoRR*, abs/1905.00537, 2019.

[54] Ben Wang and Aran Komatsuzaki. GPT-J-6B: A 6 Billion Parameter Autoregressive Language Model. `https://github.com/kingoflolz/mesh-transformer-jax`, May 2021.

[55] Shibo Wang, Jinliang Wei, Amit Sabne, Andy Davis, Berkin Ilbeyi, Blake Hechtman, Dehao Chen, Karthik Srinivasa Murthy, Marcello Maggioni, Qiao Zhang, Sameer Kumar, Tongfei Guo, Yuanzhong Xu, and Zongwei Zhou. Overlap Communication with Dependent Computation via Decomposition in Large Deep Learning Models. In *Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 1*, ASPLOS 2023, page 93–106, New York, NY, USA, 2022. Association for Computing Machinery.

[56] Gyeong-In Yu, Joo Seong Jeong, Geon-Woo Kim, Soojeong Kim, and Byung-Gon Chun. Orca: A distributed serving system for Transformer-Based generative models. In *16th USENIX Symposium on Operating Systems Design and Implementation (OSDI 22)*, pages 521–538, 2022.

[57] Biao Zhang and Rico Sennrich. Root Mean Square Layer Normalization. In *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019.

[58] Hengrui Zhang, August Ning, Rohan Baskar Prabhakar, and David Wentzlaff. LLMCompass: Enabling Efficient Hardware Design for Large Language Model Inference. In *Proceedings of the 51st Annual International Symposium on Computer Architecture*, 2024.

[59] Sheng Zhang, Xiaodong Liu, Jingjing Liu, Jianfeng Gao, Kevin Duh, and Benjamin Van Durme. ReCoRD: Bridging the Gap between Human and Machine Commonsense Reading Comprehension. *arXiv preprint 1810.12885*, 2018.

[60] Kan Zhu, Yilong Zhao, Liangyu Zhao, Gefei Zuo, Yile Gu, Dedong Xie, Yufei Gao, Qinyu Xu, Tian Tang, Zihao Ye, Keisuke Kamahori, Chien-Yu Lin, Stephanie Wang, Arvind Krishnamurthy, and Baris Kasikci. NanoFlow: Towards Optimal Large Language Model Serving Throughput. *arXiv preprint arXiv:2408.12757*, 2024.

Table 5: **Pretraining compute and setup for each Kraken configuration**

Model	Layers	Embedding Dimension	Attention Heads	Total Params.	A100 GPU Hrs	Initial Learning Rate
Kraken 2-way	12	678	12	124M	480	$2.5e-4$
Kraken 4-way	12	504	12	124.5M	500	$2.5e-4$
Kraken 6-way	12	418	12	123.2M	800	$2.5e-4$
Kraken 2-way	24	888	16	350M	1000	$1.5e-4$
Kraken 4-way	24	644	16	353.4M	1750	$1.5e-4$
Kraken 4-way	24	960	16	761M	1800	$1.5e-4$

A Appendix

A.1 Training setup and compute requirements

For all pretrained models presented in Section 4.1, we used a similarly sized GPT-3 (7) model's hyperparameters as the basis and followed the procedure outlined in Section 3.2 to calculate the embedding dimension. We did not make an effort to optimize the codebase used for training which builds off of nanoGPT (29). It is possible to replicate pretrained models by extending nanoGPT to implement the new forward pass as described in Algorithm 1. The Adam optimizer was used to train all models along with a cosine learning rate decay with linear warmup. Initial learning rates and the approximate GPU hours required to train each configuration are presented in Table 5. All models were trained for $300,000$ gradient steps. Only the $761M$ parameter model was trained on a node with 80GB A100 GPU machines. The other configurations were trained on 40GB A100 machines and consequently use many more gradient accumulation steps. This is why the largest model required a similar number of GPU hours as the next largest.

Weights for all pretrained GPT-2 models used when evaluating SuperGLUE performance (18) (45) (30) (59) (16) (5) (22) (6) (40) (46) (41) (33) in Section 4.2 were obtained from HuggingFace. Since the focus of this work is on illustrating the efficiency gains and language modeling capabilities, we do not implement any safeguards that will filter for biased and/or harmful content. Initial experiments that tried various variations of the model architecture required about another 1,000 hours of A100 compute.

A.2 Overlap plugin implementation

We used TensorRT-LLM version *0.12.0.dev2024073000* throughout the evaluation. CUDA allows kernels to be launched on different streams and depending on resource availability, these kernels may be executed in parallel. However, TensorRT does not support multi-stream execution across plugins. We circumvented this limitation by implementing a Singleton that can manage a dedicated stream and global memory meant for launching collectives. This allows different instances of the plugin to launch and synchronize kernels on the same stream. Each instance of the plugin can either: trigger an AllReduce op on a separate low-priority CUDA stream or synchronize the stream to ensure that a previously launched AllReduce completes. The plugin also implements the functionality provided in the existing GEMM plugin. This allows us to perform the following within the Multi-Head Attention block:

1. The AllReduce is launched on a dedicated CUDA stream just before the GEMM used to compute the Query, Key, and Value matrices (GEMM QKV 1) via the plugin
2. All intermediate compute is performed using existing kernels such as FlashAttention
3. The CUDA stream that the AllReduce op was placed on is synchronized after the GEMM W_{out} 1, also via the Plugin

This approach allowed us to overlap the collective with all the computation required for Multi-Head Attention and is the only addition to TensorRT-LLM aside from the definition of Kraken. However, it also requires two extraneous memory copy operations that can be avoided if the library adds support for multi-stream execution.

Table 6: **Inference latency in milliseconds for 4-way parallelism**

Model Size	Context Length	Standard	Parallel Attn. + FeedForward	Kraken
1.3B	128	3.7	3.3	3.3
1.3B	2048	17.0	13.6	13.2
6.7B	128	8.3	7.0	5.8
6.7B	2048	48.2	42.1	38.0
13B	128	13.0	11.1	10.7
13B	2048	83.7	73.8	66.9
65B	128	12.7	11.2	8.5
65B	2048	84.6	79.5	63.8
175B	128	24.6	22.5	19.9
175B	2048	243.7	230.8	158.9

Table 7: **Inference latency in milliseconds for 8-way parallelism**

Model Size	Context Length	Standard	Parallel Attn. + FeedForward	Kraken
1.3B	128	4.3	3.4	3.2
1.3B	2048	15.9	11.7	9.7
6.7B	128	7.1	5.7	4.7
6.7B	2048	37.3	29.2	27.6
13B	128	10.7	8.4	6.7
13B	2048	58.8	46.9	42.6
65B	128	8.7	7.2	6.2
65B	2048	55.9	48.9	42.9
175B	128	16.9	14.4	12.4
175B	2048	125.1	114.7	98.0

A.3 Time To First Token in milliseconds

Table 6 contains the results from Figure 4 but in terms of milliseconds. Similarly, Table 7 presents the results from Figure 5. For the 65B and 175B configurations, we ran into CUDA Out-of-Memory errors when running full sized engines because each device has only 40GB of memory. To avoid this, we reduce the number of layers to $\frac{1}{4}$ the original number. Nonetheless, the relative latency comparison should be unaffected because the runtimes of individual layers are identical.

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: Yes, the abstract and introduction clearly describe the work and refer only to attained experimental results.

   Guidelines:

   - The answer NA means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A No or NA answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: The *Discussion and Limitations* section goes over the limitations of this work and contrasts it with several categories of related work.

   Guidelines:

   - The answer NA means that the paper has no limitation while the answer No means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory Assumptions and Proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

Answer: [NA]

Justification: This paper does not make any theoretical claims about the efficacy of the proposed architecture.

Guidelines:

- The answer NA means that the paper does not include theoretical results.
- All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
- All assumptions should be clearly stated or referenced in the statement of any theorems.
- The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
- Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
- Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental Result Reproducibility**

Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

Answer: [Yes]

Justification: All pretrained models can be replicated by using the training recipe presented in the Appendix along with the Algorithm in the Architecture section. The dataset used for training as well as the codebase are both publicly available. Similarly, the TensorRT-LLM comparison can be reproduced by extending the library to support the new model and then implementing the Overlap plugin as described in the Appendix. More importantly, we have also made pertinent code open-source.

Guidelines:

- The answer NA means that the paper does not include experiments.
- If the paper includes experiments, a No answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
- If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
- Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
- While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
  (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
  (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
  (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
  (d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility.

In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

   Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

   Answer: [Yes]

   Justification: Relevant code has been published online; details can be found in the Evaluation.

   Guidelines:

   - The answer NA means that paper does not include experiments requiring code.
   - Please see the NeurIPS code and data submission guidelines (`https://nips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - While we encourage the release of code and data, we understand that this might not be possible, so "No" is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
   - The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://nips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
   - The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
   - At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
   - Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental Setting/Details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer, etc.) necessary to understand the results?

   Answer: [Yes]

   Justification: All the information necessary to train the models on OpenWebText, perform the comparison using SuperGLUE, build TensorRT-LLM engines, and replicate results can be found in the paper and Appendix.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
   - The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment Statistical Significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer: [No]

   Justification: We did not have the compute resources to train several versions of the same model configuration and report perplexity and SuperGLUE results with error bars.

   Guidelines:

   - The answer NA means that the paper does not include experiments.

- The authors should answer "Yes" if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
- The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
- The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
- The assumptions made should be given (e.g., Normally distributed errors).
- It should be clear whether the error bar is the standard deviation or the standard error of the mean.
- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g. negative error rates).
- If error bars are reported in tables or plots, The authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments Compute Resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: The number of GPU hours necessary to train the models is discussed in the Appendix along with other details about the compute resources used.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code Of Ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes]

   Justification: The Code of Ethics was reviewed carefully for full compliance.

   Guidelines:

   - The answer NA means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer No, they should explain the special circumstances that require a deviation from the Code of Ethics.
   - The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader Impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [No]

Justification: While the paper does present pretrained language models, the accompanying societal impacts and necessary safeguards are no different from that of existing language models. As such, we expect that existing mitigation strategies will be applicable to the ideas presented in this paper.

Guidelines:

- The answer NA means that there is no societal impact of the work performed.
- If the authors answer NA or No, they should explain why their work has no societal impact or why the paper does not address societal impact.
- Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
- The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
- The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
- If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

    Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pretrained language models, image generators, or scraped datasets)?

    Answer: [Yes]

    Justification: While this paper presents a few pretrained models, the focus is on the system-level efficiency gains, not the downstream performance of the models. We have highlighted this in the evaluation and Appendix.

    Guidelines:

    - The answer NA means that the paper poses no such risks.
    - Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
    - Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
    - We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

    Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

    Answer: [Yes]

    Justification: All assets such as datasets and codebases are cited and urls are provided were applicable.

Guidelines:

- The answer NA means that the paper does not use existing assets.
- The authors should cite the original paper that produced the code package or dataset.
- The authors should state which version of the asset is used and, if possible, include a URL.
- The name of the license (e.g., CC-BY 4.0) should be included for each asset.
- For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
- If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
- For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New Assets**

    Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

    Answer: [Yes]

    Justification: Yes, our code release is documented with a focus on being easy to reproduce.

    Guidelines:

    - The answer NA means that the paper does not release new assets.
    - Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
    - The paper should discuss whether and how consent was obtained from people whose asset is used.
    - At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and Research with Human Subjects**

    Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

    Answer: [NA]

    Justification: The experiments in this work did not directly include data from crowdsourcing or research with human subjects.

    Guidelines:

    - The answer NA means that the paper does not involve crowdsourcing nor research with human subjects.
    - Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
    - According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional Review Board (IRB) Approvals or Equivalent for Research with Human Subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

Answer: [NA]

Justification: None of the experiments in this work involved human subjects.

Guidelines:

- The answer NA means that the paper does not involve crowdsourcing nor research with human subjects.
- Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
- We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
- For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.