Comparative Study of Large Language Model Architectures on Frontier

Junqi Yin
Oak Ridge National Laboratory
Oak Ridge, TN
yinj@ornl.gov

Avishek Bose
Oak Ridge National Laboratory
Oak Ridge, TN
bosea@ornl.gov

Guojing Cong
Oak Ridge National Laboratory
Oak Ridge, TN
congg@ornl.gov

Isaac Lyngaas
Oak Ridge National Laboratory
Oak Ridge, TN
lyngaasir@ornl.gov

Quentin Anthony
Ohio State University
EleutherAI
Columbus, OH
qubitquentin@gmail.com

***Abstract*—Large language models (LLMs) have garnered significant attention in both the AI community and beyond. Among these, the Generative Pre-trained Transformer (GPT) has emerged as the dominant architecture, spawning numerous variants. However, these variants have undergone pre-training under diverse conditions, including variations in input data, data preprocessing, and training methodologies, resulting in a lack of controlled comparative studies. Here we meticulously examine two prominent open-sourced GPT architectures, GPT-NeoX and LLaMA, leveraging the computational power of Frontier, the world's first Exascale supercomputer. Employing the same materials science text corpus and a comprehensive end-to-end pipeline, we conduct a comparative analysis of their training and downstream performance. Our efforts culminate in achieving state-of-the-art performance on a challenging materials science benchmark. Furthermore, we investigate the computation and energy efficiency, and propose a computationally efficient method for architecture design. To our knowledge, these pre-trained models represent the largest available for materials science. Our findings provide practical guidance for building LLMs on HPC platforms.**

***Index Terms*—AI foundation model, GPT architecture, HPC**

I. Introduction

Since the inception of the Transformer architecture [1], Transformer-based large language models (LLMs) have emerged as the bedrock upon which numerous AI breakthroughs have been constructed. Between two widely adopted architectures, namely bidirectional encoder representations from Transformers (BERT) [2] and generative pre-trained Transformer (GPT) [3], it has been shown that the performance of GPT models scales [4] with both model and data sizes, while marginal benefit was observed [5] comparing BERT models of different sizes. Many efforts have then been devoted to improving GPT architectures, including GPT-1 to GPT-4 [3], GPT-NeoX [6], LLaMA [7, 8], etc. Although the record performance was refreshed repeatedly, the focus was on downstream tasks only and little was discussed on the reason behind the architecture choices. A careful examination of current practices on the end-to-end pipeline for building LLMs is needed, especially for scientific applications.

With the rise of LLMs, a new research paradigm emerges in the field of AI for sciences, which is to build a foundation model via unsupervised learning on scientific data, and then fine-tune it to apply for many downstream scientific tasks. There have been several early attempts, e.g., the ClimaX [9] — a vision Transformer foundation model for climate, bioGPT [10], and pubmedGPT [11] which are generative pre-trained Transformers based on biology and medical text data. To the best of our knowledge, there is not yet a foundation model pre-trained specifically for materials science.

Materials science is fundamental to everyday life. From lightweight materials in transportation to innovative energy storage, materials science plays a crucial role in shaping modern society. The design of new materials relies on the understanding of existing research. To extract knowledge from the materials publications, several natural language models have been built, including word2vec models [12], BERT-style models [13], etc. However, these models are limited to specific tasks and cannot be generalized. We intend to pre-train a foundation model based on the GPT architecture and demonstrate its capability on both generic language benchmarks and scientific downstream tasks.

While the current open-sourced state-of-the-art GPT variant is LLaMA [7] (more recent LLaMA2 [8] includes tweaks to improve inference performance) and the top performer Falcon model [14] employs the same LLaMA architecture but with improved data quality, little work has been done comparing different architectures and pre-training recipes. In this work, we will investigate two prominent open-sourced GPT variants, namely GPT-NeoX [6] and LLaMA, and show performance comparisons on both model training and downstream tasks. In addition to evaluating zero- and few-shot performance on question answering tasks, we also propose a scientific regression task to showcase the scientific usage of the model.

Another important aspect in building LLMs is the computational performance and energy efficiency because the process typically incurs a large computational cost and significant energy consumption. Most of the established practices, however,

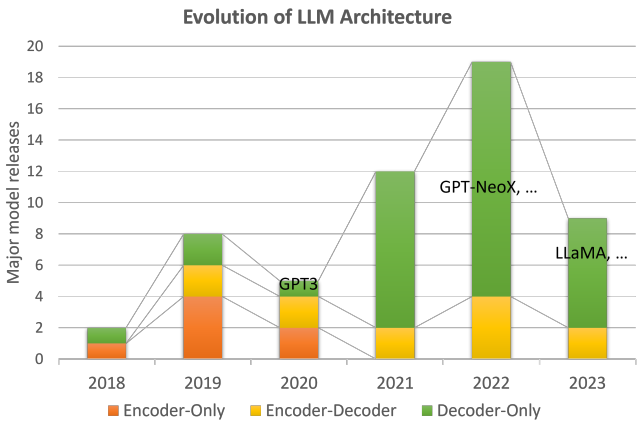


Fig. 1: Evolution of LLM architecture since 2018. Starting from 2021, the GPT architecture dominates the major model releases.

are on NVIDIA GPU-based platforms. Here we will study the performance and optimization of the popular training frameworks on Frontier — the first Exascale supercomputer which is equipped with AMD GPUs. Specifically, our contributions include:

- We comparatively studied two mainstream GPT architecture variants, i.e., GPT-NeoX and LLaMA, and evaluated their end-to-end pre-training recipes.
- We pre-trained and openly released a set of foundation models for materials science, called MatGPT.
- We proposed a new downstream task for scientific usage and achieved state-of-the-art performance on a challenging materials science benchmark.
- We established baselines and practical guidance for building LLMs on AMD GPU-based platforms.

Our focus lies in constructing LLMs for scientific purposes on HPC. While we exemplify this within the field of materials science, our methods are not constrained to a singular domain. The rest of the paper is organized as follows: Sec. II reviews the architecture trends of LLMs and current state-of-the-art applications for materials science. Sec. III details our data corpus and end-to-end computational approach. Sec. IV presents our comprehensive comparisons from architecture selection and training performance, to downstream tasks and model explainability, and we conclude in Sec. V.

II. Related work

Following the evolutionary tree of LLM architecture [15], three main branches, i.e., encoder-only, encoder-decoder, and decoder-only, stem from the introduction of Transformer architecture [1] in 2017. The number of major model releases within each branch are plotted in Fig. 1 for each year ever since. From 2018 to 2019, encoder-only models such as BERT [2] enjoyed more popularity. Since GPT-3 [16], which demonstrated emerging capability with the unprecedented parameter size (175B), the decoder-only architecture has dominated the field. GPT-NeoX [6] and LLaMA [7] are among the most popular open-source variants of GPT-3. On the other hand, the number of encoder-decoder models, e.g, T5 [17] for translation tasks, has stayed about the same.

In recent years, the materials science community has embraced the advancements in natural language processing (NLP). A study [18] in Nature built a word2vec model on 3M abstracts and demonstrated its usage in material recommendation for functional applications. Since then, BERT-style models [13, 19] pre-trained specifically on material texts have been the best performing models; and MatSciBERT [13] is considered as the current state-of-the-art for domain-specific LLM for materials science. However, the model and data sizes are limited to hundreds of millions of parameters and several millions of papers, respectively. No generalization ability has been demonstrated.

An early attempt [20] was made to apply GPT-3.5 model to study energy materials, but it is fine-tuned only on a small dataset and a generic foundation model for materials science is still lacking.

III. Method

TABLE I: Data Sources for MatGPT.

Source	#abstract	#full-text	#tokens
CORE	2.5M	0.3M	8.8B
MAG	15M		3.5B
Aminer	3M		1.2B
SCOPUS	6M		1.5B
All	26.5M	0.3M	15B

Data Sources We collect the abstracts and full-text data from the four data sources, including CORE, Microsoft Academic Graph (MAG), Aminer, and SCOPUS, as listed in Table I. For SCOPUS, we use the publisher's API to retrieve the abstracts of about 6M materials science publications [12]. For the other sources, aggregated data covering all scientific domains are downloaded and then preprocessed to filter out materials science-related ones. The screening is performed via a fine-tuned SciBERT model on a small domain-labeled dataset, and the resulting classifier can then be used to partition the aggregated data sources. In total, there are 26.5M abstracts and 0.3M full-texts, counting to about 15B tokens.

Model Architecture We build MatGPT upon two mainstream model architectures, GPT-NeoX [6] and LLaMA [7], both of which are based on GPT-3 but with different variations. As illustrated in Fig. 2, the original LLaMA employs the SentencePiece (SPM) tokenizer with a vocabulary size (vocab-size) of 32K; While GPT-NeoX utilizes the HuggingFace (HF) tokenizer with a vocab-size of 52K. SPM has fine-grained control over subword tokenization while HF is more popular for pre-trained Transformers. Both architectures use rotary positional embeddings [21] instead of absolute positional embeddings [22] as in GPT-3, and LLaMA made further modifications to the pre-normalization and activation by using RMSNorm and SwiGLU [7] activation functions. The specific choice of number of layers and hidden size, as

TABLE II: Model architectures and various data tokenization (HuggingFace and Sentencepiece tokenizer — HF and SPM, and vocabulary size of 32K and 52K).

MatGPT Arch	#parameters	hidden-size	#layers	#heads	head-dim	tokenizer	vocab-size
LLaMA	1.7B	2304	24	24	96	SPM/HF	32K/52K
	6.7B	4096	32	32	128	HF	52K
GPT-NeoX	1.7B	2304	24	24	96	HF	52K
	6.7B	4096	32	32	128	HF	52K

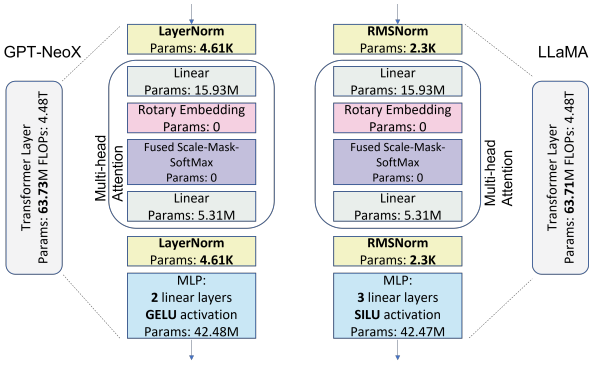


Fig. 2: Transformer layer of GPT-NeoX and LLaMA architecture, respectively. The specific parameter and FLOP numbers are for 1.7B parameter model with a sequence length of 2048 and batch size of 16.

listed in Table II, also takes the computational performance into consideration, which will be discussed in Sec. IV-B. For the models of the same specification (i.e, number of layers and attention heads, and hidden size), each Transformer layer of GPT-NeoX and LLaMA has approximately the same number of parameters and floating-point operations (FLOPs). The multi-head attention layers are exactly identical, and the main difference is the normalization layers and multi-layer perceptrons (MLPs).

The Chinchilla study [23] suggested that the optimal token-to-parameter ratio is 20, while more recent studies [7, 8] indicates a higher ratio, i.e., a smaller model and larger data, can be beneficial. Given our data size (15B tokens), we build MatGPT with a few billion parameters. To identify the most computationally efficient architecture, we perform a grid search for various numbers of layers and hidden sizes. The specific choice has to satisfy following constraints,

$$N_h \ \% \ N_a = 0 \tag{1}$$
$$N_h \ \% \ (TP) = 0 \tag{2}$$
$$N_l \ \% \ (PP) = 0 \tag{3}$$
$$N_a \ \% \ (TP) = 0 \tag{4}$$
$$(TP \times PP \times DP) \ \% \ 8 = 0 \tag{5}$$

where N_h, N_l, and N_a denote hidden size, number of layers and attention heads, and TP, PP, and DP are the tensor, pipeline, and data parallelisms, the product of which should be equal to the total number of devices (multiple of 8 for our platform). The dimension of the attention head is implemented as the ratio of N_h over N_a, hence the constraint; and the rest is to ensure the workload can be evenly distributed among parallelism dimensions.

We will compare the model performance of the two architectures, as well as different choices of tokenizers and vocab-sizes. As far as we know, this is the first controlled study of LLM architectures at a large scale.

Flash Attention One of the most important recent advances in the optimization of training Transformers is flash attention [24], which significantly reduce high-bandwidth memory (HBM) accesses which in turn speeds up the computation of the attention calculations. Because it leverage classical HPC techniques at a lower level (e.g., tiling, recomputation), the implementation is hardware-specific. In our evaluation, we make use of the open-sourced development [25] of flash attention on AMD GPUs. It is based on the composable kernel library [26], which performs many of the same functionalities of the CUTLASS library for NVIDIA GPUs, and currently both forward and backward calculations are supported. The latest flash attention v2[27] is also in the process of being incorporated into AMD GPUs. We will study the computational and memory impact of both versions. To our best knowledge, this is the first such study on recent AMD hardware (MI250X).

Training and Evaluation Framework Our evaluation is based on the GPT-NeoX [6] framework, a DeepSpeed-Megatron [28, 29] implementation. We port it to the AMD GPU-based platform, Frontier [30] with following modifications: 1) Add support for the LAMB optimizer [31], which is an enhanced Adam optimizer [32] that can mitigate the generalization gap caused by the large-batch training. 2) Hipify the fused kernel in CUDA to support the AMD ROCm stack. 3) Adapt the flash attention interface from the corresponding AMD implementation.

Because the LLM training is shown to be communication-bound on Frontier [33] and model parallelism is more demanding than data parallelism in terms of the communication requirement [34], it is desired to assign most compute resources (i.e., GPUs) to data parallelism (i.e., large batch size), in order to achieve good scaling efficiency and energy utilization. Therefore, exploring the LAMB optimizer is necessary for efficient distributed training of LLMs on HPC systems.

Regarding the downstream evaluation, we employ the generic language model evaluation framework [35] for the common question answering tasks. This will demonstrate MatGPT's generalizability as a foundation model, even though it is not pre-trained on generic web texts. More importantly,

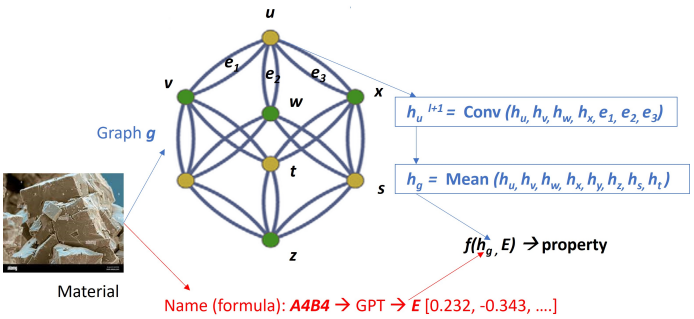


Fig. 3: A new scientific usage of LLM: combining LLM embeddings with GNN for material properties prediction.

we propose a new scientific downstream task to demonstrate the performance and science usage of MatGPT, namely the material properties prediction.

Scientific Downstream Task A direct and versatile approach for integrating LLMs into scientific applications involves enhancing the feature space of existing models by incorporating LLM embeddings [36]. These embeddings effectively capture correlations among specific terms, such as named entities, within the LLM context. Consequently, the numerical vector representation of text terms within the latent space of LLMs can serve as features to predict characteristics associated with corresponding terms. For LLMs pre-trained on scientific publications, such as MatGPT, the embeddings are expected to outperform those of LLMs pre-trained on web texts, and the larger the capacity (i.e., model parameters) of a LLM, the better quality of its embeddings. To demonstrate the capability of MatGPT, we will show the comparisons of its embeddings with other LLMs including MatSciBERT.

As an example for materials science, we explore leveraging the trained LLMs for improving learning in material properties prediction. Graph neural networks (GNNs) are the de facto paradigm for predicting the physical and functional properties of materials. Recent studies incorporate features of increasing complexity such as Gaussian radial functions, plane wave functions, and angular terms to augment the neural network models, with the expectation that these features are critical for achieving high performance [37, 38, 39, 40, 41, 42]. However, none of these efforts tap into the vast amount of publications that reflect human understandings of material research. With LLM trained on material texts, we experiment with using both material structure data, fed to GNNs, and scientific publications, reflected in the embeddings of corresponding material formulas, for learning.

For demonstration, we choose to predict band gap, a quantum chemical property of materials. The ground truth for the band gap is computed from density function theory (DFT) calculations. DFT is a quantum mechanical method for solving the electronic structure of a material widely used in computational materials chemistry. The band gap of a material is the energy difference between its highest occupied electronic energy level and the lowest unoccupied energy level. It is more challenging to predict band gap than other properties such as formation energy.

We adopt a simple methodology that concatenates the output from a GNN for a material with the LLM embedding of the corresponding material's chemistry formula. The learning paradigm is shown in Figure 3.

In Figure 3, a graph g is constructed from the structure information (e.g., atoms, their 3-dimensional positions, and their interactions) of the input material, and graph convolution is then applied to derive a feature h_g. At the same time, an embedding E is generated using the trained LLM for the formula of the material. E and h_g are then concatenated as a feature for the prediction of the property.

Our study demonstrates the boost of learning performance of GNNs from incorporating additional semantic and contextual information. Moreover, the results and insight from the experiments by employing MatSciBERT and MatGPT variants robustly support our claim of having more expressive model architectures, MatGPT trained on larger datasets with a large number of parameters compared to MatSciBERT. The dataset we use is the material project dataset [43].

IV. Evaluation

A. Experiment setup

We perform the experiments on the first Exascale supercomputer, Frontier. Each Frontier node is equipped with four AMD Instinct MI250X GPUs with dual Graphics Compute Dies (GCDs) and one third-generation EPYC CPU. A GCD is viewed as an effective GPU, and we use GCD and GPU interchangeably in the following discussion. All four MI250Xs (eight effective GPUs) are connected using 100 GB/s Infinity Fabric (200 GB/s between 2 GCDs of MI250X), and the nodes are connected via a Slingshot-11 interconnect with 100 GB/s of bandwidth. Frontier consists of 9408 nodes in total, i.e., 75,264 effective GPUs (each equipped with 64GB high-bandwidth memory).

Our evaluation is based on GPT-NeoX [6] implementation with PyTorch v1.14.0 [44] and DeepSpeed v0.7.3 [28], which are built against AMD ROCm v5.4. In the following, we will investigate the MatGPT performance, including training throughput, architecture comparisons, and model performance on our newly proposed scientific downstream task and commonly used language benchmarks. The tokenized data and pre-training code are publicly available[1].

TABLE III: Training hyper-parameters for MatGPT.

Model	Optimizer	β_1	β_2	LR	BS
1.7B	Adam	0.9	0.95	0.0002	1M
	LAMB	0.9	0.999	0.01	4M
6.7B	LAMB	0.9	0.999	0.006	4M

Considering the large GPU capacity (both memory and count) and network bandwidth (relatively limited compared to AI-oriented machines such as Selene[29]) on Frontier, it is desirable to leverage large-batch training in order to achieve good scaling efficiency and reduce time-to-solution. We train

[1]https://github.com/at-aaims/forge/tree/matgpt

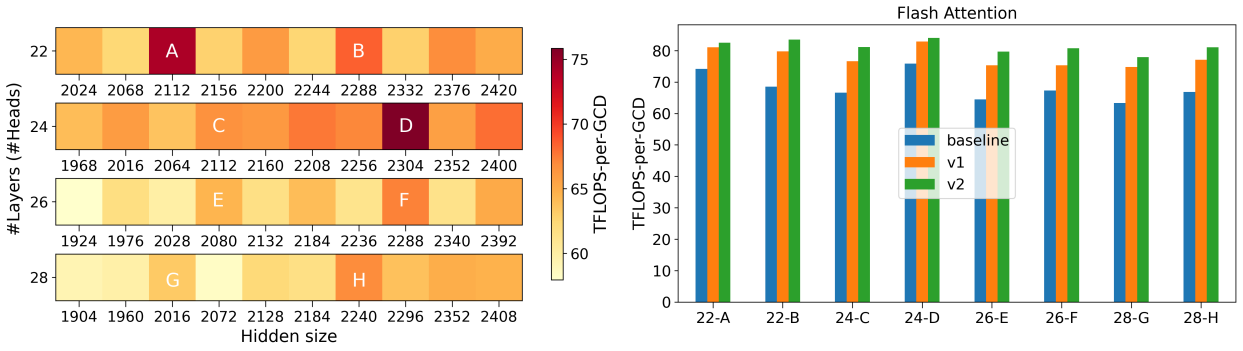


Fig. 4: (Left) The heatmap of training throughput (TFLOPS per GPU) for MatGPT with various numbers of layers and hidden sizes for model size around 1B. (Right) The performance boost for architectures eligible for flash attention, including v1 and v2, respectively.

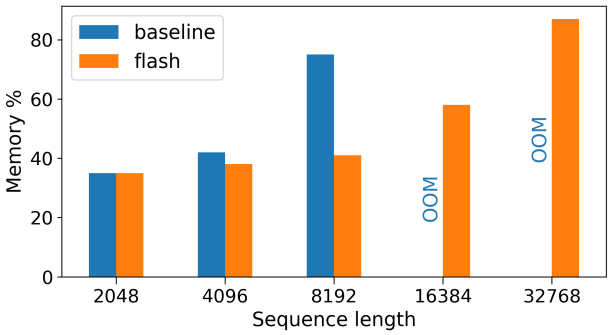


Fig. 5: The peak memory usage (percentage) during the training of MatGPT 1.7B with and without flash attention for context sequence length from 2,048 to 32,768.

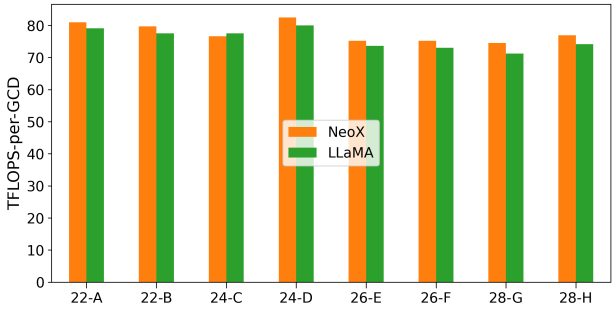


Fig. 6: The comparisons of training throughput (TFLOPS per GCD) for MatGPT -NeoX and -LLaMA achitectures.

MatGPT with the LAMB optimizer [31], a variant of the Adam optimizer with the layer-wise learning rate adjustment to mitigate the generalization gap of large-batch training. We use $\beta_1 = 0.9, \beta_2 = 0.999$, a weight decay of 0.1, and a batch size of 4M tokens. The cosine learning rate scheduler is employed with an initial learning rate 0.01 and a final learning rate set to 10% of the initial learning rate. We use 1% of the total batch steps for warmup. The training is performed in bfloat16, which provides better numerical stability.

B. Results

Following the above experiment setup, we conduct a thorough investigation in the end-to-end pipeline of building MatGPT, including the evaluation of the computational performance for training, comparisons of both the loss and zero-shot tasks performance for different architecture choices, and fine-tuning performance for the scientific downstream task we proposed.

Training Performance The scaling law [4] indicates the model performance scales with the number of parameters, however, for similar-size models, the choice of the number of layers, hidden size, and attention heads, seems arbitrary. We refine our model design by taking the computational performance into consideration. Because the workhorse of the training is matrix multiplication and the underlying math library, such as cuDNN for NVIDIA GPUs and MIOpen for AMD GPUs, is optimized for certain shapes of matrices [45].

In Fig. 4, we plot the heatmap of MatGPT-NeoX training throughput in terms of TFLOPS per GPU for various numbers of layers and hidden sizes for model sizes around 1B parameters. The performance varies from 58 to 76 TFLOPS, and the best case (achieved about 40% of the MI250X theoretical peak — 383 TFLOPS for 2 GCDs) corresponds to 24 layers with a hidden size of 2304. This is mostly due to the fact that the computation of multi-head attentions (see Fig. 2) is expensive and a head dimension of multiples of 8 is computationally favorable (it can take full advantage of matrix cores on AMD GPUs, which are equivalent to tensor cores on NVIDIA GPUs). We marked all the architectures (from A to H in Fig. 4) with head dimensions satisfying this criteria, and indeed they are among top performers for each layer size. Coincidentally, flash attention also requires the head dimension to be multiples of 8 (and up to 256 for v2), and the performance boost for each case are also shown in Fig. 4 for v1 and v2, respectively. On average, the current flash attention implementation improves the computational performance by about 14% (v1) and 19%(v2), with the best overall training throughput of about 82 TFLOPS per GCD (164 TFLOPS per MI250X) for v1 and 84/168 TFLOPS for v2, respectively,

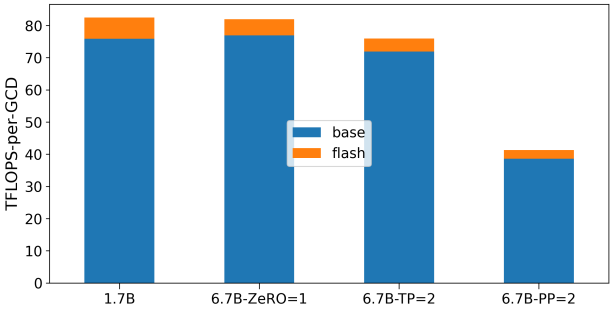


Fig. 7: The training throughput (TFLOPS per GCD) for MatGPT 1.7B and 6.7B with different parallelisms: ZeRO stage 1 (ZeRO=1), tenor parallelism of 2 partitions (TP=2), and pipeline parallelism of 2 stages (PP=2).

on Frontier. Note that these numbers are averaged per-node performance for a sequence length of 2048 and the port [25] of flash attention to the ROCm stack is still in active development, and further improvement is expected.

Furthermore, the flash attention reduces the memory complexity from quadratic to linear in terms of the sequence length, and hence enables longer context window. In Fig. 5, we plot the peak memory usage during the training of MatGPT 1.7B for context sequence lengths ranging from 2048 to 32768. Without flash attention, the training process runs out of memory (OOM) for sequences longer than 8192; With flash attention enabled, the memory growth becomes linear (after the sequence length dominates the memory usage) and the maximum supported sequence length increases by about 4X to 32768 on Frontier.

To compare the computational performance for NeoX and LLaMA, we plot the training throughput for all 8 cases with flash attention, as shown in Fig. 6. Because of the architecture similarity, especially the identical attention layer (see Fig. 2), both perform more or less the same, with NeoX showing a slight edge in 7 out of 8 cases. The difference likely comes from the parameterization of MLP layers (2 linear layers with GELU activation versus 3 linear layers with SILU activation).

Observation (1) It is computationally desirable to design the LLM architecture with the dimension of attention head to be multiples of 8. With flash attention, the achievable computational performance for training Transformers is over 43% of the theoretical peak on MI250X for a sequence length of 2048. Given a targeted model size, computational efficiency can be a criterion for architecture selection.

After identifying the most computationally efficient architecture (i.e., 24 layers and a hidden size of 2304) on MI250X for models of around 1B parameters, we extrapolate the observation to identify a 6.7B model (see Table II) with a head dimension of 128. As a rule of thumb, the memory footprint for training a GPT-style model is roughly 12 times of the parameters [33]. For the training of a 1.7B model, a single GCD on a MI250X (equipped with 64GB high-bandwidth memory) is able to accommodate the entire model. However,

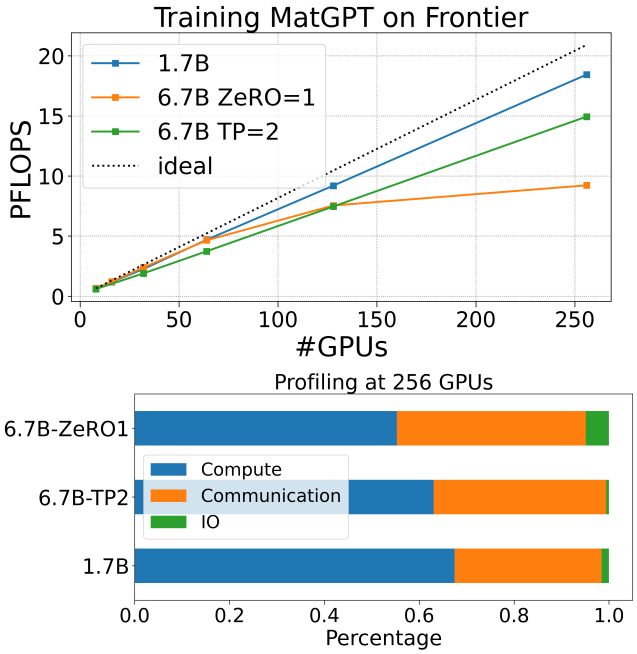


Fig. 8: (Top) The Scaling of training throughput (PFLOPS) for MatGPT 1.7B and 6.7B with different parallelisms: ZeRO stage 1 (ZeRO=1), tenor parallelism of 2 partitions (TP=2). (Bottom) The profiling breakdown of compute, communication, and IO for the three corresponding parallel distributions with 256 GPUs.

for a 6.7B model, some level of model parallelism is required. The choices can be DeepSpeed ZeRO optimization (e.g., stage 1 for partitioning the optimizer states), tensor parallelism (e.g., TP = 2 for partitioning each layer onto 2 devices), or pipeline parallelism (e.g., PP = 2 for executing layers through 2 stages, each stage on a separate device). Depending on the parallelism, the communication frequency and message size are different, i.e., imposing different requirements on the platform. In addition to the communication cost of the data parallelisms, the ZeRO parallelism fully shards the optimizer states (twice the memory footprint of model parameters for the Adam and LAMB optimizers) and hence requires all-devices collective communication during the backward propagation. On the other hand, the tensor and pipeline parallelisms have fine-grained control and can limit the extra communication for model parallelism within a subgroup of devices. Although tensor parallelism incurs more frequent messaging than pipeline parallelism (per-layer versus group of layers), there are sequential stages (leading to the so-called "bubble") in pipelining, and tensor parallelism can perform better with adequate network bandwidth.

In Fig. 7, we show the performance of training the 6.7B model on a single Frontier node, and compared with that of the 1.7B model. Compared to tensor and pipeline parallelism, ZeRO stage 1 provides the best training throughput (81 TFLOPS per GPU), with a similar boost from flash attention as for the 1.7B model due to the lesser communication frequency.

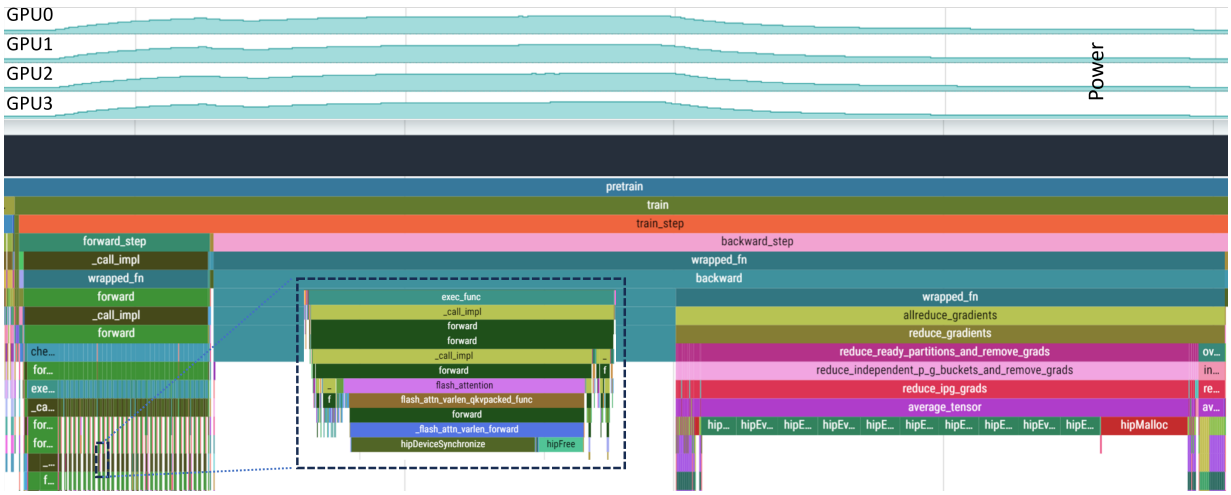


Fig. 9: The runtime and GPU power traces (single node) of distributed training of MatGPT 6.7B with ZerO stage 1 using 256 GPUs. The boxed snapshot is the zoom-in of the forward operations for one of 32 layers.

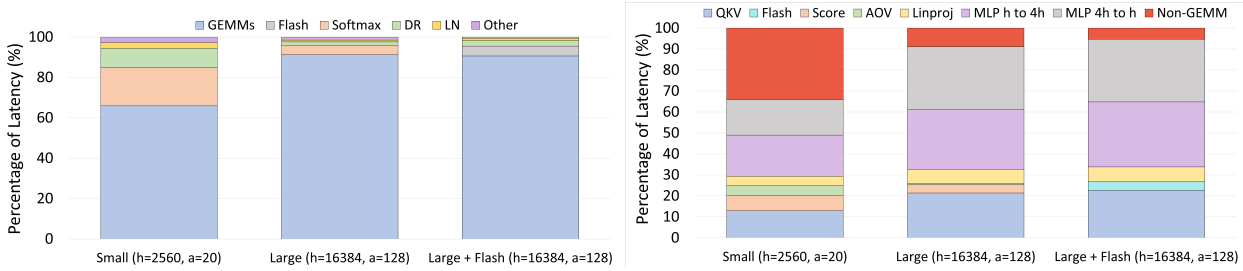


Fig. 10: (Left) The proportion of latency from each transformer component for one transformer layer of hidden dimension h and number of attention heads a. DR and LN stands for dropout and layer normalization operation, respectively. (Right) The individual GEMM proportions of latency for one transformer layer, including query-key-value (QKV), flash attention (flash), attention score (score), attention over value (AOV), linear projection (Linproj), multi-layer perceptron (MLP).

With the single-node performance optimized, we scale up the distributed training to 256 GPUs and explore 3D parallelism on Frontier. As shown in Fig. 8, for training MatGPT 1.7B model with data parallelism only, the aggregated performance of 256 GPUs on Frontier can achieve over 18 PFLOPS with a scaling efficiency of 88%. In comparison, for MatGPT 6.7B model, the per-device throughput is about the same for 64 or less GPUs with ZeRO stage 1 parallelism, and starts to drop at larger scale because of the extra communication overhead of all-device collectives. Tensor parallelism with a partition level of 2, on the other hand, can sustain a 71% scaling efficiency, owing to the fact that the 2 GPUs/GCDs are within the same MI250X with twice the network bandwidth (200 GB/s). Pipeline parallelism with 2 stages performs much worse compared to the other two parallelism dimensions even for a single node (see Fig. 7), and hence is not studied at scale. To better understand the scaling behavior, in Fig. 8, we show the profiling for the 3 corresponding parallel distributions (i.e., data parallel for 1.7B, and ZeRO stage 1 and tensor parallel TP=2 for 6.7B) with 256 GPUs. The run time statistics are collected using `rocprof` during the training, and aggregated into 3 type of kernels, i.e., computation, communication (RCCL calls), and IO (data movements including device to host, host to device, and device to device). As expected, IO doesn't play a big role in LLM training on Frontier. ZeRO has the most data movements but the IO kernels take about 5% of total run time. Communication becomes a bottleneck for training at scale, especially for larger models. For 6.7B with ZeRO stage 1, it accounts for about 40%. Note that in the above experiments, the per-device batch size is fixed. In the case of ZeRO stage 1, since the optimizer states are partitioned across all devices, the more GPUs, the less per-GPU memory footprint. Therefore, in practice, the per-device batch size can be increased to improve the scaling performance.

To better understand the profiling results, we plot the runtime and GPU power traces in Fig. 9. These traces are collected using `OmniTrace`, and to avoid the excessive overhead, only one node runs the tracing. Since the workloads are evenly distributed, the resulted traces are representative for understanding the distributed training. One training step is shown in Fig. 9, where the run time includes one forward and one backward step. The forward step consists of 32 layers (see MatGPT 6.7B architecture in Table II) operations, and each (zoom-in snapshot) is dominated by the flash attention

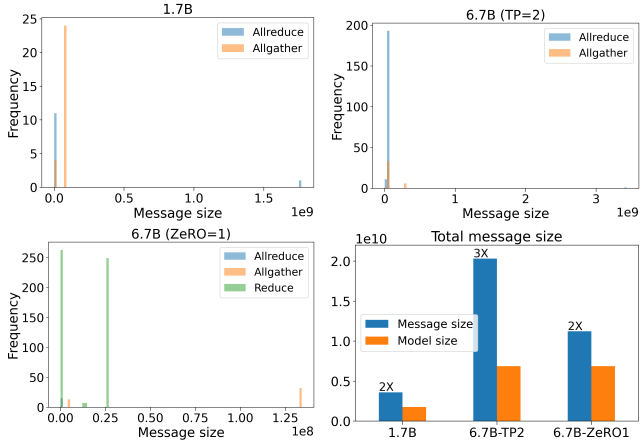


Fig. 11: The histogram and aggregated message size per-batch-step per-GPU during the training of 1.7B with data parallelism, 6.7B with ZeRO stage 1, and tensor parallelism (TP=2), respectively.

operation (v2). In the backward step, the allreduce operation takes a significant amount of time, consistent with `rocprof` profiling (see Fig. 8). The power traces (recorded per MI250X) shows high usage during the computation and drops down during the communication. All GPU traces behave similarly, confirming that the workloads are evenly distributed.

In order to understand the performance of GPU computation, we investigated the breakdown of kernels that are executed within each transformer layer. The results are depicted in Fig. 10, and demonstrate a few key takeaways. First, Fig. 10 (Left) shows that GEMMs account for the vast majority of a transformer layer's runtime, and their proportion increases with model scale (65.9% and 91.2% for medium- and large-sized models, respectively). Second, Fig. 10 (Right) shows that within these GEMM kernels, the query-key-value (QKV) transformation in the attention layer along with the MLP layer account for the most runtime. Therefore, future optimizations targeting these blocks would benefit training time the most.

To further investigate the impact of communication, as shown in Fig. 11, the histogram and aggregated message size per batch step per GPU are collected from RCCL logs (by setting `NCCL_DEBUG_SUBSYS=COLL`) for the three parallelism settings for the distributed training in Fig. 8. The ZeRO stage 1 and tensor parallelism TP=2 for 6.7B incurs over an order of magnitude more RCCL calls (e.g., Allreduce and Allgather) compared to vanilla data parallelism for 1.7B. In terms of the total message size, both data parallelism and ZeRO parallelism (considered as a memory-efficient data parallelism) require a communication size about 2X the model size, while tensor parallelism requires 3X due to the additional communication of model parameters. Although TP=2 incurs a larger communication volume than ZeRO stage 1, the scaling efficiency is actually better because the 2 GCD within an MI250X has a 2X communication bandwidth compared to the inter-node communication needed for ZeRO stage 1 (see Fig. 8).

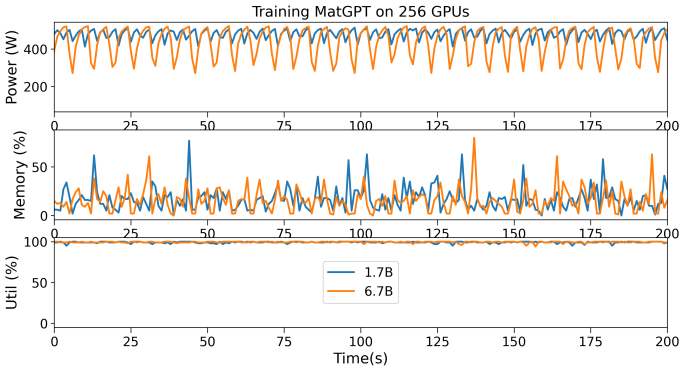


Fig. 12: The trace of power, memory, and GPU utilization for training MatGPT of 1.7B and 6.7B, respectively, with 256 GPUs on Frontier.

Observation (2) For HPC platforms optimized for workloads with less demanding communication requirements, adding extra parallelism dimensions such as tensor and pipeline usually adversely impacts the LLM training throughput. The recommended strategy is to keep model parallelism at the minimum and assign the rest of the computation resources to data parallelism. It is beneficial to map the partition of model parallelism to the platform network topology to maximize the the network bandwidth utilization.

With the optimized computational performance, we pre-train MatGPT- NeoX and LLaMA on the full set of data tokens and compare the loss and the performance of downstream tasks in the following sections.

Power Usage and Cost Analysis Pre-training LLMs is computationally expensive, and it's essential to be mindful of energy efficiency. We measure the power, memory, and GPU utilization during the training of MatGPT on Frontier with the `rocm-smi` tool. The update interval is per millisecond by default. Considering the duration of a training step is typically in seconds, `rocm-smi` can well capture the system metrics for most kernels during training. Due to the evenly distributed nature of the AI workload across all devices, the representative trace of one GPU is plotted. As shown in Fig. 12, the power trace for training 6.7B shows larger oscillation than that of 1.7B, with a mean value of 434 and 476 W for 6.7B and 1.7B, respectively. Note that there is only one power sensor on an MI250X and the reported number is for the sum of 2 GCDs. Because the communication kernels also occupy GPU, the near 100% GPU utilization for both cases is not a good indicator for the computation usage. Power actually correlates more closely with computational performance. In fact, the oscillation in power curves indicates the periodical computation and communication cycles, as shown in Fig. 9, the power trace for a single training step. Given the 75.9 and 80.5 TFLOPS per GPU for training 6.7B and 1.7B model with 256 GPUs, the energy efficiency for training can then be calculated as 0.27 and 0.33 TFLOPS/Watt, respectively. The training time and total energy consumption are also listed in Table IV. Note that the numbers are for training a single

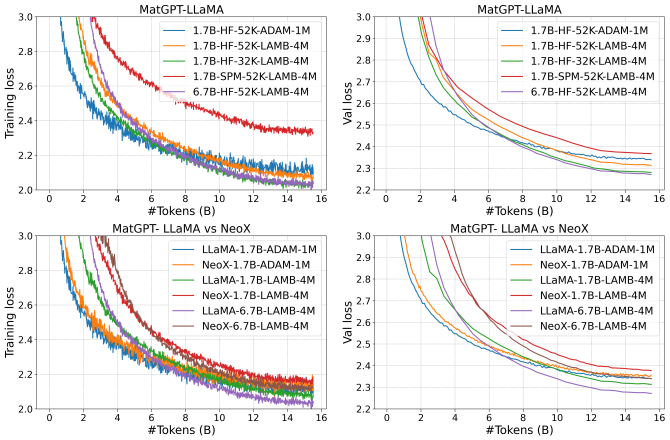


Fig. 13: The training and validation losses of MatGPT models. Model size: 1.7B and 6.7B; tokenizer: HuggingFace (HF) and sentencepiece (SPM); optimizer: Adam and LAMB; batch size: 1M and 4M.

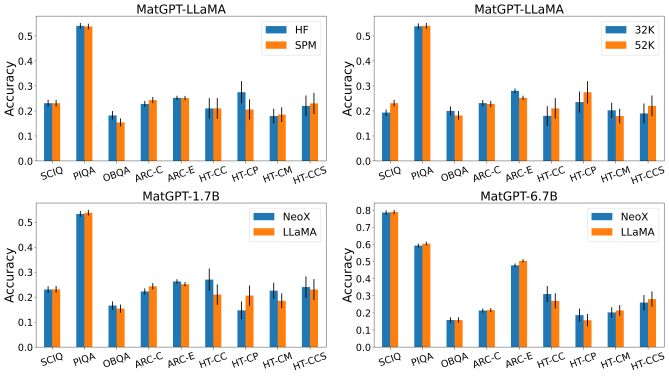


Fig. 14: Zero-shot performance. (Top) MatGPT-LLaMA 1.7B models trained with HF and SPM tokenizers on a vocabulary of size 52K, and HF tokenizer with 32K and 52K, respectively. (Bottom) MatGPT- LLaMA versus Neox models of sizes 1.7B and 6.7B.

model, and in this study we have trained 6 models in total.

TABLE IV: The time and energy usage for pre-training one 1.7B and 6.7B MatGPT model, respectively, on Frontier.

Model	GPUs	Time (hours)	Energy (MWh)	Efficiency (TFLOPS/Watt)
1.7B	256	4.1	0.23	0.33
6.7B	256	16.5	0.91	0.27

Loss Comparison It is shown [4] that the loss of LLM scales with the model and data sizes. The lower the loss, the better the model. Under the same experiment conditions, i.e., the same data processing and training pipeline, the loss can be used as an indicator to compare model performance. To identify the best model architecture on our 15B text corpus for materials science, we pre-train a suite of models on the Frontier supercomputer following a controlled recipe.

In Fig. 13, we plot the training and validation loss curves for all MatGPT-LLaMA architectures listed in Table II. As discussed in previous sections, to maximize the training throughput, most of the devices are assigned to data parallelism. This means that the training is performed with relatively large batch sizes. E.g., the GPT-3 model of a similar size[16] was trained with a batch size of 1M tokens using the Adam optimizer. Here we employ the LAMB optimizer to mitigate the generalization gap issue associated with large-batch training. By comparing the training and validation losses of the 1.7B model pre-trained on the same data (i.e., tokenized with the same HF tokenizer and 52K vocabulary size), the loss for the LAMB optimizer with 4M batch size is actually about 2% smaller, indicating it's a better training procedure. Since the original LLaMA model used the SPM tokenizer with a vocabulary size of 32K, we train two additional 1.7B models using corresponding conditions while fixing the rest of the experimental configuration. In comparison, the loss is significantly bigger for SPM and much smaller for 32K. However, because the training data tokens are effectively different in these cases, the absolute value of loss cannot be compared. On the other hand, for the 6.7B model trained with the same data (i.e., HFT tokenizer, 52K vocabulary size), the loss is indeed smaller than that of the 1.7B model. We also explored the effect of the training precision, and found that the loss curves for MatGPT 1.7B, trained with float16 and bfloat16, are almost identical.

To compare the MatGPT- NeoX and LLaMA, in Fig. 13, we plot the training and validation losses for corresponding 1.7B and 6.7B models. With the Adam optimizer and a batch size of 1M, the losses of both architectures are more or less the same. However, for the LAMB optimizer and a batch size of 4M, the MatGPT-LLaMA performs better for both model sizes.

Observation (3) The losses for LLMs pre-trained with different tokenizers and/or vocabularies are not comparable. With the same pre-training recipe, the LLaMA architecture seems to provide a smaller loss than that of NeoX.

Zero-shot Performance Although the LLM loss provides some indication of a model's performance, given LLM training is unsupervised, the downstream tasks are the ultimate metrics. Here we employ the popular question answering benchmarks [35] including SCiQ [46], PIQA [47], OpenBookQA (OBQA) [48], ARC-Easy (ARC-E) and challenge (ARC-C) [49], and Hendrycks colleague tests [50] on chemistry (HT-CC), physics (HT-CP), medicine (HT-CM), and compute science (HT-CCS).

We start with the zero-shot test, where LLMs are directly applied to tackle new, unseen tasks. It can demonstrate a model's generalizability. In Fig. 14, we show the effect of the tokenizer and vocabulary on the downstream language benchmarks. The zero-shot accuracies are plotted for the HF and SPM tokenizers with the same 52K vocabulary size, and the for 32K and 52K vocabulary sizes with the same HF tokenizer, respectively. The standard deviation in evaluating each benchmark is also plotted to show the variance in performance. The HF tokenizer seems to perform marginally better in 2 out of 9 tasks while the rest are about the same.

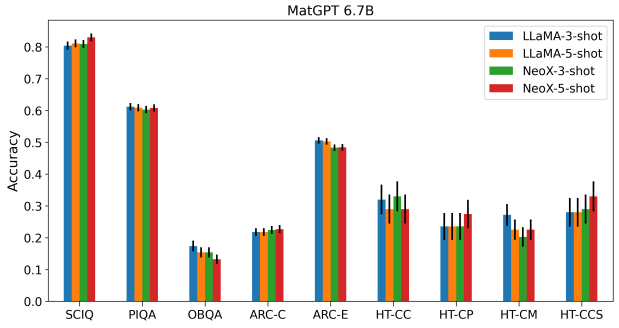


Fig. 15: Few shot (3 and 5) performance of MatGPT- NeoX and LLaMA 6.7B.

For vocabulary size, 52K performs slightly better in 4 out of 9 tasks, while 32K shows an edge in 2 tasks. Considering our text corpus comes from scientific articles, larger vocabulary seems to be able to distinguish domain terminologies such as chemical elements in materials formulae.

Using the same HF tokenizer and a vocabulary size of 52K, in Fig. 14, we compare the zero-shot performance of MatGPT-NeoX and LLaMA models. For 1.7B models, NeoX performs marginally better in 4 out of 9 tasks while LLaMA is slightly better in 2 out of 9 tasks. For 6.7B models, LLaMA shows an edge for 2 tasks while the rest are on par. The results indicate that the loss (see Fig. 13) does not fully correlate with the model's performance on downstream tasks.

Few-shot Performance In addition to zero-shot tests, it is also important to evaluate LLMs' capability in adapting responses based on a few examples, i.e., few-shot performance. In Fig. 15, we plot the 3-shot and 5-shot performance for MatGPT- NeoX and LLaMA of 6.7B parameters, respectively. For some tasks, e.g., SCiQ, prompting with more examples help to improve the model performance, and NeoX with 5-shot performs the best. Compared to zero-shot, the accuracy improvement for SCiQ is up to about 5%. Overall, LLaMA performs slightly better in 3 out of 9 tasks while NeoX shows an edge in the other 3.

Observation (4) The LLM loss can serve as an indicator of model performance but does not fully correlate with downstream task performance. GPT-NeoX and LLaMA architectures perform similarly on generic downstream tasks. Data quality seems to be the distinguishing factor, and a larger vocabulary size for scientific texts likely helps.

Fine-tuning for Scientific Task Beyond generic question answering tasks, more importantly, it is crucial to demonstrate the scientific benefit of a domain-specific LLM. Most of the efforts [13, 19] so far have shown only the classification tasks, e.g., name entity recognition, but physical science is inherently numerical, with regression analysis serving as its core foundation. Here we show that MatGPT can be used to improve the prediction quality of an important property of a material, i.e., band gap, which plays a key role in determining the material's electrical and optical properties.

The experimental results support two complementary principles (i) LLMs of different architectures trained on larger datasets and a large number of parameters and (ii) models with higher hidden dimension sizes perform better than their simpler variants if these mentioned cases are true either individually or together. We first (i) present our band gap prediction results for different model settings where a set of domain-specific LLM variants (e.g. MatSciBERT, MatGPT-NeoX, etc.) is used in conjunction with a state-of-the-art GNN architecture, then (ii) show what intrinsic characteristics enable MatGPT embedding variants to perform better than each other and in general from MatSciBERT. Lastly, (iii) we will try to present evidence of the two characteristics mentioned above for various model architectures.

Table V shows the performance of band gap prediction resulting from a GNN model used in conjunction with MatSciBERT and MatGPT material embeddings, respectively. In the table, CGCNN [37], MEGNet [51], ALIGNN [40], and *MF-CGNN* [52] are GNN implementations. ALIGNN and *MF-CGNN* represent state-of-the-art performance without pre-training, and +Scibert and +GPT represent *MF-CGNN* augmented with MatSciBERT and MatGPT embeddings, respectively. We observe 5% and 8% improvement with +SciBERT and +GPT, respectively, over *MF-CGNN* alone. As band gap prediction is extremely challenging, our result demonstrates the promise of leveraging LLMs in scientific applications.

TABLE V: Predicting band gap with various GNN implementations. Mean absolute error (MAE) is reported with the best value marked in bold.

CGCNN	MEGNet	ALIGNN	MF-CGNN	+SciBERT	+GPT
0.388	0.33	0.218	0.215	0.204	**0.197**

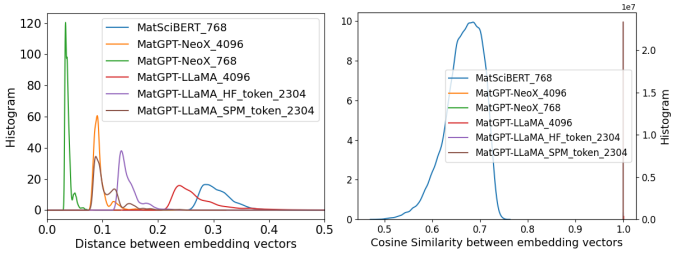
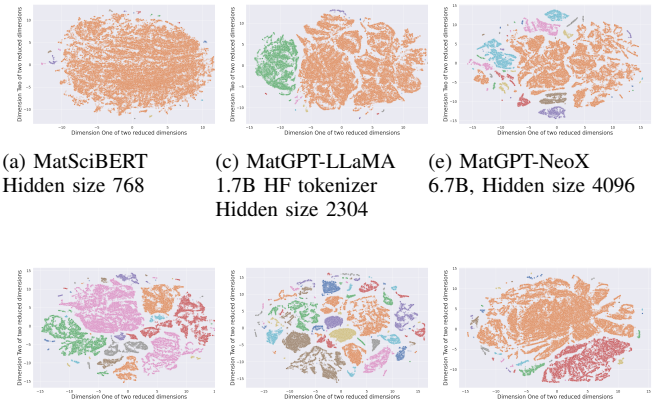


Fig. 16: Euclidean distances (Left) and cosine similarities (Right) between embedding vectors.

To understand the results in table V, we conducted fine-grained analysis on the MatsciBERT and MatGPT embedding vectors extracted via formula name of metal-organic materials (total 69240 materials) from respective models. The analyses uncover a couple of interesting facts from different viewpoints as follows that directly impact the regression results:

- MatGPT embedding vectors are closer to each other than MatSciBERT vectors. This characteristic enables MatGPT variants to perform better in regression tasks than MatSciBERT. Fig. 16 exhibits the density distribution of computed distances between embedding vectors for each LLM con-



(a) MatSciBERT Hidden size 768

(c) MatGPT-LLaMA 1.7B HF tokenizer Hidden size 2304

(e) MatGPT-NeoX 6.7B, Hidden size 4096

(b) MatGPT-NeoX Hidden size 768

(d) MatGPT-LLaMA 1.7B SPM tokenizer Hidden size 2304

(f) MatGPT-LLaMA 6.7B, Hidden size 4096

Fig. 17: Embedding clustering of material formulas for MatSciBERT and MatGPT variants after reducing dimensions by TSNE in tandem with PCA.

sidered in this study where the histograms of all the GPT variants are located near the y-axis.

- All the MatGPT vectors point in the same direction meaning that the cosines between the respective embedding vectors of all GPT variants tend to be 1. This phenomenon is demonstrated in Fig. 16 where density probabilities of cosine similarity between embedding vectors for all MatGPT variants seem to overlap on a vertical line. However, a different analysis demonstrates a model having smaller angles between MatSciBERT embedding vectors tends to lead to better performance than a model having larger angles between them. Although the embedding vectors of MatSciBERT and other GPT variants are derived from different architecture families, they both imply that smaller angles between embedding vectors perform better in bandgap prediction tasks.

To support our claim and establish the hypotheses mentioned above, we further investigate LLM embedding spaces by demonstrating clustering plots generated from respective embedding vectors in Fig. 17. Compared to MatGPT-NeoX (Fig. 17b) of similar size (hidden size 768), MatSciBERT embeddings form a very large cluster (Fig. 17a), which is an indicator of insufficient knowledge representation. Besides that, the embedding distances between MatSciBERT vectors are greater than the GPT variants' counterparts. This establishes a notion that data points are randomly disseminated in the low dimensional space, and hence perform a poor clustering. The optimal embedding clustering should reflect the characteristics of the prediction target — band gap: materials in nature can be classified by band gap into a few categories, i.e., conductor, semiconductor, or insulator, and the band gap for each category of materials has its characteristics. It seems the knowledge embedded in MatGPT can serve as additional distinctive features to improve the GNN regression. Referring to the clustering principle that the same cluster data points are closer to each other than the data points in different clusters, embeddings of GPT variants maintain a pertinent balance between the overall embedding distances and comprised distinctive features. For MatGPT-LLaMA with different data tokenization (HF and SPM), the SPM (Fig. 17d) embeddings seem to overly classify formulas, and this is consistent with the performance comparisons for language benchmarks (See Fig. 14).

Our best-performing model (See Table V) is MatGPT-NeoX with a dimension size of 4096 plotted in Fig. 17e shows cluster results consistent with the prediction result. On the other hand, clusters of MatGPT-LLaMA embedding variants of different dimension sizes are plotted in Fig. 17f, 17c, and 17d. They display either a higher number or a lower number of clusters, which seems to be the reason that LLaMA embeddings reduced performance as compared to NeoX.

Observation (5) For LLMs pretrained on scientific texts, model embeddings encode the knowledge of the literature. One hallucination-free usage of LLMs for science is the manipulation of embeddings for both classification and regression tasks.

V. Conclusion

LLMs are poised to potentially revolutionize the way we conduct science and it is critical to establish best practices for deploying them on public HPC platforms, especially leadership supercomputers, to ensure the democratized usage of subsequent breakthroughs. In this study, we have systematically investigated two popular open-source LLM architectures — GPT-NeoX and LLaMA. By designing controlled experiments, we carefully studied the effect of data tokenization, model architecure, and parameter count on both the behavior of training and validation losses, and the downstream language benchmarks. We outlined our observations and provided practical guidance for training LLMs on HPC systems.

Furthermore, based on the scientific corpus we collected, we have pre-trained a suite of LLMs for materials science. We then demonstrated our suite on a scientific downstream application by injecting the models' embeddings into the graph neural network for fine-tuning, and achieved state-of-the-art performance for the band-gap prediction.

Moreover, we reported the best-so-far LLM training performance on AMD GPUs, and demonstrated good scaling and energy efficiency on the Frontier supercomputer. The practical, end-to-end solution we establish can be applied to building LLMs on HPC systems in general.

Acknowledgment

This research was partially funded by a Lab Directed Research and Development project at Oak Ridge National Laboratory, a U.S. Department of Energy facility managed by UT-Battelle, LLC. This research used resources of the Oak Ridge Leadership Computing Facility (OLCF), which is a DOE Office of Science User Facility at the Oak Ridge National Laboratory supported by the U.S. Department of Energy under Contract No. DE-AC05-00OR22725.

References

[1] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

[2] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[3] Alec Radford, Karthik Narasimhan, Tim Salimans, Ilya Sutskever, et al. Improving language understanding by generative pre-training. 2018.

[4] Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B. Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. Scaling laws for neural language models. *CoRR*, abs/2001.08361, 2020. URL https://arxiv.org/abs/2001.08361.

[5] Zhi Hong, Aswathy Ajith, Gregory Pauloski, Eamon Duede, Carl Malamud, Roger Magoulas, Kyle Chard, and Ian Foster. Scholarbert: Bigger is not always better, 2022. URL https://arxiv.org/abs/2205.11342.

[6] Alex Andonian, Quentin Anthony, Stella Biderman, Sid Black, Preetham Gali, Leo Gao, Eric Hallahan, Josh Levy-Kramer, Connor Leahy, Lucas Nestler, Kip Parker, Michael Pieler, Shivanshu Purohit, Tri Songz, Wang Phil, and Samuel Weinbach. GPT-NeoX: Large Scale Autoregressive Language Modeling in PyTorch, 8 2021. URL https://www.github.com/eleutherai/gpt-neox.

[7] Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, Aurelien Rodriguez, Armand Joulin, Edouard Grave, and Guillaume Lample. Llama: Open and efficient foundation language models, 2023.

[8] Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, Dan Bikel, Lukas Blecher, Cristian Canton Ferrer, Moya Chen, Guillem Cucurull, David Esiobu, Jude Fernandes, Jeremy Fu, Wenyin Fu, Brian Fuller, Cynthia Gao, Vedanuj Goswami, Naman Goyal, Anthony Hartshorn, Saghar Hosseini, Rui Hou, Hakan Inan, Marcin Kardas, Viktor Kerkez, Madian Khabsa, Isabel Kloumann, Artem Korenev, Punit Singh Koura, Marie-Anne Lachaux, Thibaut Lavril, Jenya Lee, Diana Liskovich, Yinghai Lu, Yuning Mao, Xavier Martinet, Todor Mihaylov, Pushkar Mishra, Igor Molybog, Yixin Nie, Andrew Poulton, Jeremy Reizenstein, Rashi Rungta, Kalyan Saladi, Alan Schelten, Ruan Silva, Eric Michael Smith, Ranjan Subramanian, Xiaoqing Ellen Tan, Binh Tang, Ross Taylor, Adina Williams, Jian Xiang Kuan, Puxin Xu, Zheng Yan, Iliyan Zarov, Yuchen Zhang, Angela Fan, Melanie Kambadur, Sharan Narang, Aurelien Rodriguez, Robert Stojnic, Sergey Edunov, and Thomas Scialom. Llama 2: Open foundation and fine-tuned chat models, 2023.

[9] Tung Nguyen, Johannes Brandstetter, Ashish Kapoor, Jayesh Gupta, and Aditya Grover. Climax: A foundation model for weather and climate, 01 2023.

[10] Renqian Luo, Liai Sun, Yingce Xia, Tao Qin, Sheng Zhang, Hoifung Poon, and Tie-Yan Liu. BioGPT: generative pre-trained transformer for biomedical text generation and mining. *Briefings in Bioinformatics*, 23 (6), sep 2022. doi: 10.1093/bib/bbac409.

[11] Stanford HAI. Pubmedgpt, December 2022. URL https://hai.stanford.edu/news/stanford-crfm-introduces-pubmedgpt-27b. [Online; posted 15-December-2022].

[12] Zongrui Pei, Junqi Yin, Peter K Liaw, and Dierk Raabe. Toward the design of ultrahigh-entropy alloys via mining six million texts. *Nature Communications*, 14(1):54, 2023.

[13] Tanishq Gupta, Mohd Zaki, N. M. Anoop Krishnan, and Mausam. MatSciBERT: A materials domain language model for text mining and information extraction. *npj Computational Materials*, 8(1):102, May 2022. ISSN 2057-3960. doi: 10.1038/s41524-022-00784-w. URL https://www.nature.com/articles/s41524-022-00784-w.

[14] Guilherme Penedo, Quentin Malartic, Daniel Hesslow, Ruxandra Cojocaru, Alessandro Cappelli, Hamza Alobeidli, Baptiste Pannier, Ebtesam Almazrouei, and Julien Launay. The refinedweb dataset for falcon llm: Outperforming curated corpora with web data, and web data only, 2023.

[15] Jingfeng Yang, Hongye Jin, Ruixiang Tang, Xiaotian Han, Qizhang Feng, Haoming Jiang, Bing Yin, and Xia Hu. Harnessing the power of llms in practice: A survey on chatgpt and beyond, 2023.

[16] 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 H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 1877–1901, virtual, 2020. Curran Associates, Inc. URL https://proceedings.neurips.cc/paper/2020/file/1457c0d6bfcb4967418bfb8ac142f64a-Paper.pdf.

[17] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. Exploring the limits of transfer learning with a unified text-to-text transformer, 2023.

[18] Vahe Tshitoyan, John Dagdelen, Leigh Weston, Alexander Dunn, Ziqin Rong, Olga Kononova, Kristin A. Persson, Gerbrand Ceder, and Anubhav Jain. Unsupervised word embeddings capture latent knowledge from materi-

als science literature. *Nature*, 571(7763):95–98, 2019. ISSN 1476-4687. doi: 10.1038/s41586-019-1335-8. URL https://doi.org/10.1038/s41586-019-1335-8.

[19] Amalie Trewartha, Nicholas Walker, Haoyan Huo, Sanghoon Lee, Kevin Cruse, John Dagdelen, Alexander Dunn, Kristin A. Persson, Gerbrand Ceder, and Anubhav Jain. Quantifying the advantage of domain-specific pre-training on named entity recognition tasks in materials science. *Patterns*, 3(4):100488, 2022. ISSN 2666-3899. doi: https://doi.org/10.1016/j.patter.2022.100488. URL https://www.sciencedirect.com/science/article/pii/S2666389922000733.

[20] Tong Xie, Yuwei Wan, Wei Huang, Yufei Zhou, Yixuan Liu, Qingyuan Linghu, Shaozhou Wang, Chunyu Kit, Clara Grazian, Wenjie Zhang, and Bram Hoex. Large language models as master key: Unlocking the secrets of materials science with gpt, 2023.

[21] Jianlin Su, Yu Lu, Shengfeng Pan, Bo Wen, and Yunfeng Liu. RoFormer: Enhanced transformer with rotary position embedding. *Computing Research Repository*, arXiv:2104.09864, 2021. version 2.

[22] Alec Radford and Karthik Narasimhan. Improving language understanding by generative pre-training. 2018.

[23] 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, Jack W. Rae, Oriol Vinyals, and Laurent Sifre. Training compute-optimal large language models, 2022. URL https://arxiv.org/abs/2203.15556.

[24] Tri Dao, Daniel Y. Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. Flashattention: Fast and memory-efficient exact attention with io-awareness, 2022.

[25] AMD. Flash Attention on ROCm. https://github.com/ROCmSoftwarePlatform/flash-attentio, 2023. Accessed: 2023-9-30.

[26] AMD. Composable Kernel. https://github.com/ROCmSoftwarePlatform/composable_kernel, 2023. Accessed: 2023-9-30.

[27] Tri Dao. Flashattention-2: Faster attention with better parallelism and work partitioning, 2023.

[28] Jeff Rasley, Samyam Rajbhandari, Olatunji Ruwase, and Yuxiong He. Deepspeed: System optimizations enable training deep learning models with over 100 billion parameters. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '20, page 3505–3506, New York, NY, USA, 2020. Association for Computing Machinery. ISBN 9781450379984. doi: 10.1145/3394486.3406703. URL https://doi.org/10.1145/3394486.3406703.

[29] Deepak Narayanan, Mohammad Shoeybi, Jared Casper, Patrick LeGresley, Mostofa Patwary, Vijay Korthikanti, Dmitri Vainbrand, Prethvi Kashinkunti, Julie Bernauer, Bryan Catanzaro, Amar Phanishayee, and Matei Zaharia. Efficient large-scale language model training on gpu clusters using megatron-lm. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '21, New York, NY, USA, 2021. Association for Computing Machinery. ISBN 9781450384421. doi: 10.1145/3458817.3476209. URL https://doi.org/10.1145/3458817.3476209.

[30] OLCF. Frontier. https://docs.olcf.ornl.gov/systems/frontier_user_guide.html, 2023. Accessed: 2023-9-30.

[31] Yang You, Jing Li, Sashank Reddi, Jonathan Hseu, Sanjiv Kumar, Srinadh Bhojanapalli, Xiaodan Song, James Demmel, Kurt Keutzer, and Cho-Jui Hsieh. Large batch optimization for deep learning: Training bert in 76 minutes, 2020.

[32] Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *International Conference on Learning Representations*, 12 2014.

[33] Junqi Yin, Sajal Dash, John Gounley, Feiyi Wang, and Georgia Tourassi. Evaluation of pre-training large language models on leadership-class supercomputers. *Journal of Supercomputing*, 6 2023. doi: 10.1007/s11227-023-05479-7.

[34] Q. Anthony, A. Awan, J. Rasley, Y. He, A. Shafi, M. Abduljabbar, H. Subramoni, and D. Panda. Mcr-dl: Mix-and-match communication runtime for deep learning. In *2023 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pages 996–1006, Los Alamitos, CA, USA, may 2023. IEEE Computer Society. doi: 10.1109/IPDPS54959.2023.00103. URL https://doi.ieeecomputersociety.org/10.1109/IPDPS54959.2023.00103.

[35] Leo Gao, Jonathan Tow, Stella Biderman, Sid Black, Anthony DiPofi, Charles Foster, Laurence Golding, Jeffrey Hsu, Kyle McDonell, Niklas Muennighoff, Jason Phang, Laria Reynolds, Eric Tang, Anish Thite, Ben Wang, Kevin Wang, and Andy Zou. A framework for few-shot language model evaluation, September 2021. URL https://doi.org/10.5281/zenodo.5371628.

[36] Junqi Yin, Sajal Dash, Feiyi Wang, and Mallikarjun Shankar. Forge: Pre-training open foundation models for science. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '23, New York, NY, USA, 2023. Association for Computing Machinery. ISBN 9798400701092. doi: 10.1145/3581784.3613215. URL https://doi.org/10.1145/3581784.3613215.

[37] Tian Xie and Jeffrey C Grossman. Crystal graph convolutional neural networks for an accurate and interpretable prediction of material properties. *Physical Review Letters*, 120(14):145301, 2018.

[38] Andrew S Rosen, Shaelyn M Iyer, Debmalya Ray, Zhenpeng Yao, Alán Aspuru-Guzik, Laura Gagliardi, Justin M Notestein, and Randall Q Snurr. Machine learning the quantum-chemical properties of metal–organic frameworks for accelerated materials discovery. *Matter*, 4(5):1578–1597, 2021.

[39] Cheol Woo Park and Chris Wolverton. Developing an improved crystal graph convolutional neural network framework for accelerated materials discovery. *Physical Review Materials*, 4(6):063801, 2020.

[40] Kamal Choudhary and Brian DeCost. Atomistic line graph neural network for improved materials property predictions. *npj Computational Materials*, 7(1):1–8, 2021.

[41] Jiucheng Cheng, Chunkai Zhang, and Lifeng Dong. A geometric-information-enhanced crystal graph network for predicting properties of materials. *Communications Materials*, 2(1):1–11, 2021.

[42] Ruihan Wang, Yeshuang Zhong, Leming Bi, Mingli Yang, and Dingguo Xu. Accelerating discovery of metal–organic frameworks for methane adsorption with hierarchical screening and deep learning. *ACS Applied Materials & Interfaces*, 12(47):52797–52807, 2020.

[43] A Jain, SP Ong, G Hautier, W Chen, WD Richards, S Dacek, S Cholia, D Gunter, D Skinner, G Ceder, et al. The materials project: A materials genome approach to accelerating materials innovation. apl materials, 1 (1): 011002, 2013.

[44] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic Differentiation in PyTorch. 2017.

[45] Junqi Yin, Aristeidis Tsaris, Sajal Dash, Ross Miller, Feiyi Wang, and Mallikarjun (Arjun) Shankar. Comparative evaluation of deep learning workloads for leadership-class systems. *BenchCouncil Transactions on Benchmarks, Standards and Evaluations*, 1(1):100005, 2021. ISSN 2772-4859. doi: https://doi.org/10.1016/j.tbench.2021.100005. URL https://www.sciencedirect.com/science/article/pii/S2772485921000053.

[46] Johannes Welbl, Nelson F. Liu, and Matt Gardner. Crowdsourcing multiple choice science questions. *ArXiv*, abs/1707.06209, 2017. URL https://api.semanticscholar.org/CorpusID:1553193.

[47] Yonatan Bisk, Rowan Zellers, Ronan Le bras, Jianfeng Gao, and Yejin Choi. PIQA: Reasoning about physical commonsense in natural language. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 7432–7439, 04 2020. doi: 10.1609/aaai.v34i05.6239. URL https://ojs.aaai.org/index.php/AAAI/article/view/6239.

[48] Todor Mihaylov, Peter Clark, Tushar Khot, and Ashish Sabharwal. Can a suit of armor conduct electricity? A new dataset for open book question answering. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 2381–2391, Brussels, Belgium, October-November 2018. Association for Computational Linguistics. doi: 10.18653/v1/D18-1260. URL https://aclanthology.org/D18-1260.

[49] Peter Clark, Isaac Cowhey, Oren Etzioni, Tushar Khot, Ashish Sabharwal, Carissa Schoenick, and Oyvind Tafjord. Think you have solved question answering? try arc, the ai2 reasoning challenge. *ArXiv*, abs/1803.05457, 2018.

[50] Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. Measuring massive multitask language understanding. *Computing Research Repository*, arXiv:2009.03300, 2021. version 3.

[51] Chi Chen, Weike Ye, Yunxing Zuo, Chen Zheng, and Shyue Ping Ong. Graph networks as a universal machine learning framework for molecules and crystals. *Chemistry of Materials*, 31(9):3564–3572, 2019.

[52] Guojing Cong and Victor Fung. Improving materials property predictions for graph neural networks with minimal feature engineering*. *Machine Learning: Science and Technology*, 4(3):035030, aug 2023. doi: 10.1088/2632-2153/acefab. URL https://dx.doi.org/10.1088/2632-2153/acefab.