Techniques to Make Large Language Models Smaller: An Explainer

By Kyle A. Miller and Andrew J. Lohn

Large language models (LMs) are often difficult and expensive to train and use.[1] For example, the models that power ChatGPT are large—with at least 175 billion parameters—and cost millions of dollars in compute to train. Moreover, dozens of high-end processors are needed to run the models (also known as inference), which becomes expensive when ChatGPT is supporting millions of user requests daily.[2] These resources are out of reach for most individuals and less well-resourced organizations, including many in academia. At first glance, it seems that there is a high barrier to entry; a "moat" surrounding large technology companies that have the resources to develop large LMs and use them at scale.[3]

However, there are techniques to produce smaller and more efficient LMs that require fewer resources to develop and operate—and many of them are openly available online for anyone to use. The combination of both small (i.e., easy to use) and open (i.e., easy to access) could have significant implications for artificial intelligence development, including:

- Accelerating AI research and innovation. These models can be developed more quickly, in more dynamic and customized ways, and by more people than large proprietary models.
- Lowering the barrier to entry in certain areas of AI development, accelerating model proliferation, and complicating governmental regulatory efforts. This has second-order implications for AI safety, misuse, and malicious use.
- Allowing U.S. competitors to be fast followers and leverage the open software for their own ends.

This explainer outlines some of the techniques used to create smaller models. Presently, smaller models are generally less capable than the largest LMs, but there are indications that they are still valuable to a broad set of researchers and developers.

Techniques

Here, we assess two broad groups of techniques to build smaller LMs:

A. **Start Small and Stay Efficient:** Develop smaller models that use fewer resources from the outset, or fine-tune and operate models in a way that requires fewer resources.

B. **Get Small (Compression and Knowledge Distillation):** Make larger models smaller so they are easier to train and run, or transfer the knowledge of larger models to smaller models.

One reason to **start small** is to avoid the computing overhead of large models during training and inference**.** This option promotes a degree of ground-up efficiency and reduces the resources (e.g., compute and memory) used across many parts of model development and deployment.

We outline four techniques to start small and stay efficient.

1. **Design a model with fewer parameters**. Here, we generally mean models that can run effectively on a single high-end GPU. These are typically models with fewer than 7 billion parameters, but depending on various factors, can even include models with up to 70 billion parameters. Such models typically require less compute and memory to run or fine-tune (depending on the task and quantity of data), although they usually do not perform as well as larger models.[4]

2. **Prioritize data quality over data quantity**. Use less, but better, data to train smaller models on a lower compute budget without sacrificing performance.[5] Higher-quality data can be harder to collect and curate, but you typically need less of it. Since model size and dataset size are often scaled together, a smaller dataset may allow for a smaller model.[6]

3. **Build a model for a particular domain or task**. Large LMs that exhibit general knowledge and wide-ranging capabilities need many parameters to encode that vast knowledge. Smaller models designed for a specific task may require fewer parameters to encode the more limited knowledge required for that task.

4. **Fine-tune models efficiently**. Fine-tuning involves feeding more data to a trained model so it hones the ability to perform certain tasks. While this requires less compute and memory than the original training, it can still be difficult to fine-tune an entire model. To circumvent this bottleneck, there are techniques to further reduce those resource demands, such as by updating the most important parts of a model while keeping the rest of it “frozen.”[7]

Next are techniques to **get small** by compressing larger language models or distilling their knowledge. This generally involves making larger models smaller and more efficient, or distilling knowledge from larger models to smaller models, thereby reducing the compute and memory needed to run or improve them.[8]

We outline three techniques to get small. This is not an exhaustive list, rather an overview of some popular techniques.

1. **Reduce the memory used for each parameter in a model.** Each parameter is held using some number of bits. For example, a high-precision model may have 64 bits for each parameter, which can be reduced to 16 bits. With fewer bits, each parameter is less precise but requires less memory and compute to run or train. Importantly, there are limits to how much precision can be reduced, with 2-bit precision being considered the extreme lower bound.[9]

2. **Transfer knowledge from a large model to a smaller one.** One technique is to “distill” a subset of a large model's superior knowledge and capabilities into a smaller model, analogous to how an older teacher transfers some of their knowledge to a younger student. Here, the outputs of a larger, more capable model are used to fine-tune a smaller model. Researchers are still determining how well smaller student models can retain the generality of their larger teacher models.[10] However, it may be possible for developers to improve smaller and cheaper models by distilling knowledge from expensive and proprietary models, such as those behind ChatGPT.[11]

3. **Prune redundant or unnecessary parts of a larger model.** This reduces the number of parameters with only a limited impact on performance because the pruned parameters are chosen specifically because of their low importance.[12]

Ultimately, small and open models are impacting AI innovation, proliferation, regulation, and international competition—and the above methods are lowering the barriers to entry to develop and use them. Although the performance of these models is generally poorer than large, proprietary models, many people opt to use them due to their accessibility and customizability. Moreover, if the performance of these models continues to increase, even on narrow tasks, then their use will likely also increase. As these open models continue to multiply, it will be increasingly difficult to track how they are used or control their diffusion through policies such as compute-based export controls.

Endnotes

[1] Jack W. Rae et al., “Scaling Language Models: Methods, Analysis & Insights from Training Gopher,” arXiv preprint arXiv:2112.11446 (2022) https://arxiv.org/abs/2112.11446.

[2] Some estimate that OpenAI spends about $700,000 per day to service ChatGPT users.

Aaron Mok, “ChatGPT could cost over $700,000 per day to operate. Microsoft is reportedly trying to make it cheaper,” *Business Insider*, April 2023, https://www.businessinsider.com/how-much-chatgpt-costs-openai-to-run-estimate-report-2023-4.

[3] Dylan Patel and Afzal Ahmad, “Google ‘We Have No Moat, And Neither Does OpenAI,’” *Semianalysis*, May 2023, https://www.semianalysis.com/p/google-we-have-no-moat-and-neither.

[4] Pre-training smaller models can still be resource-intensive.

“Llama 2: Open Foundation and Fine-Tuned Chat Models,” *Meta AI*, July 2023, https://ai.meta.com/research/publications/llama-2-open-foundation-and-fine-tuned-chat-models/; Kourosh Hakhamaneshi and Rehaan Ahmad, “Fine-Tuning Llama-2: A Comprehensive Case Study for Tailoring Models to Unique Applications,” *AnyScale*, August 2023, https://www.anyscale.com/blog/fine-tuning-llama-2-a-comprehensive-case-study-for-tailoring-models-to-unique-applications; Jared Kaplan et al., “Scaling Laws for Neural Language Models,” arXiv preprint arXiv:2001.08361 (2020), https://arxiv.org/abs/2001.08361; Philipp Schmid et al., “Llama 2 is here - get it on Hugging Face,” *HuggingFace*, July 2023, https://huggingface.co/blog/llama2; “ExLlamaV2,” *GitHub*, September 2023, https://github.com/turboderp/exllamav2.

[5] Suriya Gunasekar et al., “Textbooks Are All You Need,” arXiv preprint arXiv:2306.11644 (2023), https://arxiv.org/abs/2306.11644.

[6] Jordan Hoffmann et al., “An empirical analysis of compute-optimal large language model training,” *DeepMind*, 2022, https://proceedings.neurips.cc/paper_files/paper/2022/hash/c1e2faff6f588870935f114ebe04a3e5-Abstract-Conference.html.

[7] This technique is called “Low-Rank Adaption (LoRA),” of which there are several variants.

Edward Hu et al., “LoRA: Low-Rank Adaptation of Large Language Models,” arXiv preprint arXiv:2106.09685 (2021), https://arxiv.org/abs/2106.09685; Qingru Zhang et al., “Adaptive Budget Allocation for Parameter Efficient Fine-Tuning,” arXiv preprint arXiv:2303.10512 (2023), https://arxiv.org/abs/2303.10512; Tim Dettmers et al., “QLORA: Efficient Finetuning of Quantized LLMs,” arXiv preprint arXiv:2305.14314 (2023), https://arxiv.org/abs/2305.14314.

[8] Knowledge distillation is a compression technique, but we distinguish it for the purpose of clarity; Xunyu Zhu et al., “A Survey on Model Compression for Large Language Models,” arXiv preprint arXiv:2308.07633 (2023), https://arxiv.org/abs/2308.07633.

[9] This technique is called “quantization.”

Elias Frantar et al., “GPTQ: Accurate Post-Training Quantization for Generative Pre-Trained Transformers,” arXiv preprint arXiv:2210.17323 (2023), https://arxiv.org/abs/2210.17323; Amir Gholami et al., “A Survey of Quantization Methods for Efficient Neural Network Inference,” arXiv preprint arXiv:2103.13630 (2021), https://arxiv.org/abs/2103.13630; “ExLlamaV2,” *GitHub*, September 2023, https://github.com/turboderp/exllamav2.

[10] Samuel Stanton et al., “Does Knowledge Distillation Really Work?” arXiv preprint arXiv:2106.05945 (2021), https://arxiv.org/abs/2106.05945.

[11] This technique is called “knowledge distillation.”

Jianping Gou et al., “Knowledge Distillation: A Survey,” arXiv preprint arXiv:2006.05525 (2021), https://arxiv.org/abs/2006.05525; Yuxian Gu et al., “Knowledge Distillation of Large Language Models,” arXiv preprint arXiv:2306.08543 (2023), https://arxiv.org/abs/2306.08543; Rishabh Agarwal et al., “Generalized Knowledge Distillation for Auto-regressive Sequence Models,” arXiv preprint arXiv:2306.13649 (2023), https://arxiv.org/abs/2306.13649.

[12] There are four types of pruning (i.e., sparsification) techniques, each of which involves pruning different parts of a neural network.