Chapter 8

The Compute Schism

The scale of deep architectures is critical to their performance and, as we saw in § 3.7, Large Language Models in particular may require amounts of memory and computation that greatly exceed those of consumer hardware.

While training such a model from scratch requires resources available only to large corporations or public bodies, techniques have been developed to allow inference and adaptation to specific tasks under strong resource constraints. Allowing to run models locally instead of through a provider may be highly desirable for cost or confidentiality reasons.

8.1 Prompt Engineering

The simplest strategy to specialize or improve a Large Language Model with a limited computational budget is to use prompt engineering, that is, to carefully craft the beginning of the text sequence to bias the autoregressive process [Sahoo et al., 2024]. This approach moves a part of the information traditionally encoded in the model's parameters to the input.

We saw in § 7.1 a simple example of few-shot prediction, to use an LLM for a text classification task without fine-tuning. A long and sophisticated prompt allows generalizing this strategy to complex tasks.

Since the prompt's role is to leverage the "good" biases that were present in the training set, it benefits from surprising strategies such as stating that the response is generated by a skilled professional [Xu et al., 2023].

The context size of a language model, that is, the number of tokens it can operate on, directly modulates the quantity of information that can be provided in the prompt. This is mostly constrained by the computational cost of standard attention models, which is quadratic with the context size (see § 4.8).

Q: Gina has 105 beans, she gives 23 beans to Bob, and prepares a soup with 53 beans. How many beans are left? A: There are 29 beans left.

Q: I prepare 53 pancakes, eat 5 of them and give 7 to Gina. I then prepare 26 more. How many pancakes are left? A: **27 pancakes are left.**

Q: Gina has 105 beans, she gives 23 beans to Bob, and prepares a soup with 53 beans. How many beans are left? A: Let's proceed step by step: Gina has 105 beans, she gives 23 beans to Bob (82 left), and prepares a soup with 53 beans (29 left). So there are 29 beans left.

Q: I prepare 53 pancakes, eat 5 of them and give 7 to Gina. I then prepare 26 more. How many pancakes are left? A: Let's proceed step by step: **53 pancakes, eat 5 of them (48 left), give 7 to Gina (41 left), prepare 26 more (67 left). So there are 67 pancakes left.**

Figure 8.1: *Example of a chain-of-thought to improve the response of the Llama-3-8B base model. In the two examples, the beginning of the text in normal font is the prompt, and the generated part is indicated in bold. The generation without chain-of-thought (top) leads to an incorrect answer, while the generation with it (bottom) generates a correct answer, by explicitly producing multiple simple arithmetic operations.*

Chain of Thought

A remarkable type of prompting aims at making the model generate intermediate steps before generating the response itself.

Such a chain-of-thought is composed of successive steps that are simpler, hence have been better modeled during training, and are predicted more deterministically [Wei et al., 2022; Kojima et al., 2022]. See Figure 8.1 for an example.

Retrieval-Augmented Generation

Prompt engineering can also be put to use to connect a language model to an external knowledge base. It plays the role of a smart interface that allows the end user to formulate questions in natural language and get back a response that combines information that is not encoded in the model's parameters [Lewis et al., 2020].

For such Retrieval-Augmented Generation (RAG), an embedding model is used to retrieve documents whose embedding is correlated to that of the user's query. Then, a prompt is constructed by joining these retrieved documents with instructions to combine them, and the generative model produces the response to the user.

8.2 Quantization

Although training or generating multiple streams can benefit from high-end parallel computing devices, deployment of a Large Language Model for individual use requires generally single-stream inference, which is bounded by memory size and speed far more than by computation.

As stated in § 2.1, parameters, activations, and gradients are usually encoded with 32 or 16 bits. The precision it provides is necessary for training, to allow gradual changes to accumulate.

However, since activations are the sums of many terms, quantization during inference is mitigated by an averaging effect. This is even more true with large architectures, and models quantized down to 6 or 4 bits per parameter exhibit remarkable performance. Additionally to reducing the memory footprint, quantization also improves inference speed significantly.

This has motivated the development of software to quantize existing models with Post-Training Quantization, and run them in single-stream inference on consumer hardware, such as llama.cpp [Llama.cpp, 2023]. This framework implements multiple formats, that apply specific

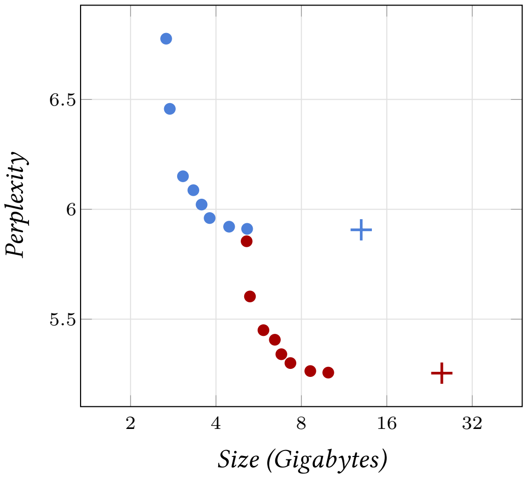

Figure 8.2: *Perplexity of quantized versions of the language models Llama-7B (blue) and 13B (red) [Touvron et al., 2023] on the wikitext corpus, as a function of the parameters' memory footprint. The crosses are the original FP16 models and the dots correspond to different levels of quantization with llama.cpp [Llama.cpp, 2023].*

quantization levels for the different weight matrices of a language model. For instance the quantization may use more bits for the W^{V} weights of the attention blocks, and for the weights of the feed-forward blocks.

An example of llama.cpp's quantization is $Q4\_1$.

It quantizes individually sub-blocks of 32 entries of the original weight matrix by storing for each a scaling factor d and a bias m in the original FP16 encoding, and encoding each entry x with 4 bits as a value $q \in \{0,\ldots,2^4-1\}$. The resulting de-quantized value being $\tilde{x} = dq+m$.

Such a block was encoded originally as 32 values in FP16, hence 64 bytes, while the quantized version needs 4 bytes for q and m and $32 \cdot 4$ bits = 16 bytes for the entries, hence a total of 20 bytes.

Such an aggressive quantization surprisingly degrades only marginally the performance of the models, as illustrated on Figure 8.2.

An alternative to Post-Training Quantization is Quantization-Aware Training that applies quantization during the forward pass but keeps high-precision encoding of parameters and gradients, and propagates the gradients during the backward pass as if there was no quantization [Ma et al., 2024].

8.3 Adapters

As we saw in § 3.6, fine-tuning is a key strategy to reuse pre-trained models. Since it aims at making only minor changes to an existing model, techniques have been developed that add components with few parameters, referred to as adapters, to the pre-trained architecture, and freeze all the original parameters [Houlsby et al., 2019].

The current dominant method is the Low-Rank Adaptation (LoRA), which adds low-rank corrections to some of the model's weight matrices [Hu et al., 2021].

Formally, given a linear operation of the form $XW^\top$, where X is a $N \times D$ tensor of activations for a batch of N samples, and W is a $C \times D$ weight matrix, the LoRA adapter replaces this operation with $X(W+BA)^\top$, where A and B are two trainable matrices of size $R \times D$ and $C \times R$ respectively, with $R \ll \min(C,D)$, and the matrix W is removed from the trainable parameters. The matrix A is initialized with random Gaussian values, and B is set to zero, so that the fine-tuning starts with a model that computes an output identical to that of the original one.

The total number of parameters to optimize with this approach is generally a few percent of the number of parameters in the original model.

The standard procedure to fine-tune a transformer with such adapters is to change only the weight matrices in the attention blocks, and to keep the MLP of the feed-forward blocks unchanged. The same strategy has been used successfully to tune diffusion denoising models by fine-tuning the attention blocks responsible for the text-based conditioning.

Since fine-tuning with LoRA adapters drastically reduces the number of trainable parameters, it reduces the memory footprint required by optimizers such as Adam, which generally store two running average per parameter to optimize. Also, it reduces slightly the computation during the backward pass.

For commercial applications that require a large number of fine-tuned models, the AB pairs can be stored separately from the original model, which has to be stored only once. And finally, contrary to other type of adapters, the modifications can be integrated into the original architecture, simply by adding AB to W, resulting in an architecture and parameter count for inference strictly identical to that of the base model.

We saw that quantization degrade models' accuracy only marginally. However, gradient descent requires high precision in both the gradient and the trained parameters, to allow the accumulation of small changes. The QLoRA approach combines a quantized base model and unquantized Low-Rank Adaptation to reduce the memory requirement even more [Dettmers et al., 2023].

8.4 *Model merging*

An alternative to the fine-tuning and prompting methods seen in the previous sections consists of combining multiple models with diverse capabilities into a single one, without additional training.

Model merging relies on the compatibility between multiple fine-tuned versions of a base model.

Ilharco et al. [2022] showed that models obtained by fine-tuning a CLIP base model on several image classification data-sets can be combined in the parameter space, where they exhibit Task Arithmetic properties.

Formally, let θ be the parameter vector of a pre-trained model, and for $t = 1, \ldots, T$, let θ_t and $\tau_t = \theta_t - \theta$ be respectively the parameters after fine-tuning on task t and the corresponding residual. Experiments show that the model with parameters $\theta + \tau_1 + \cdots + \tau_T$ exhibits multi-task capabilities. Similarly, subtracting a τ_t degrades the performance on the corresponding task.

Methods have been developed to reduce the interference between the different residuals and improve the performance when the number of tasks increases [Yadav et al., 2023; Yu et al., 2023].

An alternative to merging models in parameter space is to recombine their layers. Akiba et al. [2024] combine merging the parameters and re-combining layers, and rely on a stochastic optimization to deal with the combinatorial explosion. Experiments with three fine-tuned versions of Mistral-7B [Jiang et al., 2023] show that combining these two merging strategies outperforms both of them.