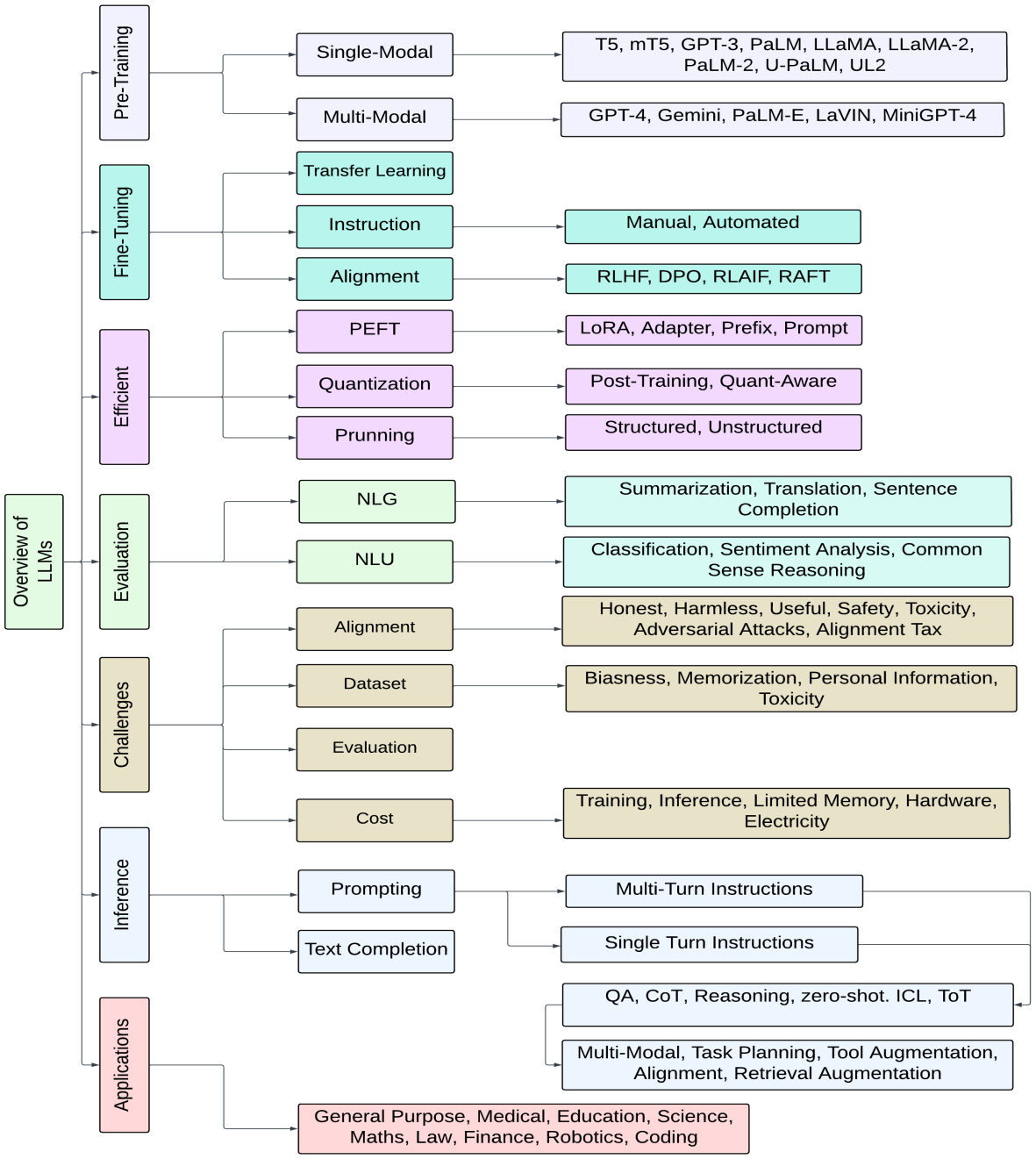

Figure 3: A broader overview of LLMs, dividing LLMs into seven branches: 1. Pre-Training 2. Fine-Tuning 3. Efficient 4. Inference 5. Evaluation 6. Applications 7. Challenges

multi-modal LLMs, augmented LLMs, LLMs-powered agents, datasets, evaluation, etc.

We loosely follow the existing terminology to ensure a standardized outlook of this research direction. For instance, following [50], our survey discusses pre-trained LLMs with 10B parameters or more. We refer the readers interested in smaller pre-trained models to [51, 52, 53].

The organization of this paper is as follows. Section 2 discusses the background of LLMs. Section 3 focuses on LLMs overview, architectures, training pipelines and strategies, fine-tuning, and utilization in different domains. Section 4 highlights the configuration and parameters that play a crucial role in the functioning of these models. Summary and discussions are presented in section 3.8. The LLM training and evaluation, datasets, and benchmarks are discussed in section 5, followed by challenges and future directions, and conclusion in sections 7 and 8, respectively.

2. Background

We provide the relevant background to understand the fundamentals related to LLMs in this section. We briefly discuss necessary components in LLMs and refer the readers interested in details to the original works.

2.1. Tokenization

Tokenization [59] is an essential pre-processing step in LLM training that parses the text into non-decomposing units called tokens. Tokens can be characters, subwords [60], symbols [61], or words, depending on the tokenization process. Some of the commonly used tokenization schemes in LLMs include wordpiece [62], byte pair encoding (BPE) [61], and unigramLM [60]. Readers are encouraged to refer to [63] for a detailed survey.

2.2. Encoding Positions

The transformer processes input sequences in parallel and independently of each other. Moreover, the attention module in the transformer does not capture positional information. As a result, positional encodings were introduced in transformer [64], where a positional embedding vector is added to the token embedding. Variants of positional embedding include absolute, relative, or learned positional encodings. Within relative encoding, Alibi and RoPE are two widely used positional embeddings in LLMs.

Alibi [65]: It subtracts a scalar bias from the attention score that increases with the distance between token positions. This favors using recent tokens for attention.

RoPE [66]: It rotates query and key representations at an angle proportional to the token absolute position in the input sequence, resulting in a relative positional encoding scheme which decays with the distance between the tokens.

2.3. Attention in LLMs

Attention assigns weights to input tokens based on importance so that the model gives more emphasis to relevant tokens. Attention in transformers [64] calculates query, key, and value mappings for input sequences, where the attention score is obtained by multiplying the query and key, and later used to weight values. We discuss different attention strategies used in LLMs below.

Self-Attention [64]: Calculates attention using queries, keys, and values from the same block (encoder or decoder).

Cross Attention: It is used in encoder-decoder architectures, where encoder outputs are the queries, and key-value pairs come from the decoder.

Sparse Attention [67]: Self-attention has $O(n^2)$ time complexity which becomes infeasible for large sequences. To speed up the computation, sparse attention [67] iteratively calculates attention in sliding windows for speed gains.

Flash Attention [68]: Memory access is the major bottleneck in calculating attention using GPUs. To speed up, flash attention employs input tiling to minimize the memory reads and writes between the GPU high bandwidth memory (HBM) and the on-chip SRAM.

2.4. Activation Functions

The activation functions serve a crucial role in the curve-fitting abilities of neural networks [69]. We discuss activation functions used in LLMs in this section.

ReLU [70]: The Rectified linear unit (ReLU) is defined as:

$$ReLU(x) = max(0, x) \tag{1}$$

GeLU [71]: The Gaussian Error Linear Unit (GeLU) is the combination of ReLU, dropout [72] and zoneout [73].

GLU variants [74]: The Gated Linear Unit [75] is a neural network layer that is an element-wise product ($\otimes$) of a linear transformation and a sigmoid transformed (σ) linear projection of the input given as:

$$GLU(x, W, V, b, c) = (xW + b) \otimes \sigma(xV + c), \tag{2}$$

where X is the input of layer and l, W, b, V and c are learned parameters. Other GLU variants [74] used in LLMs are:

$$\begin{aligned} ReGLU(x, W, V, b, c) &= max(0, xW + b)\otimes, \\ GEGLU(x, W, V, b, c) &= GELU(xW + b) \otimes (xV + c), \\ SwiGLU(x, W, V, b, c, \beta) &= Swish\beta(xW + b) \otimes (xV + c). \end{aligned}$$

2.5. Layer Normalization

Layer normalization leads to faster convergence and is an integrated component of transformers [64]. In addition to LayerNorm [76] and RMSNorm [77], LLMs use pre-layer normalization [78], applying it before multi-head attention (MHA). Pre-norm is shown to provide training stability in LLMs. Another normalization variant, DeepNorm [79] fixes the issue with larger gradients in pre-norm.

2.6. Distributed LLM Training

This section describes distributed LLM training approaches briefly. More details are available in [13, 37, 80, 81].

Data Parallelism: Data parallelism replicates the model on multiple devices where data in a batch gets divided across devices. At the end of each training iteration weights are synchronized across all devices.

Tensor Parallelism: Tensor parallelism shards a tensor computation across devices. It is also known as horizontal parallelism or intra-layer model parallelism.

Pipeline Parallelism: Pipeline parallelism shards model layers across different devices. This is also known as vertical parallelism.

Model Parallelism: A combination of tensor and pipeline parallelism is known as model parallelism.

3D Parallelism: A combination of data, tensor, and model parallelism is known as 3D parallelism.

Optimizer Parallelism: Optimizer parallelism also known as zero redundancy optimizer [37] implements optimizer state partitioning, gradient partitioning, and parameter partitioning across devices to reduce memory consumption while keeping the communication costs as low as possible.