Chapter 1
From Deep Neural Language Models to LLMs

Andrei Kucharavy

Abstract *Large Language Models* (LLMs) are scaled-up instances of Deep Neural Language Models—a type of *Natural Language Processing* (NLP) tools trained with *Machine Learning* (ML). To best understand how LLMs work, we must dive into what technologies they build on top of and what makes them different. To achieve this, an overview of the history of LLMs development, starting from the 1990s, is provided before covering the counterintuitive purely probabilistic nature of the Deep Neural Language Models, continuous token embedding spaces, recurrent neural networks-based models, what self-attention brought to the table, and finally, why scaling Deep Neural Language Models led to a qualitative change, warranting a new name for the technology.

1.1 What LLMs Are and What LLMs Are Not

Generative Machine Learning—often referred to as Generative AI—has seemingly taken the world by storm in late 2022–2023, with modern *Large Language Models* (LLMs) demonstrating human-like performances across a range of tasks, leading to heated debates as to whether it needed to be included into every product and process.

However, the concept of generative language models is significantly older. What made 2022–2023 LLMs so attractive is their compliance with the expectations of the general public as to how an AI assistant could behave, made possible with instructional fine-tuning, followed by a polish with the *Reinforcement Learning from Human Feed-back* (RLHF) (Chap. 4). However, already before that—in 2022—trials were run with conversationally fine-tuned models perfectly impersonating internet forum users. In 2020, base LLMs could already write journals and blogs in the hands of competent users. Before the Transformer model allowed the model size and datasets to out-scale most hardware and most internet, *Recurrent Neural*

A. Kucharavy (✉)
HES-SO Valais-Wallis, Sierre, Switzerland
e-mail: andrei.kucharavy@hevs.ch

A. Kucharavy et al. (eds.), *Large Language Models in Cybersecurity*,
https://doi.org/10.1007/978-3-031-54827-7_1

Network (RNN)—based LLMs such as ELMo could perform text analysis and text generation. Before that, smaller RNN generative models could do specialized tasks such as autocompletion of news articles. Before Deep Neural Language Models, there were Hidden Markov Model-based models. Before that—rule-based text generation bots, ..., all the way back to 1966's ELIZA that could fool its users into believing they were speaking to a sentient being using only pattern matching and substitution rules, all while running with 18 kB of RAM and less processing power than today's USB chargers.

Because of that, there is a general disconnect in the vocabulary used by today's general public who discovered the progress of last decades in Generative ML-based NLP and the practitioners and scientists who developed and implemented technologies that made ChatGPT, Copilots, and other LLaMAs possible.

To keep the vocabulary consistent with prior research, I will follow the historical conventions of the ML-NLP community before 2022 in this book. Specifically, I use the term LLMs to designate post-ELMo models (Chap. 3). Here, I designate as LLMs Deep Neural Language Models that:

1. Perform a probabilistic token regression based on training data and user input
2. Have over 100 million parameters
3. Were trained on over 1 billion tokens
4. Or are smaller members of the families whose larger models satisfy the conditions above

This definition directly follows the first papers that introduced the concept of "Large Language Models": [1, 2], and is representative of the smooth scaling of capabilities for LLMs as they increase in size from 100M to 1000B parameters [3, 4].

This definition means that while the original Transformer is not an LLM, RNN-based ELMo is. Similarly, the BERT model that is generally used for text classification rather than generation is an LLM, just like the translation-tuned T5. Including smaller models within LLM families allows us also to include models historically considered as LLMs, such as DistilBERT with 66M parameters or even CODEX models with 12M parameters. Similarly, this means I am not differentiating LLMs generating text from the ones generating code, binary, pass requests to search engines, or accepting images as inputs.

1.2 Principles of LLMs

1.2.1 Deep Neural Language Models

LLMs are direct descendants of *Deep Neural Language Models* (DNLMs), differing only in model and training dataset size. Understanding Deep Neural Language

Models requires several important departures from an intuitive natural language understanding.

First, from the point of view of DNLMs, letters, words, or even sentences do not exist (as intuitively defined by humans). Instead, they operate in a continuous *embedding space*, where segments of characters of a fixed length are interpreted as vectors based on how close their meanings are. Elements of such embedding space are often referred to as *tokens*.[1]

Second, from the perspective of DNLMs, a suite of such tokens is nothing more than a trajectory—a line—in that continuous *embedding space*. It does not select words deterministically and, depending on the model type, might not even look further ahead to ensure that what it is generating can have a continuation that would make sense.

Third, such trajectories are not deterministic but rather probabilistic distributions, indicating how frequently trajectories combining a suite of similar tokens in similar order have been encountered in the training data.

While this representation is highly counterintuitive, it is not exactly new. Succeeding to the probabilistic view of Natural Language texts introduced by Claude Shannon in 1948 [5], it was introduced in the early 1990s by [6] and [7]. Shortly after that, it was popularized by [8] and shown to outperform existing state-of-the-art methods in tasks such as machine translation [9], as long as it was provided sufficient training data, which at the time was made possible by Google Web crawls and digitization of existing translations, such as EU Council parallel texts. However, this model was not only suited for translation. Many NLP tasks could be represented as sequence-to-sequence translation [10], including text generation.

However, this approach had a major problem—learning and representing the trajectories in the "embedding space." Whereas rule-based chatbots could have a combination of pattern matching and response "else-if" rules, LLMs learned by themselves from massive amounts of data and in high dimensions. A first breakthrough was achieved by using RNNs [11], and a proper text generation in a simple entailment context[2] has been shown to be possible by [12], after the introduction of an improved algorithm to train RNNs.

Despite their great initial performance, RNN-based architectures suffered from two major drawbacks. First, their training was inherently sequential. The principle of RNNs consists of reusing a processing cell (neuron) output as its own input. This makes these neural networks recurrent, but it also means that processing cells cannot start processing the next item in a sequence before it is done with the previous one. Second, the length of sequences RNNs is practically limited due to the information

[1] The optimal way to convert a text to tokens and back is still an active research subject. Current LLMs seem to favor varying length tokens, using roots for words and common suffixes/prefixes in English, single characters for digits and abbreviations, and a combination of the two for more rare words and other languages. I will not be reviewing the subject here and use tokens and words interchangeably.

[2] An example would be "Complete the suite: "Dog, cat, cow, …" with "goat," "sheep," or other domestic animal being an acceptable answer.

about the previously seen sequence contents being passed as a recurrent output that eventually vanishes, as well as the fact that every single token they have seen had to be accounted for in the learning and generation process. Not only did it make them unable to pick up any additional context outside that length window, but in the generative mode, it meant they would often generate a distribution of tokens they never saw in their training data and, in the absence of learned trajectories distribution that would continue such a sequence, fail to generate any meaningful continuation [13].

This latter problem was addressed by the *self-attention* mechanism, introduced by [14]. The idea was to leverage the fact that for longer token lengths, the space of trajectories effectively encountered in the linear space is sparse, meaning that instead of having to take into account all of the preceding tokens to perform inference, RNN models could be trained only on a smaller set of "important" tokens, that would be selected by a separate "attention" neural network. Widely successful, this mechanism was rapidly adopted for the Google Translate engine [15] and is still in use there to the best of our knowledge. The interesting property of that "attention" neural network is that it was not recurrent. Computing an attention vector for one sequence was unnecessary before computing it on the other, meaning it could be efficiently parallelized.

The now-seminal "Attention is all you need" by [16] demonstrated that by increasing the overall size of the model and adopting a multi-layer, multi-head pure self-attention architecture with normalization between layers (i.e., the Transformer itself), RNN processing units could be removed from the architecture entirely. Without recurrent elements, the network did not need to wait anymore to process prior tokens to obtain the recurrent input for the next token but instead could be trained synchronously. While it might not seem as much, it is hard to understate how transformative it was. Model training can now be fully parallelized across as many computation nodes as available, leading to an arms race in model size, training dataset size, and the amount of computational power invested into training them. The resulting model size exponential growth, represented in Fig. 1.1, is still ongoing, and was halted by some recent models only by a transition to smaller, more practical, *compute-optimal models*.[3]

One of the interesting features of the Transformer is that due to its intended use as a neural translation engine, it contains two mostly independent parts. First, the encoder, whose role is to parse the input text and produce an *encoding space* representation of it. Second, the decoder receiving that representation will generate the corresponding translation one word at a time while looking at previously generated words to make sure the next one is still coherent with them (Fig. 1.2).

As such, models that are not translation-specific can use only the decoder part if they are generating text based on only the previous token and only the encoder part if they do not necessarily seek to generate text but rather understand the structure of

[3] Due to entirely different scaling rules, I have not included pure Mixture-of-Experts models in this Figure.

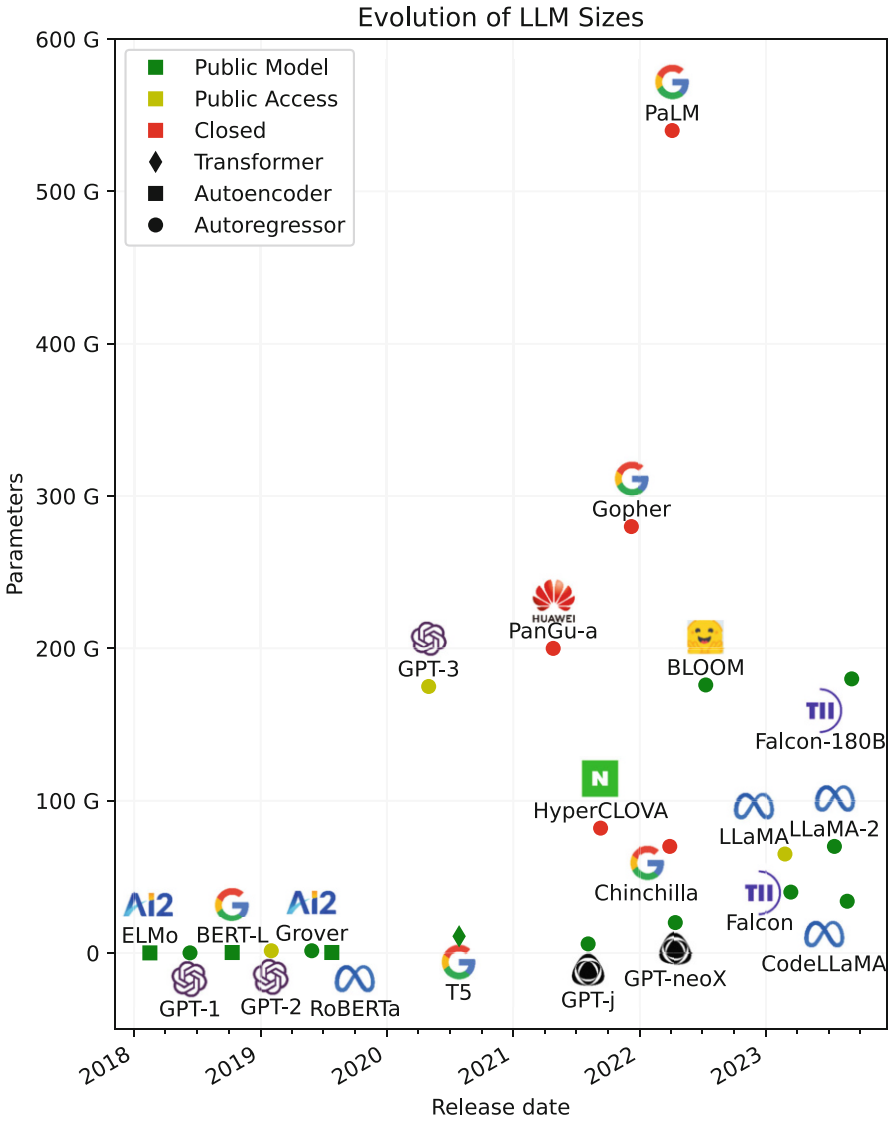

Fig. 1.1 Evolution of size, type, and availability of common LLMs with published architecture. For that reason, ChatGPT, GPT-4, and several common LLMs were omitted

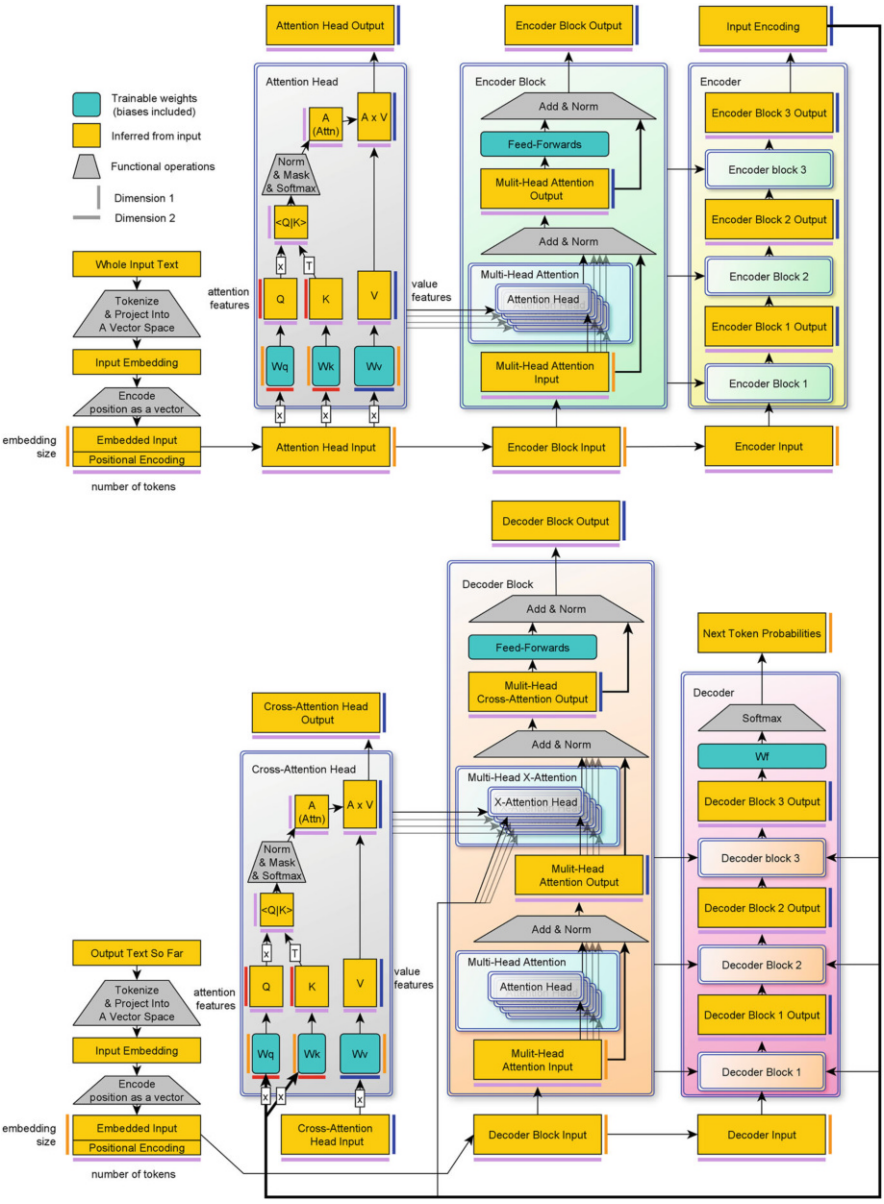

Fig. 1.2 Encoder-decoder transformer architecture based on [16]

the text. Because of that, purely generative text models tend to use only the decoder part. In contrast, models destined for a more general text understanding tend to use the encoder, which offers some major advantages in exchange for drawbacks that have remained until recently limited.

1.2.2 Generative Deep Neural Language Models

From the point of view of a Transformer-based LLM, generating new text is equivalent to generating a translation, except that there is not necessarily an embedding space representation. Instead, it is the continuation of an initial word sequence, where the word to be generated can be located either at the end of the original sentence or in the middle of it. These two cases correspond to the two main approaches to training Generative Deep Neural Language Models.

Autoregressive models are trained by a random sampling suite of words in the source dataset ("utterances"), masking a word and training the model to predict the masked word accurately [17]. Autoregressive models are best thought of as autocomplete—they are trained to complete sentences in a way that is representative of their training dataset.

Autoencoding models are trained similarly, except that the masked word can be located anywhere in the randomly sampled suite of words [18]. Autoencoding models are best thought of as search or autocorrect suggestions—they are trained to rewrite whole sentences in a way that is representative of their training dataset. Just as Google search suggestions, they can suggest words at the end of a query to refine it, but they can also add one at the beginning or even rewrite a word (for instance, to correct a typo). While Autoencoding models can be used to generate text based on utterances, their strength is rather in applications that require understanding the utterance as a whole.

While the autoencoding models are generally considered more powerful than autoregressive ones, their generative capabilities are not necessarily optimal for the model's size, training dataset, or the computational resources spent training the model. After all, the training mode relevant to the generation represents only a fraction of the training time of autoregressive models.

There are several paradigms of how generative models can be trained. However, only one is currently dominant and is referred to either as "generative pre-training" or "teacher forcing." Specifically, the model is provided with a large set of utterances with the last word masked and is asked to predict that last token. Based on the proximity of predicted tokens to the tokens in the training dataset, a loss is calculated, and the model is trained through backpropagation.[4] Each set of such

[4] Backpropagation is an algorithm used to train artificial neural networks. The goal of backpropagation is to adjust the weights of the neurons in the network, with the final purpose of minimizing the error between the predicted and actual outputs.

utterances is commonly called a "batch." In some cases, a refinement process is possible late in the training process when the model is allowed to predict more and more tokens to stabilize the generation of longer texts [13].

1.2.3 Generating Text

Once trained, LLMs can then be used to generate new text. For that, they need a starting text called "prompt," for which they will look for the word that would most likely continue that prompt in their training dataset. However, as I explained above, models do not learn specific words but rather the probabilities of related tokens. Hence, when they are used to generate texts for every word, they can only provide the probabilities of all words in their vocabulary. Hence, to generate a single next word, they need a "sampling strategy" to choose that single word.

Four sampling strategies are most used: *maximum likelihood*, *beaming*, *top-K*, and *top-p/nucleus*. The latter is considered to be *State-of-the-Art* (SotA) and has been introduced by [19], which also reviews other sampling strategies.

Maximum Likelihood—also known as *temp 0 sampling* always picks the most probable next word. While it can be a good idea for short texts with long prompts, the model is likely to end up in a state where the chosen chain of words has no continuation it would know of, and it would start generating nonsense. This is known as "output degeneration" [13, 19]. Beaming allows us to mitigate some of those issues by looking at the cumulative probability of the next X tokens and choosing the word that maximizes the probability of having a high probability branch.

However, in both cases, the same prompt will generate a similar, if not the same, response, which is usually not what is wanted. For instance, getting always told the same tale in response to a "Tell a tale starting with *Once upon a time*" would be boring, especially since the model can do better. That is specifically the problem that the top-K sampling is meant to solve. Rather than sampling deterministically, it randomly samples one of the top K most probable tokens. While it solves the repetitiveness problem, it creates a new one—in a setting where a unique suite of words is the only answer, it is susceptible to pick one of the other K continuation words.

Finally, top-p, or temperature-based sampling, tries to combine the best of the top worlds by sampling randomly out of the tokens with the highest probability, such that their cumulative probability stays above p. In this case, a token that almost surely needs to be generated will be alone in the random sampling pool. In contrast, in the cases where many different continuations are acceptable, one will be chosen randomly.

Once the next token has been generated with one of the sampling strategies, it is added to the original prompt, and that combined text becomes a prompt for the generation of the next token.

1.2.4 Memorization vs Generalization

LLMs learn the distribution of token sequences in the model embedding space based on the data present in the training set and generate utterances based on a sampling strategy and a prompt. This means they are consistently on edge between citing the elements of the training dataset they have memorized if the prompt is sufficiently precise and unique and composing a new, never-seen continuation to the prompts. The former is referred to as the "memorization" ability of the model, whereas the latter is "generalization." Historically, "memorization" of the models has been thought of as overfitting the training data and easily avoided by exposing the model to the same training as little as possible.

However, results that followed shortly after the first GPT (*Generative Pre-trained Transformer*) models release—notably [20] have shown that LLMs can memorize things they have seen in the dataset only once, even if a specifically designed prompt needs to be found to trigger the full recall. In this way, authors of [20] could retrieve valid SSIDs, telephone numbers, and emails from the training dataset of the GPT-2 model. Perhaps even more impressively, the GPT-2 model authors used could recite 834 digits of Pi, which it has encountered in the source dataset only once.

However, today, no known rules or approaches exist to improve or discourage memorization of the models. The research into strategies to understand what private information the model has memorized and how it can be retrieved is an active field. It has historically been referred to as "Model Red Teaming" [21], although today the term is used for general work of characterization of model failure modes. Conversely, active research is ongoing to understand why the model generalizes when memorization is desired—notably for counterfactual statements, commonly called "hallucinations."[5]

With this research still ongoing, as a rule of thumb, currently, no data used to train an LLM can be considered safe, and no text generated by an LLM can be assumed to be factually correct or as not containing memorized information.

1.2.5 Effect of the Model and Training Dataset Size

The dramatic growth in the models' size between 2019 and 2022, illustrated in Fig. 1.1, has been driven by almost perfect predictability of the generative models'

[5] Although several researchers prefer the term "confabulation" as closer to the mechanism.

performance increase. As long as it is provided with a sufficient amount of data and computational resources to train on that data, a larger model is going to keep improving its ability to predict the next word in a text—across a variety of contexts represented in the training dataset [3, 4, 22]. Correspondingly, that translates to a better ability of a pre-trained LLM to understand a variety of contexts, generate higher quality, more nuanced, and longer texts, and remember more context present in the prior text it is encountering.

While the predictive base model performance is an interesting ability, after exceeding a certain size, even general models start "unlocking" new capabilities. For instance, between 10 and 100B parameters, LLMs not trained for the task start being able—to some degree—to generate, compile, and run computer code, translate between languages, or emit predictions of human behavior. While they are less data, compute, and parameter-efficient than specialized models, they have been claimed to outperform them, making LLMs particularly interesting as general-purpose models that, with few-shot transfer learning, can make specialized models redundant [4]. Such abilities are commonly referred to as emerging capabilities. While metrics of performance on many such specialized tasks improve linearly with the model's size and the training dataset, they are often not noticeable on smaller models, meaning that the capabilities of still larger models remain to be seen.

Overall, the emergent abilities are generally believed to be made possible—parameter-wise through a combination of a bigger attention span of the model, allowing it to take into account more context, a larger "hidden state," allowing it to encode more different context-continuation matches, and finally, more parallel layers that allow learning more different ways to map texts to "hidden states." Conversely, larger unstructured datasets are more likely to contain niche reasoning utterances relevant to the problem, such as explaining code or interest in chess moves in different contexts.

However, the existence of the emerging capabilities and their usefulness is still a contested topic. Re-analysis of claimed emergent capabilities led some researchers to suggest that they are artifacts of the choice of performance metrics rather than representative or inherent model capabilities [23].

What is less of a contested topic is the ability of larger LLMs to unlock not only emerging capabilities but also emerging failure modes. Right around the scale of size and training data where LLMs learn to program and play chess games, they also acquire bias based on sex, gender, race, religion, and a propensity to insert unprompted slurs and toxic discourse into their output [4]. This is not entirely surprising. Large unstructured datasets of texts from the Internet might or might not contain better chess moves with rationale. However, they will surely contain more slurs, toxicity, and fringe extremist content banned from mainstream media and social networks.

With both the expected output quality improvement and the emergent abilities requiring larger models, more data, and more compute, and the Transformer architecture allowing for efficient parallelization of the training, the race to the best

model through data and compute accumulation kicked off (Chap. 3). However, given the amount of computing committed to the largest model, it made sense to check if there was an optimal ratio between the model size and the amount of training data fed to it.

Such a trade-off is known as *scaling laws*—an optimal relationship between the model size, dataset size, and computational power investment required to achieve the best model performance while minimizing computational expenses. There are currently two schools of thought on what this scaling law is.

Historically, as one of the first teams to venture into the 1B+ models domain, OpenAI came up with a scaling law that is approximately 1.7 token/model parameter, and computational power requirement multiplied by 4 with every model doubling [3]. OpenAI claims that GPT-3 and GPT-4 were both trained according to this scaling law.

More recently, the Google DeepMind team found a more conservative scaling law of approximately 10 tokens/parameter with computational power requirement similarly multiplied by 4 with every model size doubling [22]. Google used this new scaling law to train Chinchilla—a 70B model that outperformed Gopher, a 280B parameter model trained according to the classical scaling law. Given the computational price to train LLMs and infer on them, almost all LLMs released since have been trained according to this law—including LLaMA, Falcon, and others- are generally referred to as *compute-optimal*.

While there is still debate as to which law is preferable, the size of compute-optimal models track closely the human perception of their performance and performance on benchmarks [24]. When considering the training dataset size rather than the model size alone, a completely different picture emerges by trimming the declared model size to a compute-optimal one with the same size as the training dataset. Figure 1.3 shows the progress of the training dataset size for notable released models, suggesting an explanation as to why some smaller models are known to perform particularly well compared to the models of similar or even larger size (RoBERTa, T5, GPT-j, GPT-neo(X), GPT3). In Fig. 1.4, I attempted to renormalize notable models to the training dataset size if they were trained according to the optimal resource utilization rule. While this renormalization dismisses any considerations regarding the dataset's quality, it allows for an intuitive model comparison.

Unfortunately, there are few replication or ablation studies due to the extreme requirements of the computational resources needed to train LLMs. Our understanding of LLM performance scaling and emergent possibilities are still evolving and will likely change as more affordable hardware allows more researchers and practitioners to train and re-train LLMs.

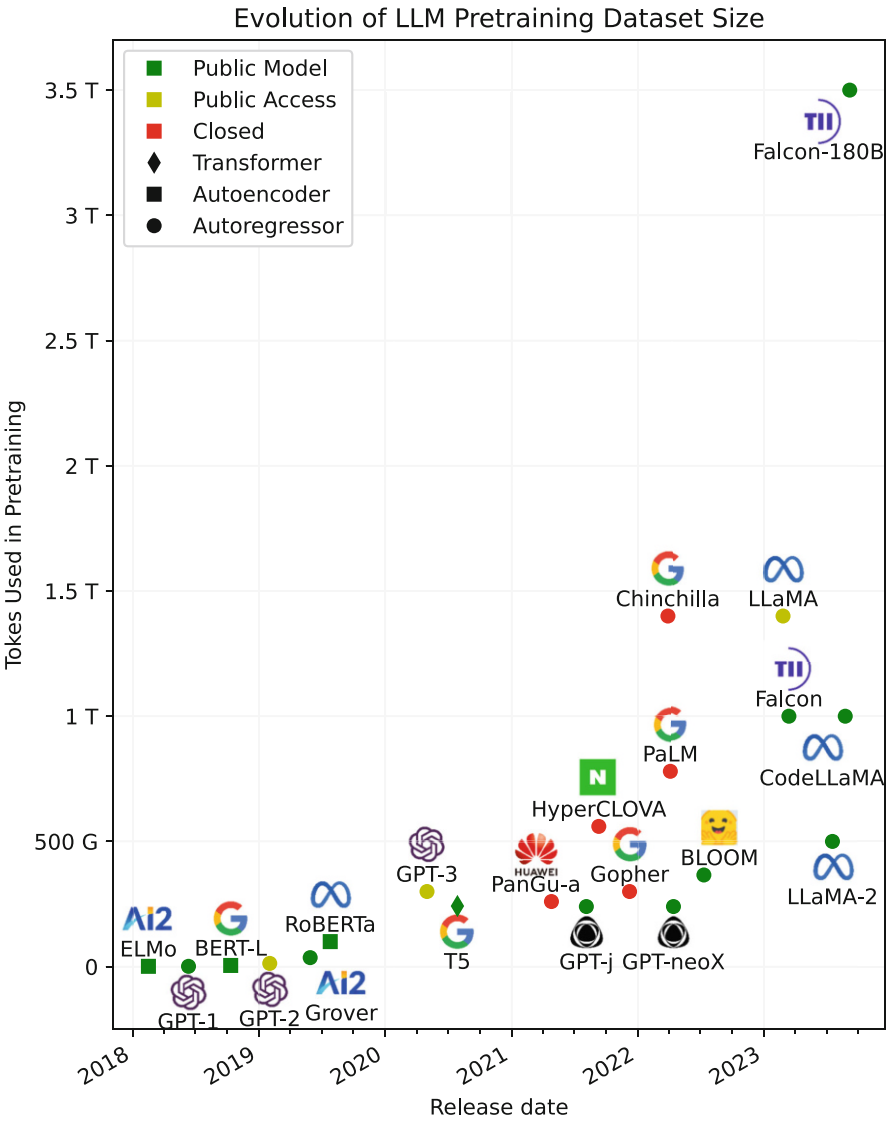

Fig. 1.3 Progression of the pre-training dataset size for common LLMs over time. The pre-training dataset size in tokens has been taken from the model announcement whenever available and otherwise estimated at 0.3 tokens/byte, based on the results presented by [25]

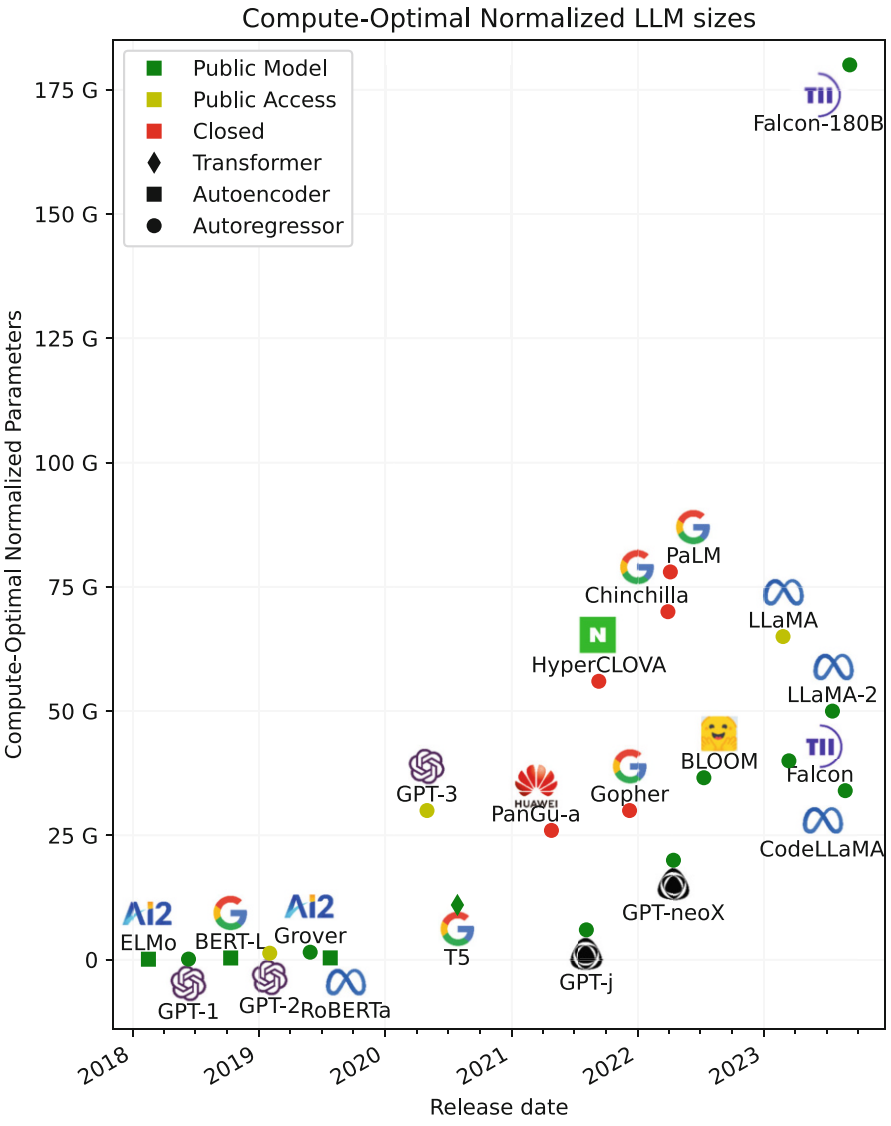

Fig. 1.4 Progression of compute-optimal equivalent LLMs with published architecture and training dataset sizes. ChatGPT, GPT-4, and several other common LLMs are excluded for that reason

References

1. Emily M. Bender, Timnit Gebru, Angelina McMillan-Major, and Shmargaret Shmitchell. On the dangers of stochastic parrots: Can language models be too big? In Madeleine Clare Elish, William Isaac, and Richard S. Zemel, editors, *FAccT '21: 2021 ACM Conference on Fairness, Accountability, and Transparency, Virtual Event / Toronto, Canada, March 3–10, 2021*, pages 610–623. ACM, 2021.
2. Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. Distilbert, a distilled version of BERT: smaller, faster, cheaper and lighter. *CoRR*, abs/1910.01108, 2019.
3. Jared Kaplan et al. Scaling laws for neural language models. *CoRR*, abs/2001.08361, 2020.
4. Deep Ganguli et al. Predictability and surprise in large generative models. In *FAccT '22: 2022 ACM Conference on Fairness, Accountability, and Transparency, Seoul, Republic of Korea, June 21 - 24, 2022*, pages 1747–1764. ACM, 2022.
5. Claude E Shannon. A mathematical theory of communication. *The Bell system technical journal*, 27(3):379–423, 1948.
6. Risto Miikkulainen and Michael G. Dyer. Natural language processing with modular PDP networks and distributed lexicon. *Cogn. Sci.*, 15(3):343–399, 1991.
7. Jürgen Schmidhuber and Stefan Heil. Sequential neural text compression. *IEEE Trans. Neural Networks*, 7(1):142–146, 1996.
8. Yoshua Bengio, Réjean Ducharme, and Pascal Vincent. A neural probabilistic language model. In Todd K. Leen, Thomas G. Dietterich, and Volker Tresp, editors, *Advances in Neural Information Processing Systems 13, Papers from Neural Information Processing Systems (NIPS) 2000, Denver, CO, USA*, pages 932–938. MIT Press, 2000.
9. Holger Schwenk, Daniel Déchelotte, and Jean-Luc Gauvain. Continuous space language models for statistical machine translation. In Nicoletta Calzolari, Claire Cardie, and Pierre Isabelle, editors, *ACL 2006, 21st International Conference on Computational Linguistics and 44th Annual Meeting of the Association for Computational Linguistics, Proceedings of the Conference, Sydney, Australia, 17–21 July 2006*. The Association for Computer Linguistics, 2006.
10. Ronan Collobert and Jason Weston. A unified architecture for natural language processing: deep neural networks with multitask learning. In William W. Cohen, Andrew McCallum, and Sam T. Roweis, editors, *Machine Learning, Proceedings of the Twenty-Fifth International Conference (ICML 2008), Helsinki, Finland, June 5–9, 2008*, volume 307 of *ACM International Conference Proceeding Series*, pages 160–167. ACM, 2008.
11. Tomas Mikolov, Martin Karafiát, Lukas Burget, Jan Cernockỳ, and Sanjeev Khudanpur. Recurrent neural network based language model. In *Interspeech*, volume 2, pages 1045–1048. Makuhari, 2010.
12. Ilya Sutskever, James Martens, and Geoffrey E. Hinton. Generating text with recurrent neural networks. In Lise Getoor and Tobias Scheffer, editors, *Proceedings of the 28th International Conference on Machine Learning, ICML 2011, Bellevue, Washington, USA, June 28 - July 2, 2011*, pages 1017–1024. Omnipress, 2011.
13. Ferenc Huszar. How (not) to train your generative model: Scheduled sampling, likelihood, adversary? *CoRR*, abs/1511.05101, 2015.
14. Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. In Yoshua Bengio and Yann LeCun, editors, *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7–9, 2015, Conference Track Proceedings*, 2015.
15. Melvin Johnson et al. Google's multilingual neural machine translation system: Enabling zero-shot translation. *Trans. Assoc. Comput. Linguistics*, 5:339–351, 2017.
16. Ashish Vaswani et al. Attention is all you need. In Isabelle Guyon et al., editor, *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4–9, 2017, Long Beach, CA, USA*, pages 5998–6008, 2017.

17. Dumitru Erhan et al. Why does unsupervised pre-training help deep learning? *J. Mach. Learn. Res.*, 11:625–660, 2010.
18. Samuel R. Bowman et al. Generating sentences from a continuous space. In Yoav Goldberg and Stefan Riezler, editors, *Proceedings of the 20th SIGNLL Conference on Computational Natural Language Learning, CoNLL 2016, Berlin, Germany, August 11–12, 2016*, pages 10–21. ACL, 2016.
19. Ari Holtzman, Jan Buys, Li Du, Maxwell Forbes, and Yejin Choi. The curious case of neural text degeneration. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26–30, 2020*. OpenReview.net, 2020.
20. Nicholas Carlini et al. Extracting training data from large language models. In Michael Bailey and Rachel Greenstadt, editors, *30th USENIX Security Symposium, USENIX Security 2021, August 11–13, 2021*, pages 2633–2650. USENIX Association, 2021.
21. Ethan Perez et al. Red teaming language models with language models. *CoRR*, abs/2202.03286, 2022.
22. Jordan Hoffmann et al. Training compute-optimal large language models. *CoRR*, abs/2203.15556, 2022.
23. Rylan Schaeffer, Brando Miranda, and Sanmi Koyejo. Are emergent abilities of large language models a mirage? *CoRR*, abs/2304.15004, 2023.
24. Aakanksha Chowdhery et al. Palm: Scaling language modeling with pathways. *CoRR*, abs/2204.02311, 2022.
25. Teven Le Scao et al. BLOOM: A 176b-parameter open-access multilingual language model. *CoRR*, abs/2211.05100, 2022.