

1.1 GPT 系列

2018 年 OpenAI 提出了 GPT-1 [1](Generative Pre-Training), GPT-1 跟 BERT 类似, 也是先用大量无标注的数据做无监督学习得到 pre-trained 模型, 然后用 task-specific 的标注数据来做 fine-tuning 这个 pre-trained 模型从而完成特定任务, 做 fine-tuning 的方法就是在最后加个线性层, 然后只更新这个线性层和 embedding 层的参数。GPT-1 的模型架构如图1.1, 他采用堆叠 12 层 Transformer decoder, 注意没有 multi-head attention 部分。GPT-1 的输入也跟 BERT 类似, 也是一个 sequence 中的每个 token, 每个 token 的 word embedding 与 positional embedding 相加, 然后依次经过多层 Transformer-Block。GPT-1 与 BERT 其他的

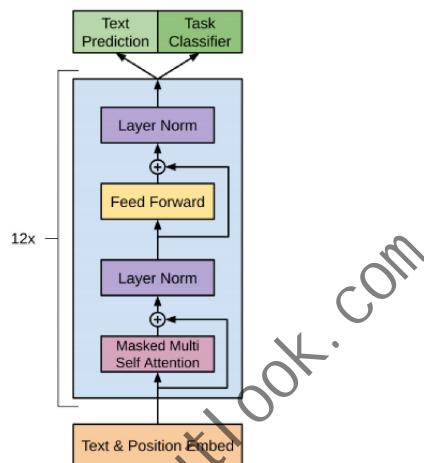


图 1.1: GPT^① 架构示意图

共同点为:

- 位置编码都是自动学习的参数;
- 激活函数都是采用 GELU。

他们的区别为:

- BERT 采用了 Masked Language Model 和 Next Sentence Prediction 两个任务来训练模型, GPT-1 采用自回归的方式训练, 也就是根据前面的多个 token 预测下一个 token。这样就导致双方采用不同的架构: BERT 采用了 Transformer encoder, 而 GPT 则采用了 Transformer decoder。
- BERT 是双向模型, 它可以从上下文中获取丰富的信息, 具有较强的上下文理解能力;而 GPT 是单向模型, 它只能依赖已生成的上文来预测下一个词语。BERT 在词语级别的任务中表现出色, 如命名实体识别、问答等; GPT 在生成式任务中表现较好, 如对话生成、文本生成等。
- BERT 采用了 WordPiece 做 tokenization, GPT-1 采用 BPE 做 tokenization。

GPT-1 模型并没有引起人们的关注，反倒是谷歌随即提出的 BERT 模型 [2] 产生了更大的轰动。不过，OpenAI 继续沿着 GPT-1 的技术思路，在 2019 年发布了 GPT-2 [3]。GPT-2 试图创建一个无需做 supervised fine-tuning 就能完成下游任务的模型，也就不对模型做任何参数或者结构上的改变就能完成很多下游任务。而到目前 Multitask learning 是训练通用语言模型很有前景的方法。于是 GPT-2 将不同 task 的输入规整成同一个形式，比如：

- 翻译任务中，一个样本被写成 (translate to french, \$(english text), \$(french text));
- 阅读理解任务中，一个样本被写成 (answer the question, \$(document), \$(question), \$(answer))

，然后统一用相同的 objective(就是 autoregressive language model) 来训练模型。用相同格式的多种任务的训练样本同时训练一个模型，使得该模型获得同时执行多种任务的能力。不过这样的训练存在的重大挑战就是收敛很慢甚至可能不收敛。openAI 团队为训练 GPT-2，首先创建了一个高质量多样化且大小为 40G 的数据集：WebText。GPT-2 的模型架构几乎与 GPT-1 一样，只有下面几个修改：

- 每层 decoder 中，两个 Layer normalization 分别移动到 Masked Multi Self Attention 之前和 Feed Forward 之前。最后一层 decoder 后还加一个 Layer normalization。因为修改了网络结构，初始化也跟着稍微调整了。
- GPT-2 可以处理最长 1024 的 token 序列。另外，GPT-2 的 embedding size 是 1600。

GPT-2 有 1.5B 个参数 (48 层)，网络架构如图1.2。相比 GPT-1，GPT-2 的参数数量和训练

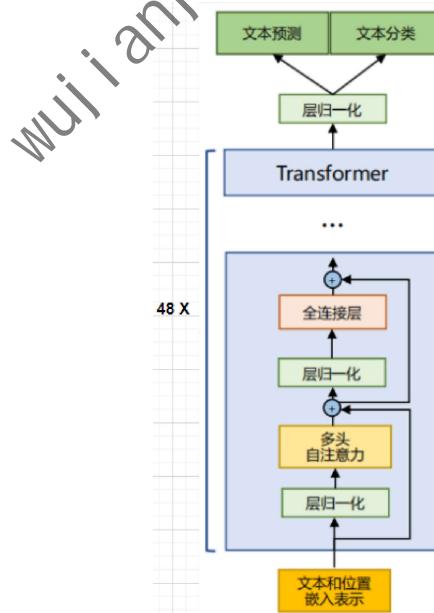


图 1.2: GPT-2 架构示意图

数据集的大小均增加了十倍。GPT-2 的最大贡献是验证了通过海量数据和大量参数训练出来的模型有迁移到其它任务中而不需要额外的训练。

GPT-3 [4] 依旧延续 GPT-2 的训练方式，只不过把模型的参数量增大到了 175B(96 层 decoder, 12288 的 embedding size)，并且使用 570GB 的高质量数据 (原始 45TB, 过滤清洗后剩下 570GB) 进行训练。GPT-3 提出“提示语”(Prompt) 的概念，只要提供具体任务的提示语，即便不对模型进行调整也可完成该任务，如：输入“我太喜欢 ChatGPT 了，这句话的情感是？”，那么 GPT-3 就能够直接输出结果“褒义”。如果在输入中再给一个或几个示例，那么任务完成的效果会更好，这也被称为语境学习 (in-context Learning)。GPT-3 的模型架构与 GPT-2 基本一致，只有下面几处不同：

- 在 decoder layer 之间交替使用原始的 attention(Dense) 和带状的 attention(Sparse)。
- 可以处理的序列长度从 1024 个 token 变成 2048。

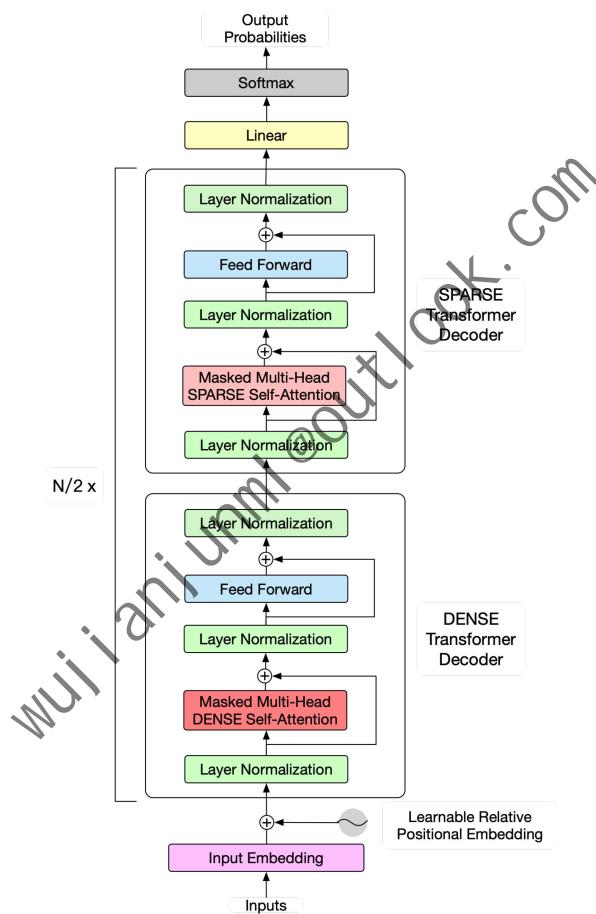


图 1.3: GPT-3 架构示意图

GPT-3 无需进行任何梯度计算或者参数更新，仅仅输入 zero-shot/one-shot/few-shot 的文本就可以完成任务，在许多任务上表现卓越，甚至能进行算术运算，让人们看到了通用人工智能的希望。同时需要指出，在大多数任务上，它离人类水平还很远，在部分任务上，它的表现都无法超过最佳的微调模型。另外，与英文不同，中文需要进行分词，而英文就是天然单词；其次中文的歧义性很强，比如说“喜欢上/一个人”，“喜欢 / 上一个/人”，“喜欢上/一个/人”，这些都表达了不同的意思；所以中文版 GPT-3 的研发充满更大挑战。

语言模型存在的捏造事实，有偏见，甚至不听从人类指令的情况，导致这些问题的原因在于训练他们的时候仅仅用的是自回归的目标函数，没有要求模型遵循用户指令。InstructGPT [5] 的模型架构跟 GPT-3 一样，但是使用 reinforcement learning from human feedback(RLHF) 来 fine tune GPT-3。InstructGPT 的训练过程可以概括为：

- 首先从 GPT-3 的 API 日志中采集一批用户指令/prompt(input)，并雇佣一批人来编写每个 prompt 的回答 (output)，用这些 (input, output) 对儿 (约 1.3 万条) 来 supervised fine-tune (SFT) GPT-3 模型得到一个 SFT 模型。
- 接着利用 SFT 模型对同一个 prompt 输出多个 output，然后让人工对这多个 output 进行从最好到最差的排序，然后利用这些排序好的数据 (约 3.3 万条) 训练一个 reward model (RM)。这个 RM 就可以对 LLM 的输出进行打分。RM 模型架构与 GPT-3 一样，这个 RM 模型的 embedding 层是 copy 自一个 pretrained GPT-3 模型的，RM 模型的参数量为 6B。每个 prompt x 对应多个 completion(就是输出)，取这多个 completion 的两两组合，每个组合中 y_w 比 y_l 更好，训练 RM 的 loss 函数如下：

$$loss = -E[\log \sigma(r_\theta(x, y_w) - r_\theta(x, y_l))]$$

其中 r 表示 RM 的输出，也就是最大化 $r_\theta(x, y_w)$ 与 $r_\theta(x, y_l)$ 的差值。

- 以 Reward Model 作为奖励函数，以 baseline 模型为初始 policy，采用 PPO [6] 改善 policy 的表现，以求每个 output 的 reward 最大，得到强化版本的模型 PPO-ptx，这个版本也就是所谓的 InstructGPT。训练 RL 时，state 是 Prompt 和已经生成的 output 序列，动作是下一个 token。Reward 在中间步骤都是 0，只有在最后 output 结束输出 EOS 才有奖励，折扣因子为 1。奖励中增加了一个 KL penalty，以令所学到的 policy 与初始 policy 相差不至太远。这一步的训练时最大化下面的目标函数如下：

$$\begin{aligned} objective = & E[\\ & r_\theta(x, y) \\ & -\beta \log \frac{\pi^{RL}(y|x)}{\pi^{SFT}(y|x)} \quad \text{KL penalty} \\ &] + \gamma E_{x \sim D_{pretrain}} [\log \pi^{RL}(x)] \quad \text{pretraining gradients} \end{aligned} \tag{1.1}$$

其中 $\pi^{RL}(y|x)$ 是学习中的 policy， $\pi^{SFT}(y|x)$ 是 SFT 模型。 $E[\log \pi^{RL}(x)]$ 意思是从 GPT-3 的预训练文本中采样一些文本 x ，希望模型尽量增大生成文本 x 的概率。

要更进一步理解 InstructGPT，我们就需要深入理解 RLHF [7] 和 PPO [6]。微调训练过程为：将 prompt x 输入初始 LLM 和当前微调的 LLM，分别得到了输出文本 y_{SFT}, y_{RL} ，将 y_{RL} 传递给 RM 得到 reward $r(x, y_{RL})$ 。将两个模型的生成文本 y_{SFT}, y_{RL} 进行比较计算 KL penalty。最后根据 PPO 算法，我们按当前 batch 的 *objective* 进行优化。

ChatGPT 和 InstructGPT 在模型结构，训练方式上都完全一致，即都使用了指示学习 Instruction Learning 和人类反馈的强化学习 Reinforcement Learning from Human Feedback 来指导模型的训练，它们不同的仅仅是采集数据的方式上有所差异。

wujianjunml@outlook.com

1.2 LLaMA 系列

LLama 1 [8] 使用了 CommonCrawl, C4 等公开数据集，并采用 BPE 作为 Tokenizer，最后整个训练数据 1.4 万亿个 token。LLaMA 的模型架构要点如下：

- LLaMA 采用了 Pre-normalization 和 RMSNorm 来改善训练的稳定性。RMSNorm 是基于 LN 的一种变体，主要是去掉了减均值的部分。LayerNorm 原始的计算公式为：

$$\mu = \frac{1}{n} \sum_{i=1}^n a_i \quad \sigma = \sqrt{\frac{1}{n} \sum_{i=1}^n (a_i - \mu)^2} \quad \bar{a}_i = \frac{a_i - \mu}{\sigma} g_i + b_i \quad (1.2)$$

RMSNorm 的计算公式如下：

$$RMS(\mathbf{a}) = \sqrt{\frac{1}{n} \sum_i^n a_i^2} \quad \bar{a}_i = \frac{a_i}{RMS(\mathbf{a})} g_i + b_i \quad (1.3)$$

- 用 SwiGLU [9] 替换 Relu。

$$Swish_\beta(x) = x\sigma(\beta x) = \frac{x}{1 + e^{-\beta x}}$$

， 在他们的实验中， β 取为 1。

- 用 rotary positional embeddings (RoPE) [10] 替换 absolute positional embedding。

为了加速模型的训练

- LLaMA 使用 causal multi-head attention，就是不计算也不存储被 mask 掉的 K 和 Q (式??)，这个可以大大减少显存的占用并且加快运算。注意，要高效地实现这个过程并不是那么容易，所以这里直接使用的是 Xformer 库的实现。
- LLaMA 缓存一些 layer 的输出 (比如 FC 层)，避免后向传播时重复计算。注意，要实现这一点就不能使用 pytorch 的自动微分功能了，要自己实现后向传播过程。

除此之外，pipeline 并行和 GPipe 等类似的技术也被用于分布式训练。LLaMA 用 AdamW 优化器来和 cosine 学习率调度算法来训练模型。LLaMA-13B 在很多任务上的性能超过 GPT-3 170B，而且可以运行在单张 GPU 上。

LLama 2 [11] 的模型架构跟 LLama 1 大体一样，只有几处小的差异：

- context length 从 2048 扩大到 4096。
- 使用 grouped-query attention(GQA) 来缓解 KV Cache 的存储压力。

LLama 2 70B 的效果堪比 GPT-3.5。LLama 2-Chat 是对 LLama 2 做指令微调 (SFT,Supervised Fine-Tuning SFT) 和强化学习微调 (RLHF,Reinforcement Learning with Human Feedback) 后得到的模型。在做 RLHF 时候有几个值得注意的点：

- 训练了两个 reward model: Helpfulness RM 和 Safety RM。
- 使用了两种强化学习算法: Proximal Policy Optimization (PPO) 和 Rejection Sampling fine-tuning。

wujianjunml@outlook.com

1.3 GLM 系列

GLM(General Language Model) [12] 的预训练目标为 autoregressive blank infilling。对于一个文本序列 $\mathbf{x} = [x_1, \dots, x_m]$, 随机把其中一个或则多个子序列 (span) 用 MASK 这个特殊的 token 替换掉, 用 $x_{corrupt}$ 表示替换后的序列, 让模型去预测出被替换的子序列。对于一个句子 $x_1, x_2, x_3, x_4, x_5, x_6$, 我们采样得到两个 span(子序列): x_3 和 x_5, x_6 , 接着得到两个 part: part A = $x_{corrupt} = x_1, x_2, [M], x_4, [M]$ 和 part B = $[S], x_5, x_6, [S], x_3$ 。注意, 对采样得到的多个 span 做了下随机排列。part A 的 token 相互之间都可以看见, 但是不能看见 part B 的任何 token。part B 的任何 token 看得见 part A 的全部 token 以及 part B 中位于本 token 之前的 token。见图1.4所示的 attention mask 矩阵。可以看出: part A 是双向的, part B 则

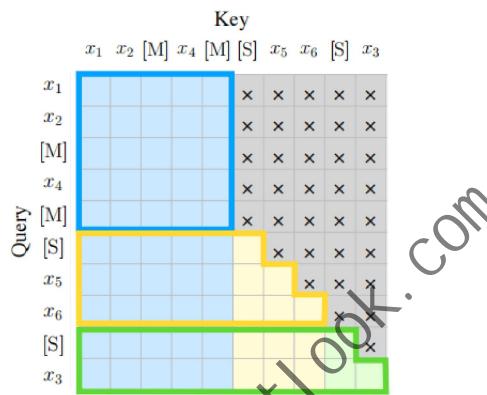


图 1.4: GLM Self-attention mask

是单向的。我们向 encoder 输入 part A(做双向理解), 然后期望 decoder 逐个 token 地输出 part B 中全部 token, decoder 每次会看到 part A 的全部 token。为能做 text generation, 除了 blank infilling, GLM 还做了 Multi-Task Pretraining, 区别在于后者针对文档输入, mask 掉的 span 长度更长, 且 mask 掉的 span 是一句完整的句子。在模型架构上, GLM 与原始的 Transformer 一样, 不过使 GeLU 作为 FFN 的激活函数。GLM 还使用了 2D Positional Encoding, 因为 GLM-130B 没有沿用, 所以我们这里不做介绍。

GLM-130B [13] 其表现性能上与 GPT3 不相伯仲。为了同时支持理解和生成, GLM-130B 使用两种不同的掩码标识符 ([MASK] 和 [gMASK]), 分别用于短文和长文的生成。每个句子的 $x_{corrupt}$ 是下面两种之一:

- **MASK:** 30% 的句子用空白符替换一个连续子序列, 子序列的长度 (也是空白符的个数) 服从 Possion 分布并加上句子长度的 15%。是句子中的短空白。
- **gMASK:** 70% 的句子, 前缀保持不变, 把句子的尾巴用空白符替换, 尾巴的长度服从均匀随机分布。是句尾随机长度的长空白。

另外, GLM-130B 使用 74 个指令数据集 (prompted instruction datasets) 来做 Multi-Task Instruction Pre-Training(MIP), 这些数据集的 token 占整个训练集 token 的 5%。GLM-130B

指出 LLM 训练最大的挑战在于稳定性，对此其提出的改进点包括：

- 采用 Post-LN(Layer Normalization)，并且用 DeepNorm [14] 来初始化，能够大大提升训练的稳定性。因为采用混合精度训练来加速训练，也就是前向传播用 float16，后向更新用 float32，但是这样做会导致训练中出现 loss spikes 问题，采用 Post-LN+DeepNorm 可以缓解这个问题。DeepNorm 也是 LN 的一种改进，主要有两点改进：
 - DeepNorm 在进行 Layer Norm(Post-LN) 之前，会以 α (大于 1) 参数扩大输入。
 - 在 Xavier 参数初始化过程中，对 FFN 中的参数， W_h^v, W_o 三部分的参数(参数的具体形式见公式??)以 β 减小部分参数的初始化范围。

不同的 transformer 架构， α 和 β 的取值不一样。DeepNorm 见图1.5。

Architectures	Encoder		Decoder	
	α	β	α	β
Encoder-only (e.g., BERT)	$(2N)^{\frac{1}{4}}$	$(8N)^{-\frac{1}{4}}$	-	-
Decoder-only (e.g., GPT)	-	-	$(2M)^{\frac{1}{4}}$	$(8M)^{-\frac{1}{4}}$
Encoder-decoder (e.g., NMT, T5)	$0.81(N^4 M)^{\frac{1}{16}}$	$0.87(N^4 M)^{-\frac{1}{16}}$	$(3M)^{\frac{1}{4}}$	$(12M)^{-\frac{1}{4}}$

图 1.5: DeepNorm 运算过程与参数取值

- 在位置编码上采用旋转位置编码 (Rotary Positional Encoding)。
- 使用 Embedding Layer Gradient Shrink (EGS) 调小 embedding 的梯度，来缓解 loss spikes 问题。EGS 其实就是计算 Embedding Layer 的梯度时乘以一个 α (比如 0.1)，但是不是计算梯度时，则还是保持原样大小。一行代码就可以实现(见图1.6)。

```
word_embedding = word_embedding * alpha + word_embedding.detach() * (1 - alpha)
```

图 1.6: EGS 实现代码

- 采用 FasterTransformer 库来加速推理，对权重采用 int4 quantization 来减小推理时显存占用。

GLM 的研发团队发布了一系列模型：

- ChatGLM [15] 使用了和 ChatGPT 相似的技术 (SFT + RLHF)，针对中文问答和对话进行了优化。
- ChatGLM2-6B [16] 的训练数据集更大，基于 FlashAttention 技术，将上下文长度由 ChatGLM-6B 的 2K 扩展到了 32K。采用了 MQA(Multi-Query Attention) 技术来加速推理。

- ChatGLM3 [17] 采用了更多样的训练数据、更充分的训练步数和更合理的训练策略。除正常的多轮对话外，同时原生支持工具调用 Function Call、代码执行 Code Interpreter 和 Agent 任务等复杂场景。

GLM-4 在十余项指标逼近或达到 GPT-4。CogView [18] 支持根据指令生成和编辑图像，实现“以文生图”和“以文改图”的功能。VisualGLM-6B [19] 能够理解图片，所以可以在对话中输入图像。CogVLM-17B [20] 拥有 100 亿的视觉参数和 70 亿的语言参数，支持 490*490 分辨率的图像理解和多轮对话。在 10 个经典的跨模态基准测试中取得了最先进的性能。CogAgent [21] 是一个基于 CogVLM 改进的开源视觉语言模型。CogAgent-18B 拥有 110 亿的视觉参数和 70 亿的语言参数，支持 1120*1120 分辨率的图像理解。在 CogVLM 的能力之上，它进一步拥有了 GUI 图像 Agent 的能力。

wujianjunml@outlook.com

wujianjunml@outlook.com

参考文献

- [1] Alec Radford, Karthik Narasimhan, Tim Salimans, Ilya Sutskever, et al. Improving language understanding by generative pre-training. 2018.
- [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, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019.
- [4] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020.
- [5] Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, et al. Training language models to follow instructions with human feedback. *Advances in Neural Information Processing Systems*, 35:27730–27744, 2022.
- [6] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.
- [7] Paul Christiano, Jan Leike, Tom B. Brown, Miljan Martic, Shane Legg, and Dario Amodei. Deep reinforcement learning from human preferences, 2023.
- [8] 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.
- [9] Prajit Ramachandran, Barret Zoph, and Quoc V. Le. Searching for activation functions, 2017.

- [10] Jianlin Su, Yu Lu, Shengfeng Pan, Ahmed Murtadha, Bo Wen, and Yunfeng Liu. Roformer: Enhanced transformer with rotary position embedding, 2023.
- [11] Hugo Touvron, Louis Martin, and Stone. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*, 2023.
- [12] Zhengxiao Du, Yujie Qian, Xiao Liu, Ming Ding, Jiezhong Qiu, Zhilin Yang, and Jie Tang. Glm: General language model pretraining with autoregressive blank infilling, 2022.
- [13] Aohan Zeng, Xiao Liu, Zhengxiao Du, Zihan Wang, Hanyu Lai, Ming Ding, Zhuoyi Yang, Yifan Xu, Wendi Zheng, Xiao Xia, Weng Lam Tam, Zixuan Ma, Yufei Xue, Jidong Zhai, Wenguang Chen, Peng Zhang, Yuxiao Dong, and Jie Tang. Glm-130b: An open bilingual pre-trained model, 2023.
- [14] Hongyu Wang, Shuming Ma, Li Dong, Shaohan Huang, Dongdong Zhang, and Furu Wei. Deepnet: Scaling transformers to 1,000 layers, 2022.
- [15] Chatglm-6b. [EB/OL]. <https://github.com/THUDM/ChatGLM-6B>.
- [16] Chatglm2-6b. [EB/OL]. <https://github.com/THUDM/ChatGLM2-6B>.
- [17] Chatglm3. [EB/OL]. <https://github.com/THUDM/ChatGLM3>.
- [18] Ming Ding, Zhuoyi Yang, Wenyi Hong, Wendi Zheng, Chang Zhou, Da Yin, Junyang Lin, Xu Zou, Zhou Shao, Hongxia Yang, et al. Cogview: Mastering text-to-image generation via transformers. *Advances in Neural Information Processing Systems*, 34:19822–19835, 2021.
- [19] Visualglm-6b. [EB/OL]. <https://github.com/THUDM/VisualGLM-6B/tree/main>.
- [20] Weihan Wang, Qingsong Lv, Wenmeng Yu, Wenyi Hong, Ji Qi, Yan Wang, Junhui Ji, Zhuoyi Yang, Lei Zhao, Xixuan Song, Jiazheng Xu, Bin Xu, Juanzi Li, Yuxiao Dong, Ming Ding, and Jie Tang. Cogvilm: Visual expert for pretrained language models, 2024.
- [21] Wenyi Hong, Weihan Wang, Qingsong Lv, Jiazheng Xu, Wenmeng Yu, Junhui Ji, Yan Wang, Zihan Wang, Yuxuan Zhang, Juanzi Li, Bin Xu, Yuxiao Dong, Ming Ding, and Jie Tang. Cogagent: A visual language model for gui agents, 2023.