

目录

0 从PLM到LLM

1 LLM的基础架构及演化

2 LLM 预训练

3 LLM 微调与对齐

4 LLM 推理优化

5 模型编辑

从 PLM 到 LLM：预训练模型架构的演进之路

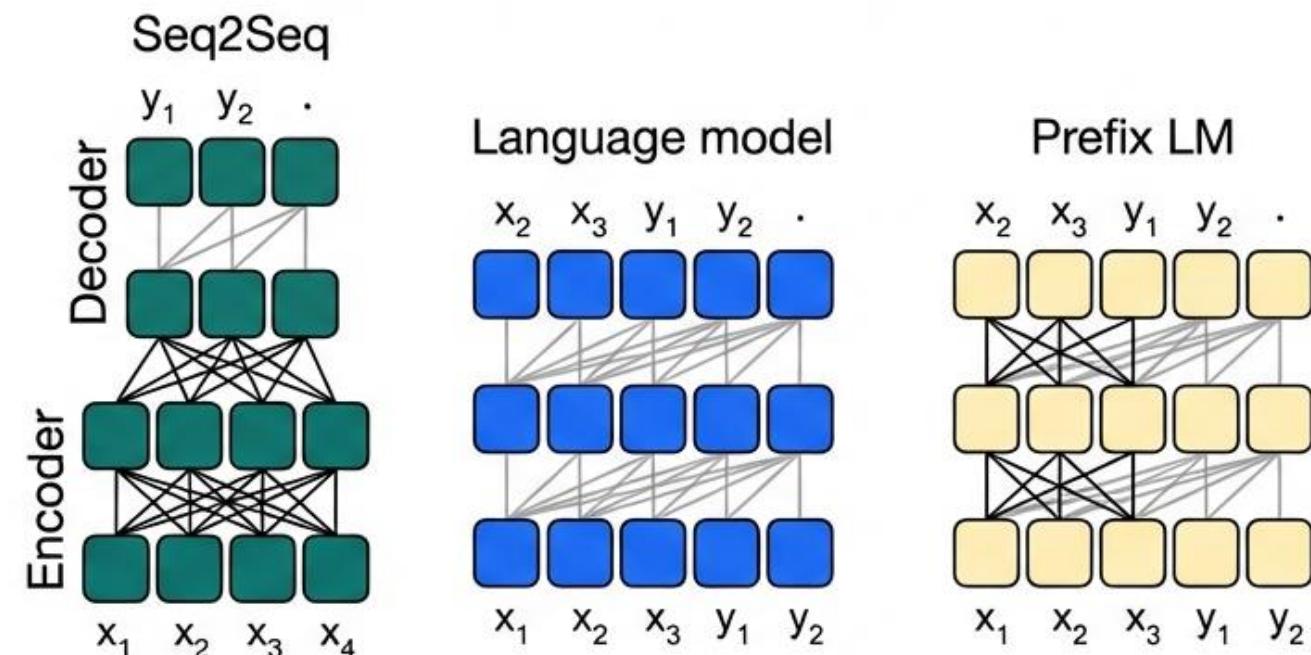
基于 BERT、T5 与 GPT 的架构解析与范式比较

Transformer 的演变与分化

针对 Encoder 与 Decoder 的不同特性，预训练模型发展出三种主流架构思路。

主要架构路线

- BERT (Encoder-only) : 专注 NLU, 堆叠 Encoder, 双向理解。
- GPT (Decoder-only) : 专注 NLG, 堆叠 Decoder, 生成式任务。
- T5 (Encoder-Decoder) : 保留完整结构, 探索大一统范式。
- **核心范式:** 从单纯模型发展为“预训练+微调”模式。



三种基于 Transformer 的主流架构对比：Seq2Seq, Language Model, Prefix LM。

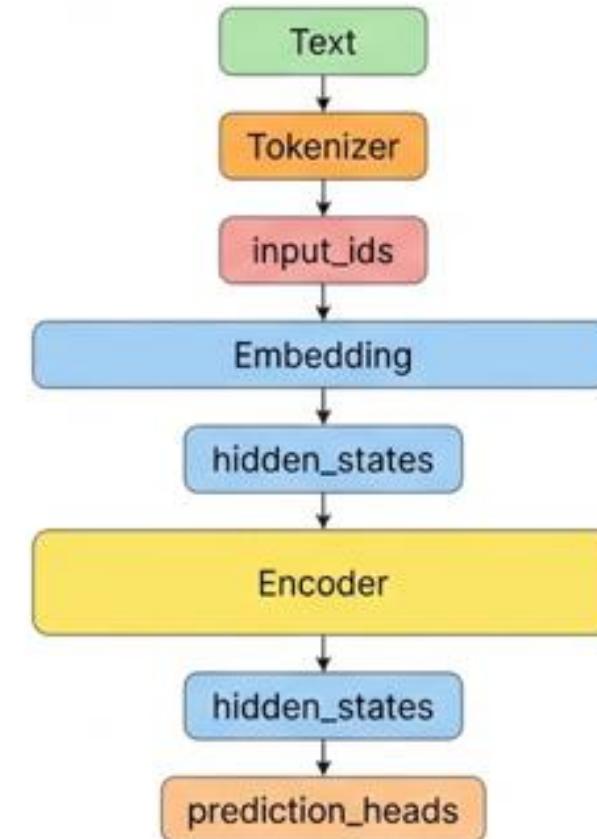
Encoder-only PLM—BERT：面向 NLU 的深度双向表征 (2018-10)

舍弃 Decoder 以专注文本理解，通过堆叠 Encoder 获取高质量语言特征。

BERT--Bidirectional Encoder Representations from Transformers

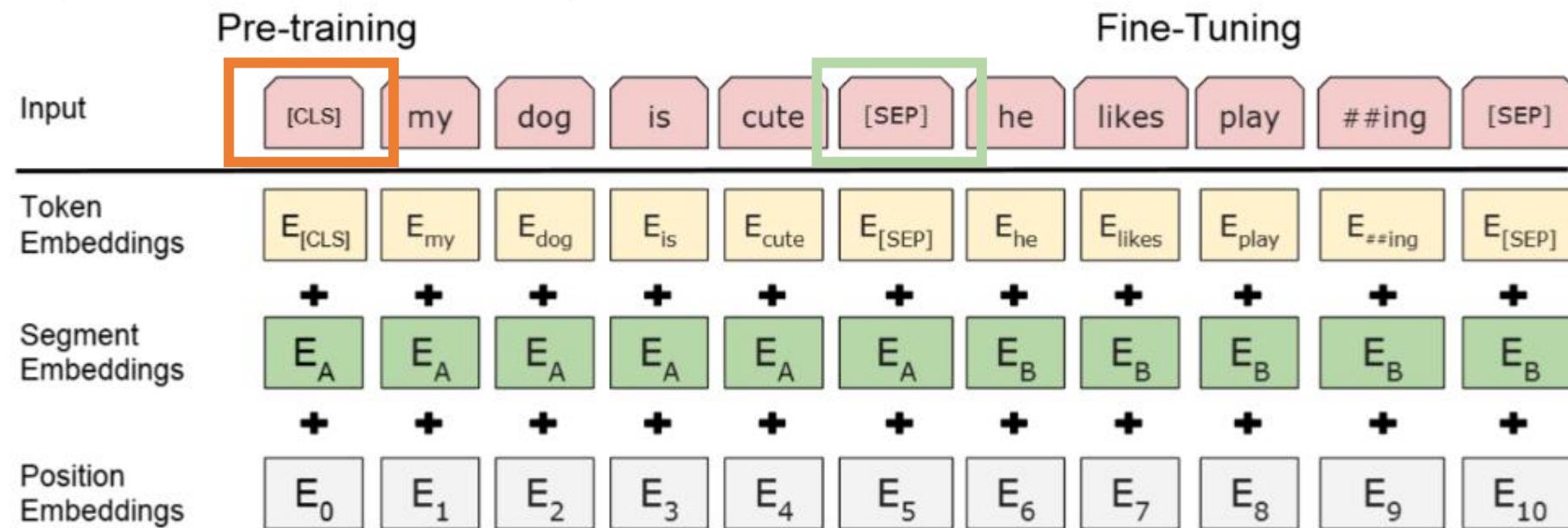
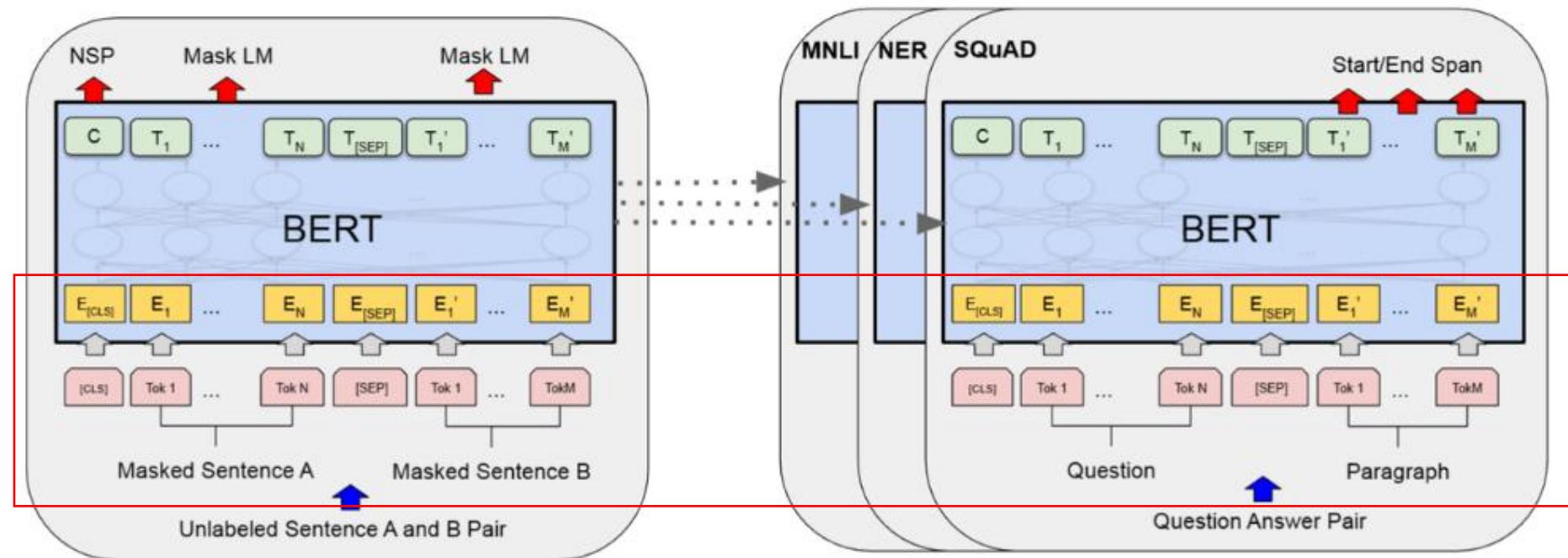
《BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding》

- **核心思想：**堆叠 Encoder 扩大参数，专注“理解”而非“生成”。
- **范式革新：**继承 ELMo“预训练+微调”思路，推向高潮。
- **模型规模：**Base (12层，110M参数) 与 Large (24层，340M参数)。
- **输出层：**通过 Prediction Head 将隐藏状态映射到分类维度。



BERT 模型处理流程：从 Embedding 到 Encoder 堆叠再到 Prediction Head。

输入的改进



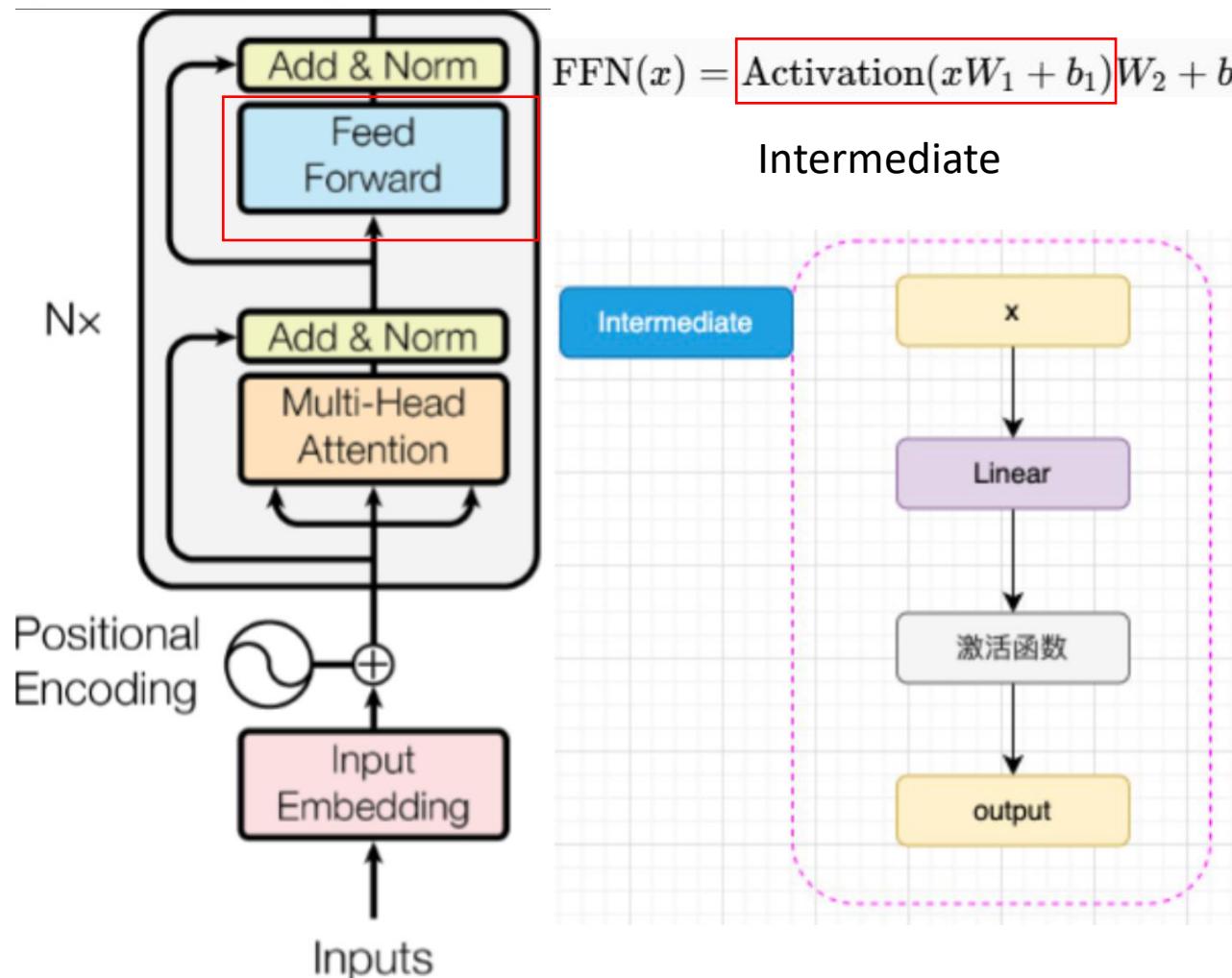
Encoder 内部结构的细微改进

引入 GELU 激活函数，通过随机正则思想提升深层模型表现。

- **结构拆分：**Transformer 的 FFN 被拆分为 **Intermediate** 和 **Output** 层。
- **激活函数：**弃用 ReLU，改用 **GELU**（高斯误差线性单元）。
- **GELU原理：**引入随机性，输入越小被丢弃概率越大。
- **优势：**在深层架构中提供更细致梯度流，捕捉复杂语义。

$$GELU(x) = 0.5x(1 + \tanh(\sqrt{\frac{2}{\pi}}(x + 0.044715x^3)))$$

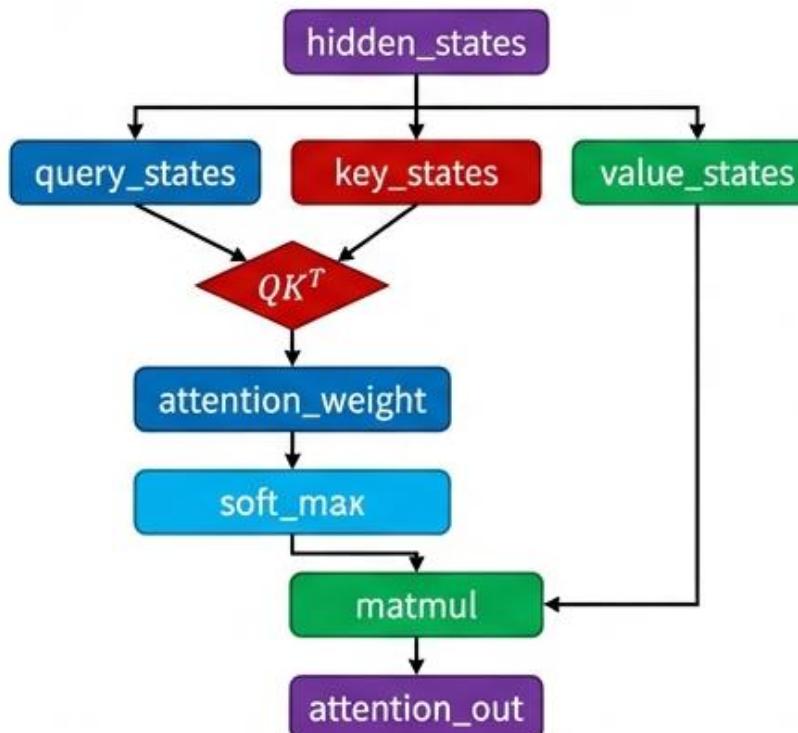
GELU 激活函数公式，结合了随机正则的思想



对抗“浅层拼接”的深度双向机制

BERT 在所有层中同时融合左右上下文，超越了 ELMo 的浅层拼接模式。

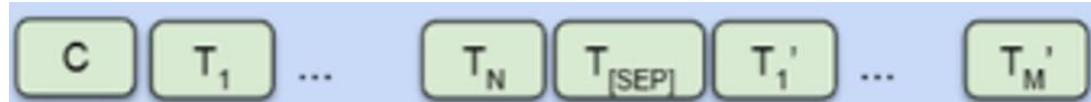
- **全向可见：** 使用无掩码 Self-Attention，全景式融合信息。
- **动机：** 词的含义由两侧词共同决定，单向模型存在缺陷。



- **对比 ELMo：** ELMo 仅在输出层拼接双向 LSTM，属于“浅层拼接”。
- **BERT 优势：** 每一层都在同时权衡左右两边信息 (Deep Fusion)。

BERT 的 Attention 机制允许 Query 关注所有 Key/Value，无因果掩码。

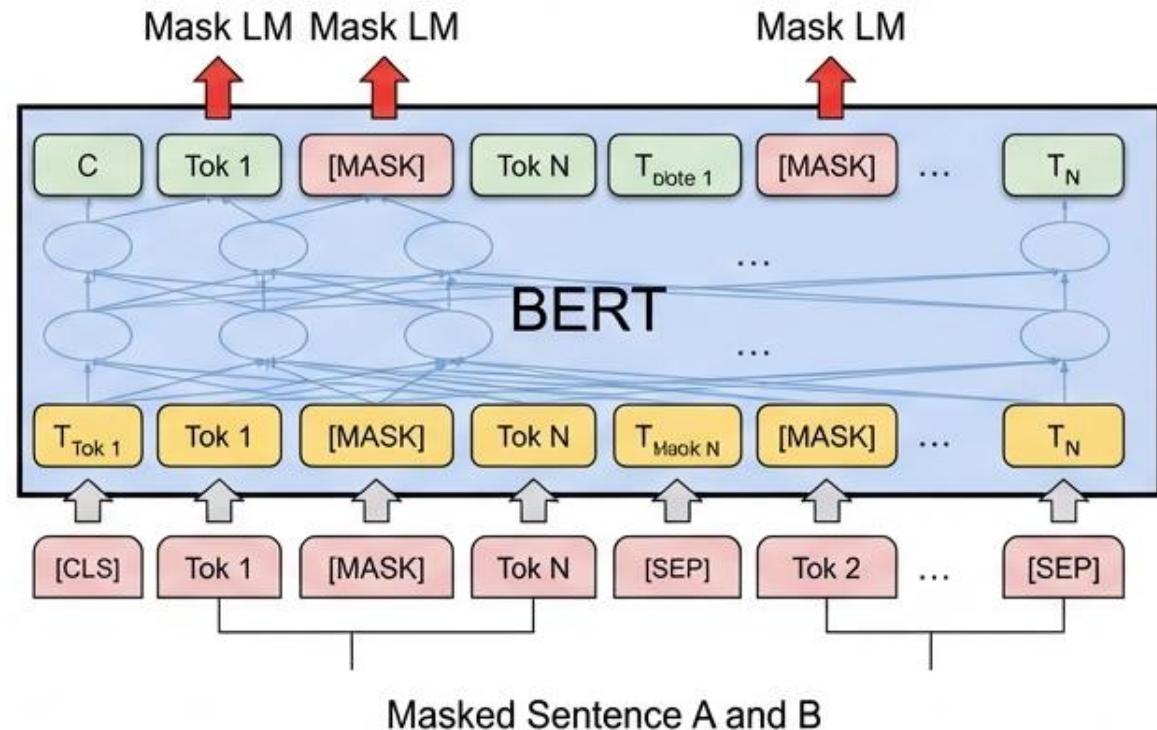
BERT的输出：



Masked Language Model (MLM)

通过“完形填空”任务强迫模型拟合双向语义，解决双向注意力带来的标签泄露问题。

- **机制：**随机遮蔽输入序列中的部分 Token，要求模型预测。
- **必要性：**传统 LM 若双向看会泄露标签，MLM 避免了此问题。
- **数据优势：**利用海量无监督语料，无需人工标注。
- **缺陷：**预训练的 **[MASK]** 标记在微调/推理时不存在（一致性问题）。



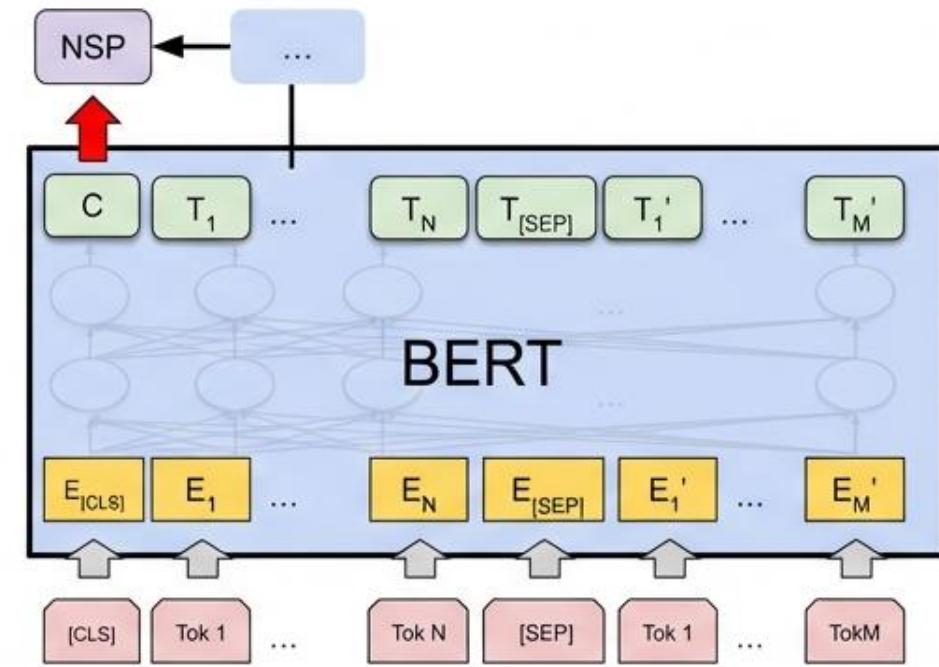
MLM 任务示意：输入被遮蔽的句子，利用上下文预测 [MASK]。

解决：15% token 预测-->80% Mask / 10% Random / 10% Original

Next Sentence Prediction (NSP)

Thesis: 针对句级 NLU 任务设计的预训练目标，增强模型对句子关系的理解。

- **目标:** 判断两个句子是否为连续上下文（二分类任务）。
- **样本构建:** 正样本为连续句，负样本为随机抽取的句子。
- **适用任务:** 问答匹配 (QA) 、自然语言推理 (NLI) 。
- **训练规模:** 3.3B token, 40 个 Epoch (1M steps) 。



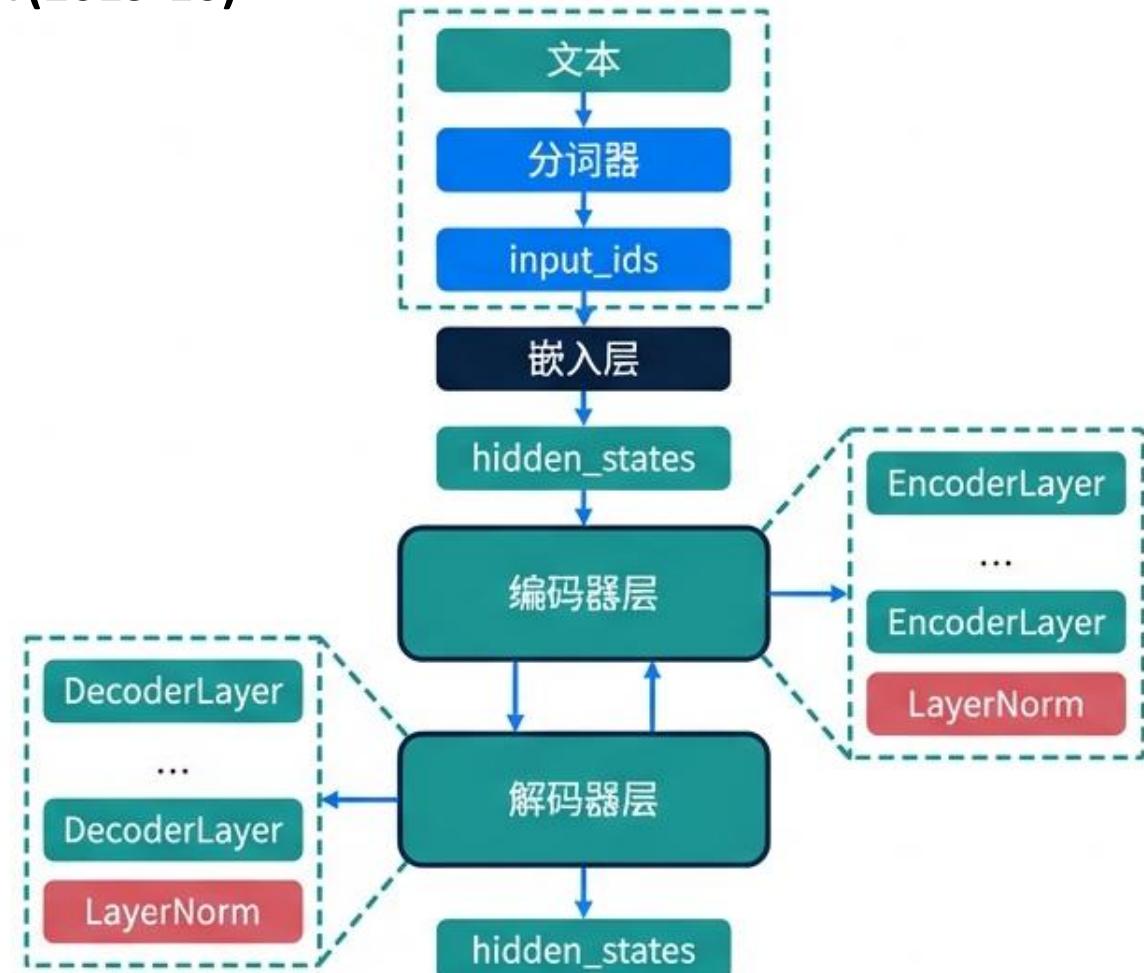
通过 [CLS] Token 对应的输出进行二分类预测，判断句子关系。

T5：探索 NLP 的大一统范式

将所有 NLP 任务统一为“文本到文本”的转换问题，简化模型设计。

T5：Text-To-Text Transfer Transformer(2019-10)

- **架构选择：**经实验验证，Encoder-Decoder 结构效果最佳。
- **核心理念：**Text-to-Text，输入输出均为纯文本。
- **通用性：**分类、翻译、摘要均使用同一套模型和训练框架。
- **模块化：**Encoder 和 Decoder 由可堆叠的 Block 组成。



T5 模型架构：标准的 Transformer Encoder-Decoder 结构。

T5 的架构细节优化

论文：引入相对位置偏置和 RMSNorm，提升模型对长文本的适应性与训练稳定性。

- **相对位置偏置：**替代绝对位置编码，通过学习相对距离的标量偏置。
- **对数分桶：**对远距离位置使用对数增长范围，增强长文本鲁棒性。
- **RMSNorm：**简化版 LayerNorm，去除了加性偏置 (Bias)。
- **Norm位置：**置于残差连接之前 (Pre-Norm 思想)。

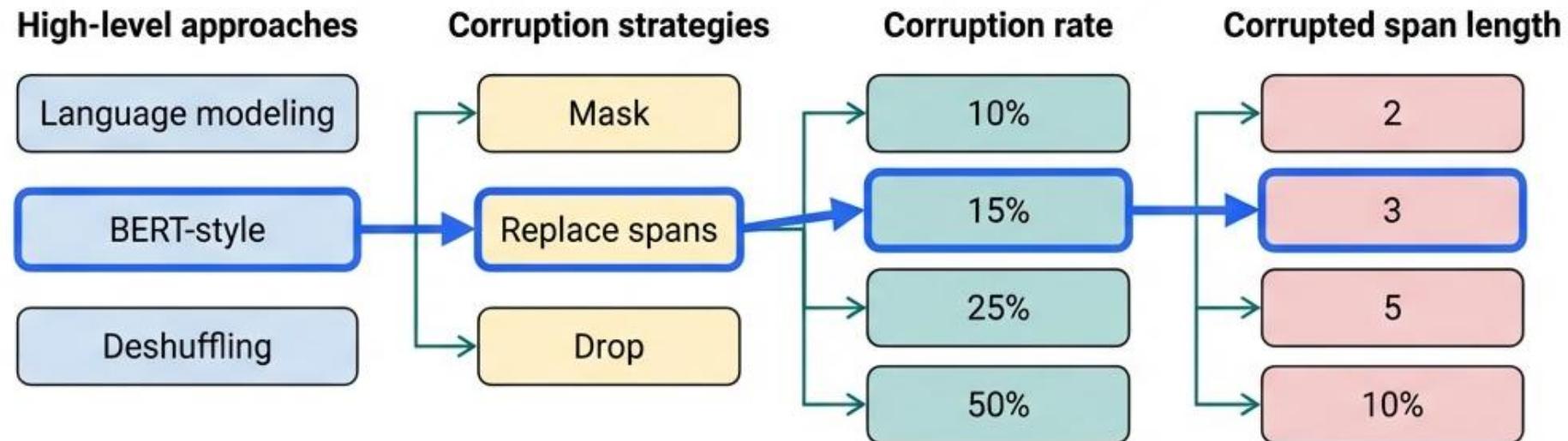
$$\text{Attention}(Q, K, V) = \text{softmax} \left(\frac{QK^T}{\sqrt{d_k}} + \mathbf{B} \right) V$$

$$\text{RMSNorm}(x) = \frac{x}{\sqrt{\frac{1}{n} \sum_{i=1}^n x_i^2 + \epsilon}} \cdot \gamma$$

上：带相对位置偏置 \mathbf{B} 的注意力计算；下：RMSNorm 公式（仅缩放）。

预训练策略的全面探索

T5 对预训练目标进行了大规模实验，确立了 BERT-style (Span Mask) 的最优地位。

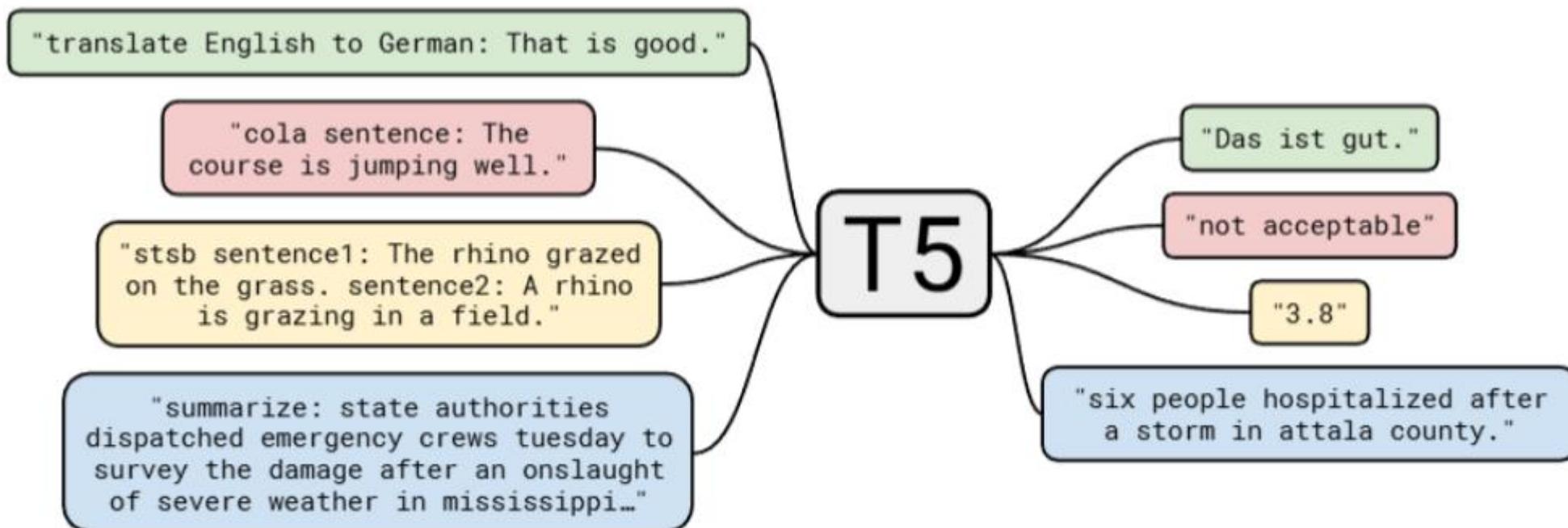


T5 团队对预训练策略的四个维度进行的系统性对比实验。

- **高层方法：**BERT-style 优于 LM 和 Deshuffling。
- **破坏策略：**Replace Spans 优于 Mask 和 Drop。
- **破坏比例：**15% 的破坏率效果最佳。
- **Span长度：**平均长度为 3 的 Span 效果最好。

大一统思想：Text-to-Text

通过任务描述前缀（Task Prefix），将不同任务映射到同一语义空间。



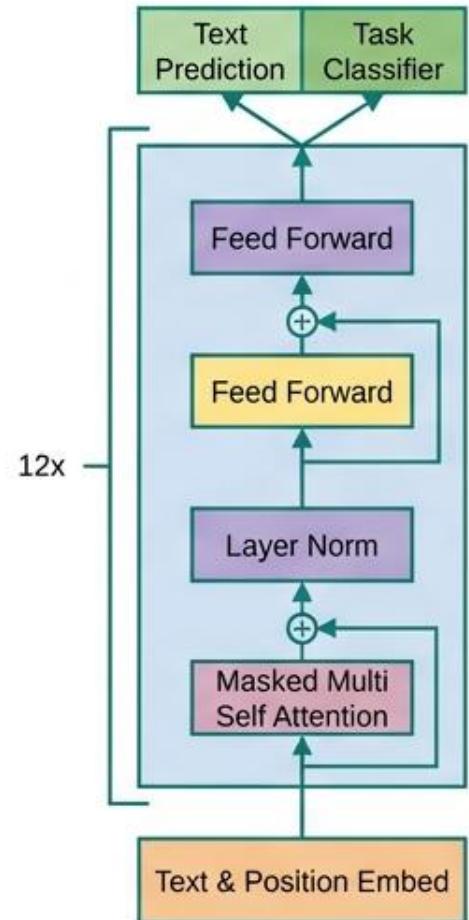
翻译、分类、回归、摘要等任务均被转化为文本生成任务。

- 统一格式: Input: 'Task Prefix: Content' -> Output: 'Target Text'。
- 任务前缀: 如 'translate English to German:', 'summarize:'。
- 优势: 参数共享, 无需针对特定任务修改输出头。
- 数据: C4 数据集 (750GB), 多任务混合预训练。

GPT：生成式预训练 (2018-06)

坚持 Decoder-Only 架构，通过通用预训练与微调范式，奠定现代 LLM 基石。

- **定义：**Generative Pre-Training，通过预测下一个词进行训练。
- **历史地位：**首次明确提出“预训练+微调”思想（早于 BERT）。
- **核心组件：**堆叠 Decoder，Masked Self-Attention。
- **愿景：**Ilya Sutskever 认为“完美预测下一个词即理解世界”。



GPT-1 结构：12层 Decoder，结合特定任务的输入变换。

为什么选择 Decoder-Only 架构？

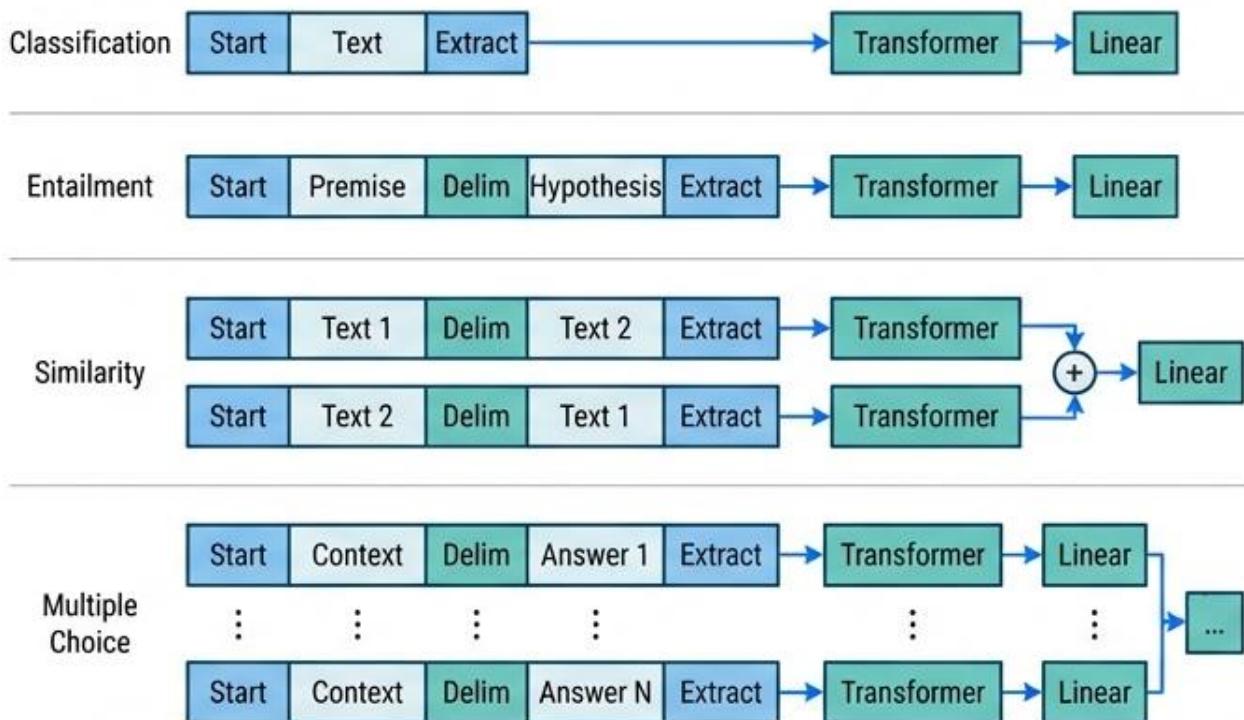
Decoder 架构在长距离依赖建模和通用生成能力上具有天然优势。

-  1. 任务契合 生成式任务 (CLM) 本质是单向预测，需 Mask 未来信息。
-  2. 结构化记忆 相比 LSTM，Self-Attention 解决长距离依赖瓶颈。
-  3. 生成即理解 具备生成能力的模型必然包含理解能力。
-  4. 简洁通用 结构比 Enc-Dec 简单，参数利用率高，易于 Scaling。
-  5. 防标签泄露 单向结构天然避免了 Encoder 双向“偷看”答案的问题。

GPT 的无监督预训练与有监督微调

结合因果语言建模 (CLM) 与任务感知输入变换，实现广泛迁移。

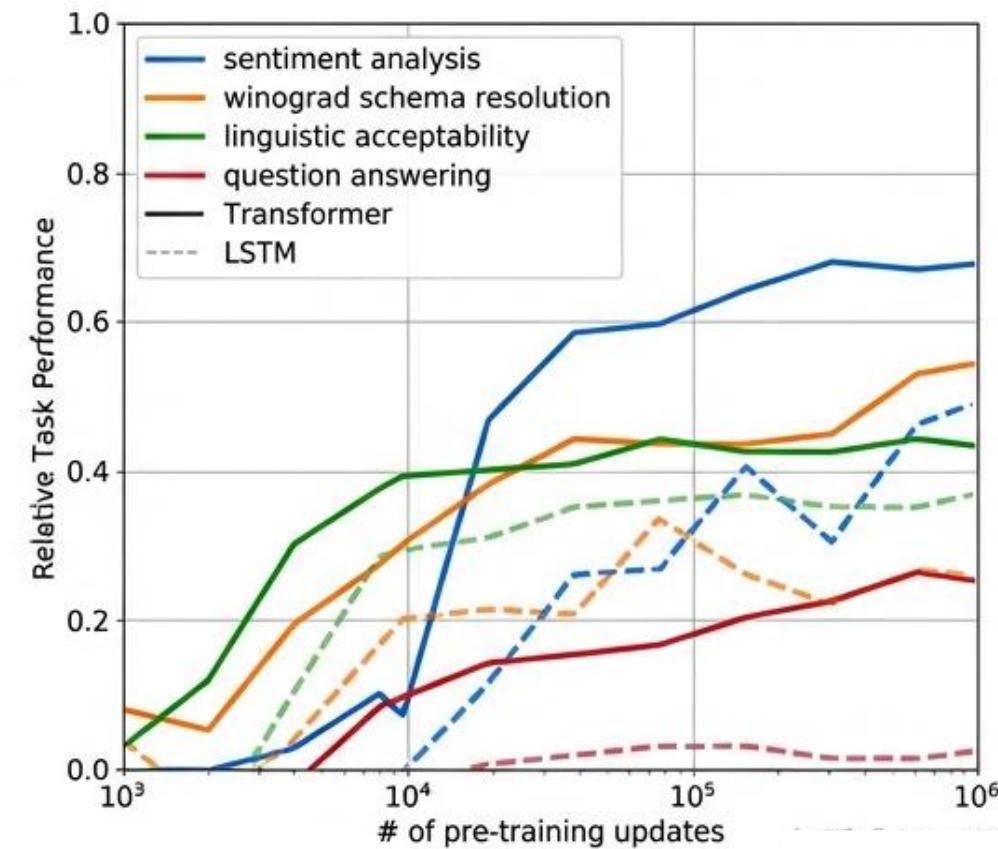
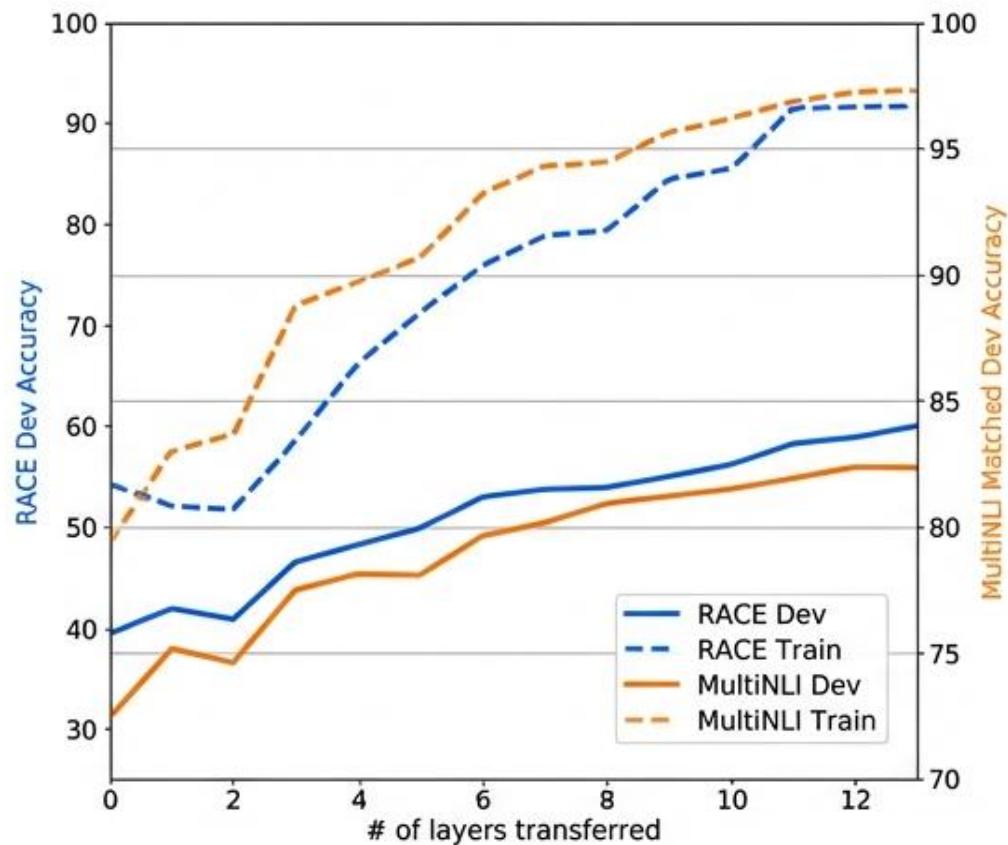
- **预训练 (CLM)**：最大化给定上文预测当前词的似然概率。
- **微调**：将最后一层输出输入线性层，最大化有监督目标。
- **辅助目标**：微调时加入语言模型辅助目标，提升泛化能力。
- **输入变换**：通过起始符、分隔符、抽取符将不同任务序列化。



分类、蕴含、相似度、多选题的输入格式变换示意。

规模法则与 Zero-shot 潜力的初现

GPT-1 的实验证明了性能随层数增加而提升，并展现了零样本学习的雏形。



左图：模型深度与性能的正相关性；右图：预训练步数与 Zero-shot 性能的关系。

层数效应：随着解码器层数增加，NLU 任务准确率持续上升。

Zero-shot：预训练进行中，模型在各种任务上的相对性能逐渐增强。

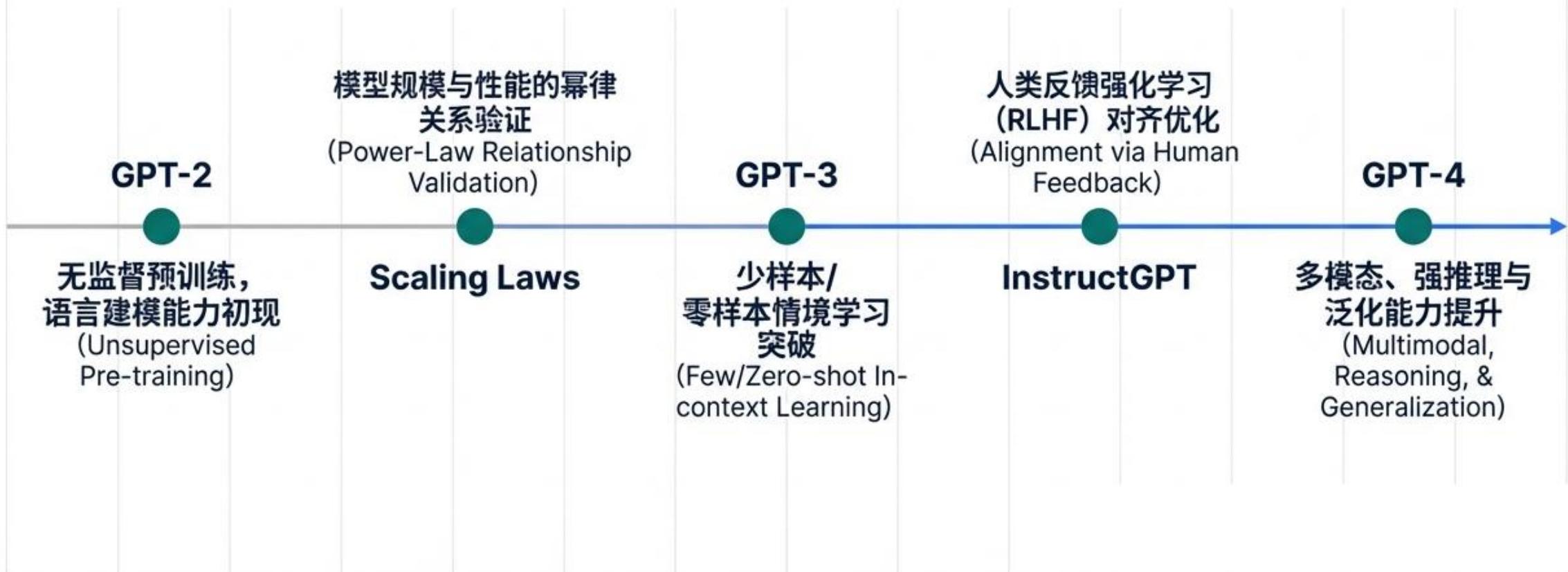
启示：预示了 GPT-2/3 的方向——更大参数、更多数据。

总结：

三种架构各展所长，最终 Decoder-Only 架构凭借其生成能力与扩展性成为 LLM 主流。

	BERT	T5	GPT
Architecture	Encoder-Only (Bidirectional)	Encoder-Decoder (Unified)	Decoder-Only (Auto-regressive)
Core Strength	NLU (Deep Understanding)	Transfer Learning (Text-to-Text)	NLG (Generative & Scaling)
Legacy	确立“深度双向”在 NLU 的地位	验证“万物皆文本”的 大一统思想	验证 Scaling Laws， 开启 LLM 时代

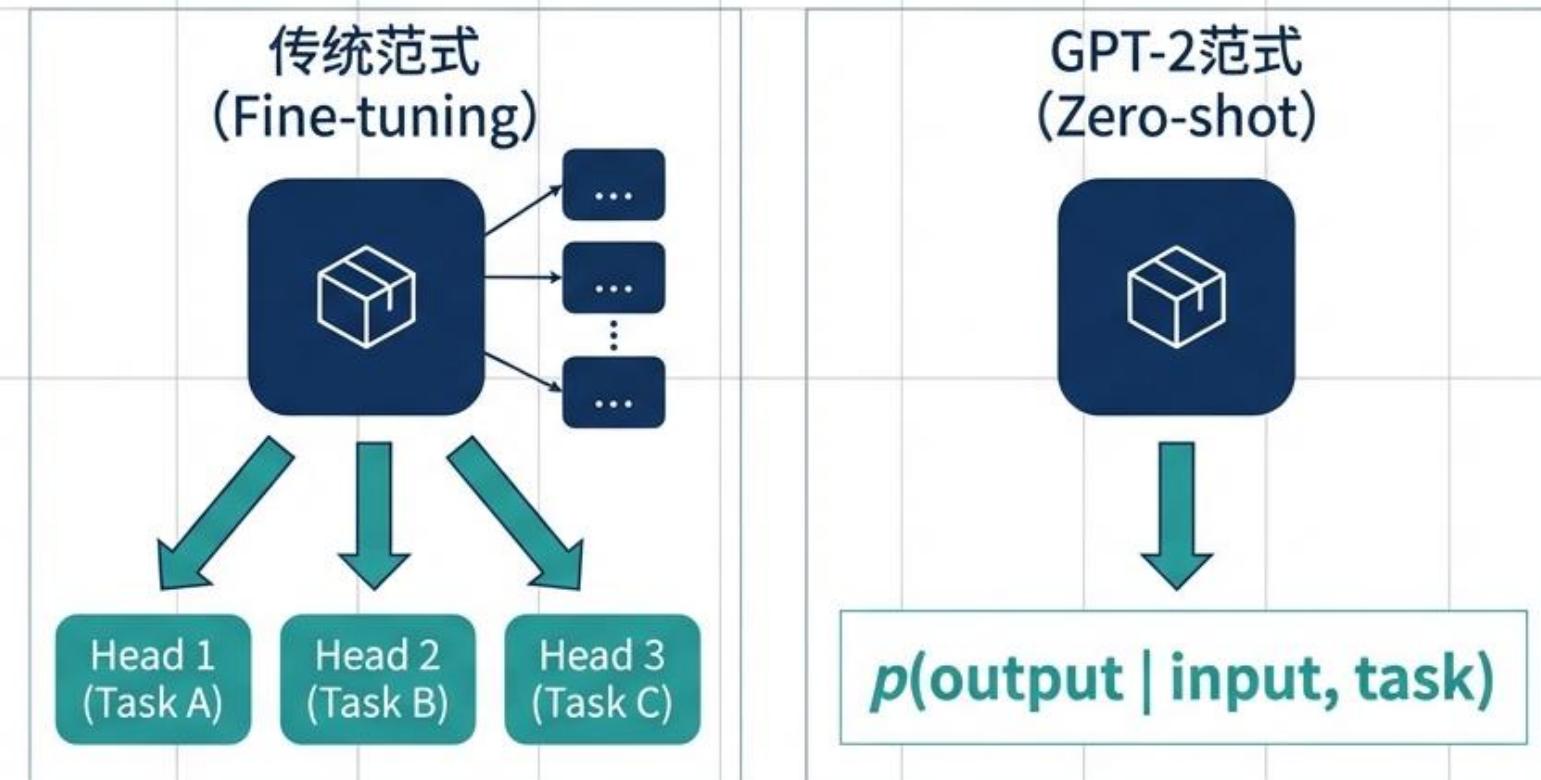
从PLM到LLM：GPT系列的演进与范式转移



GPT-2：迈向零样本无监督多任务学习

《Language Models are Unsupervised Multitask Learners》

- **范式转变：**从预训练+微调向转向零样本多任务
- **核心假设：**语言建模本质即多任务学习
- **数据基础：**WebText (40GB)，人工过滤高质量网页
- **能力涌现：**1.5B参数模型在多任务达SOTA
- **参数量：** GPT-2 XL 1.5B，相对于GPT-1提升了 10 倍以上
- **词汇量扩展到**50,257
- **Layers** 从 12 层-> 48 层, **dmodel** 从 768->1600, **Context Window**从 512 个 token->1024 个 token



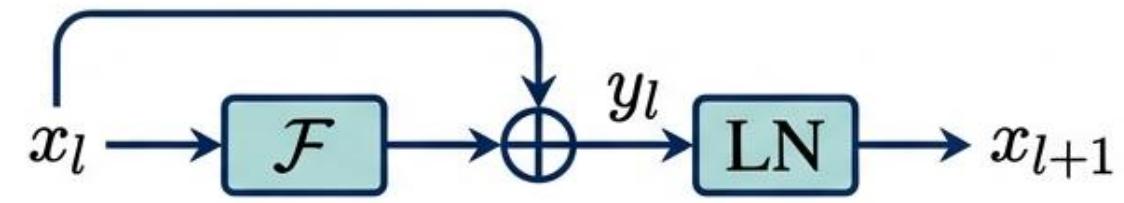
为特定任务定制输出头与微调

统一建模：单一模型，零样本推理

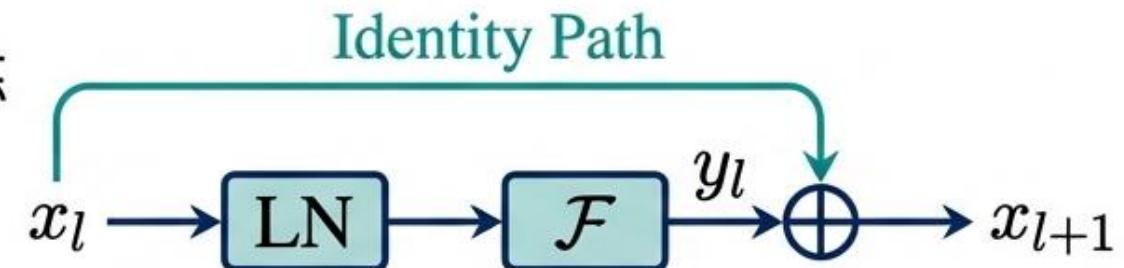
GPT-2架构优化：Pre-norm机制

《Learning Deep Transformer Models for Machine Translation》

- 位置调整：Post-norm → Pre-norm
- 恒等路径：输入 x 直通深层，保护梯度流
- 解决痛点：缓解深层网络的梯度消失/爆炸
- 实验验证：同类工作证明Pre-norm更易训练



(a) Post-norm residual unit

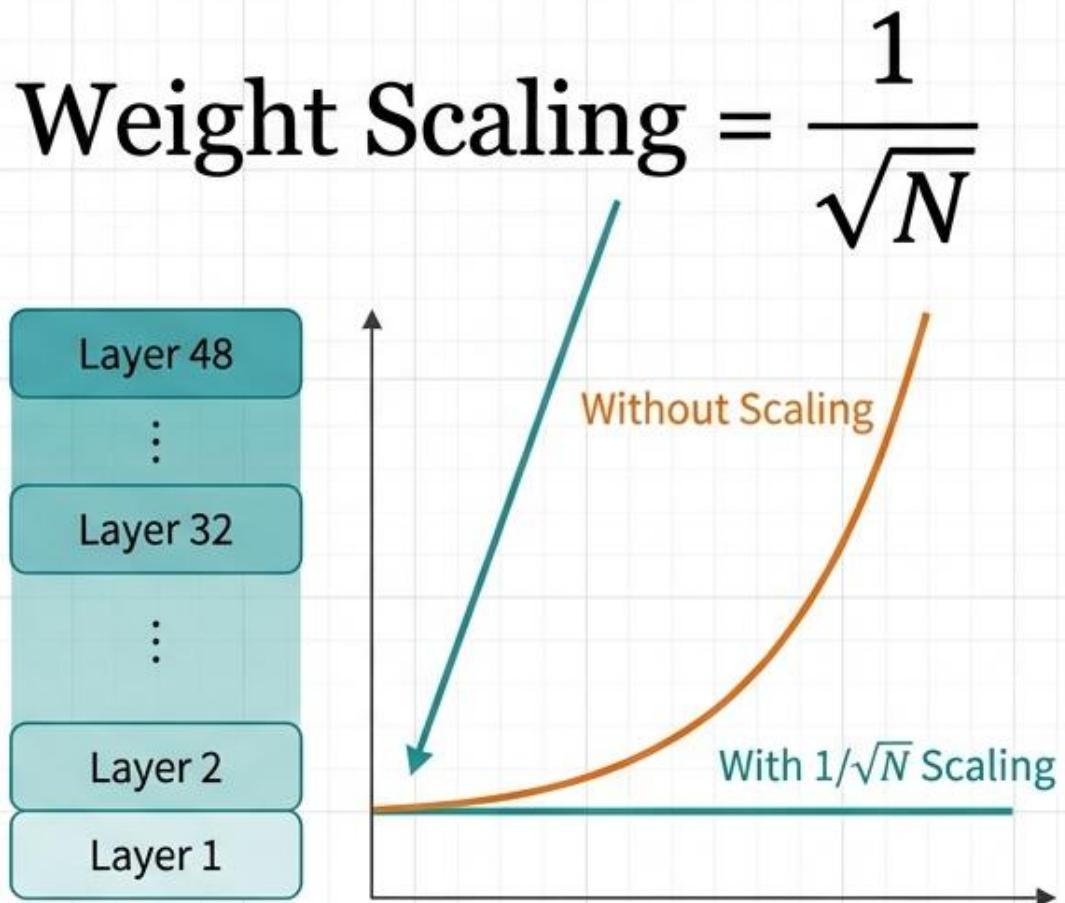


(b) Pre-norm residual unit

GPT-2架构优化：权重初始化缩放

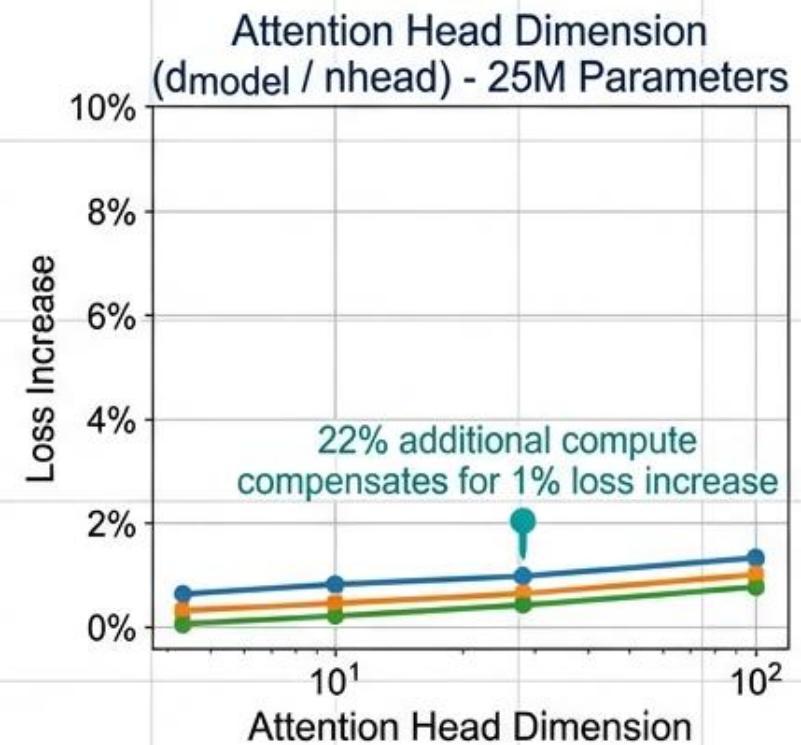
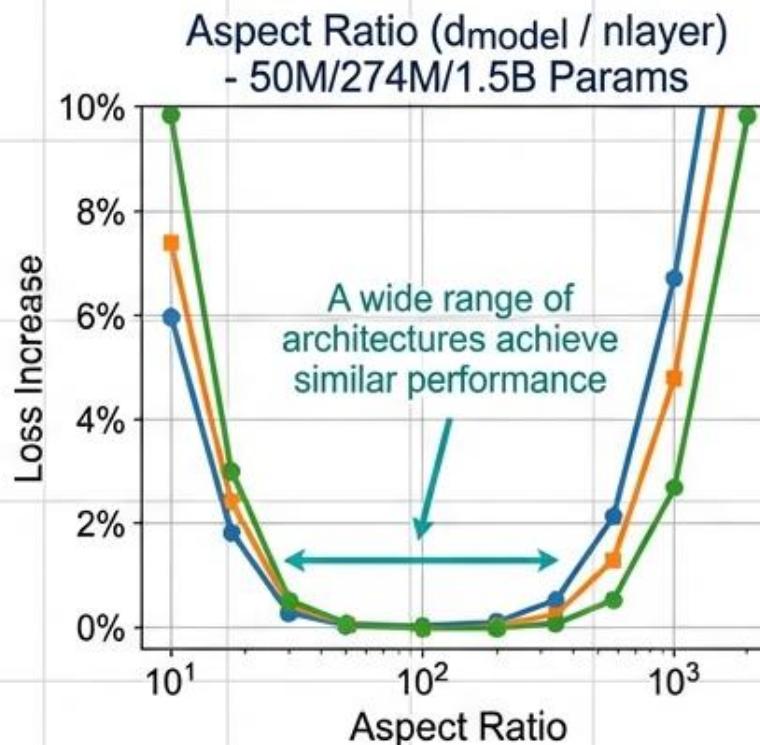
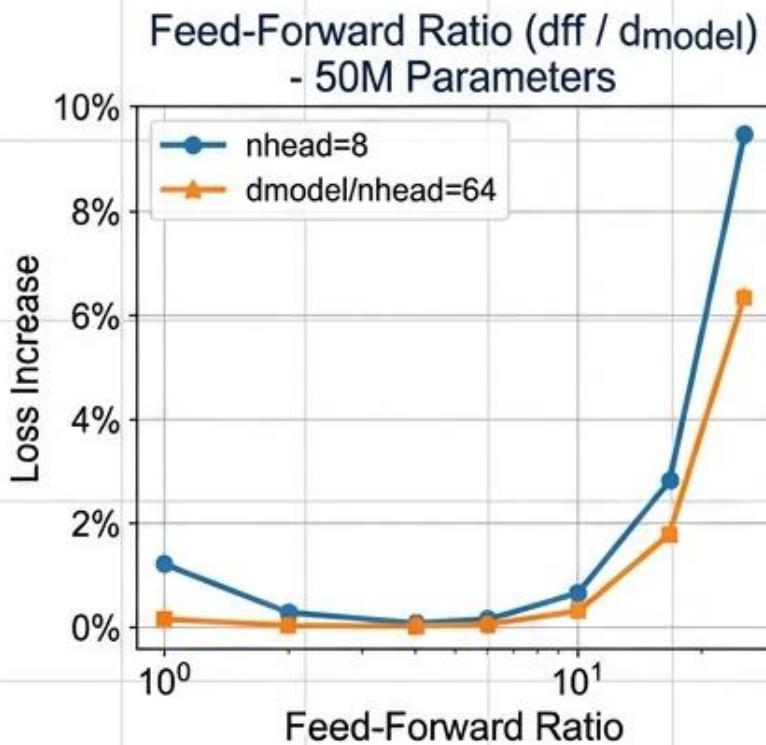
$$x_{l+1} = x_l + \text{Sublayer}(x_l)$$

- 方差问题：残差累加导致主干路径数值膨胀
- 缩放公式：权重缩放因子 $1/\sqrt{N}$
- 参数定义： N 为残差层总数（GPT-2中 $N=48$ ）
- 效果：总方差保持在 ≈ 1 量级，防止饱和



Scaling Laws: 规模优于架构

- 核心发现：性能依赖规模，而非架构细节
- 架构独立性：深度/宽度比例变化对Loss影响微小
- 关键因素：非嵌入参数N、数据量D、计算量C $C \approx 6ND \approx 6NBS$
- 关键因素：非嵌入参数N、数据量D、计算量C
- 补偿机制：22%额外计算量可补偿1% Loss



Scaling Laws: 幂律关系公式

《Scaling Laws for Neural Language Models(2020)》

研究内容：模型性能 L 与模型大小 N , 训练数据量 D , 训练模型所需要的计算量 C 的关系

- 通用规律: $L(X) \propto X^{-\alpha}$, 跨越六个数量级 $L(D) = (\frac{D_c}{D})^{\alpha_D}$ $L(N) = (\frac{N_c}{N})^{\alpha_N}$ $L(C) = (\frac{C_c}{C})^{\alpha_C}$
- 参数量 N : 非嵌入参数决定性能上限
$$L(N, D) = \left[\left(\frac{N_c}{N} \right)^{\alpha_N/\alpha_D} + \left(\frac{D_c}{D} \right) \right]^{\alpha_D}$$
- 联合幂律: 需同时增加 N 和 D 避免过拟合
- 计算关系: $C \approx 6ND$ (Transformer架构)

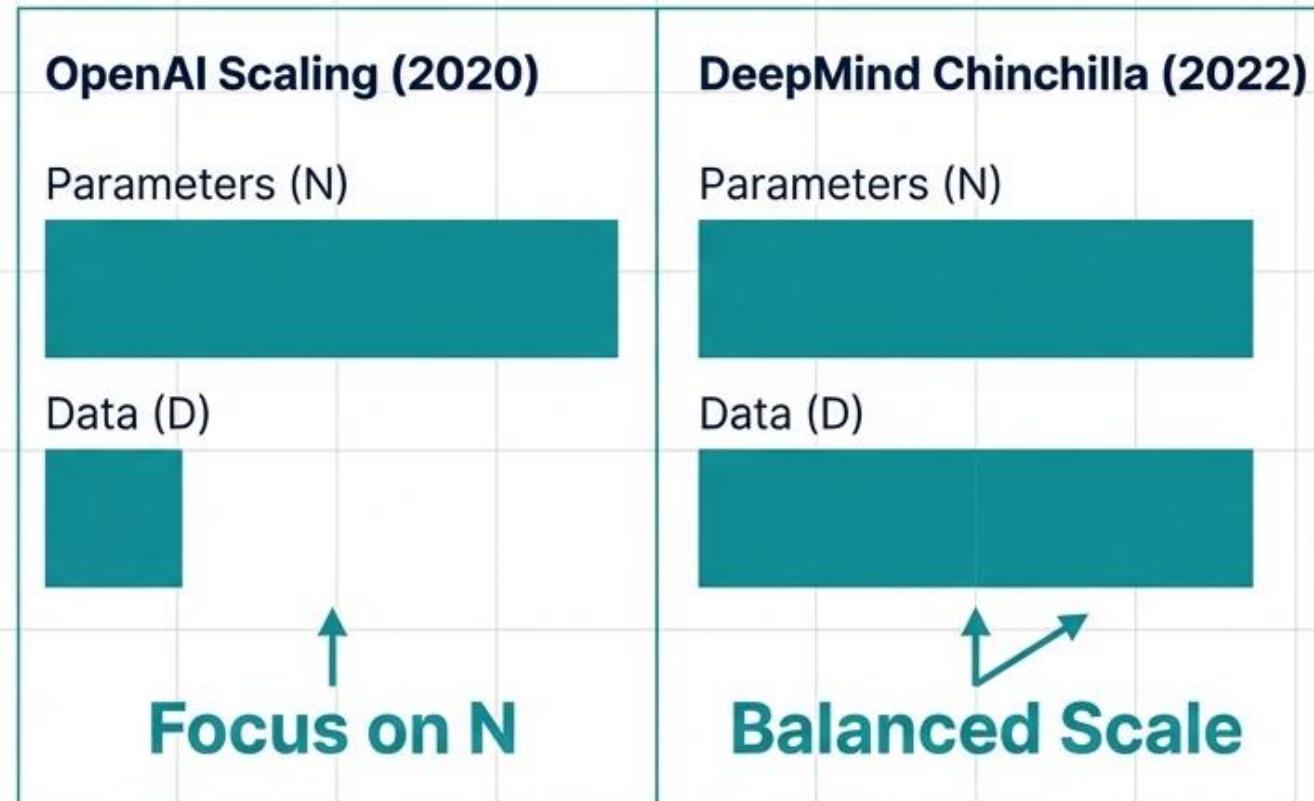
N = Model Parameters

D = Dataset Size

L = Test Loss

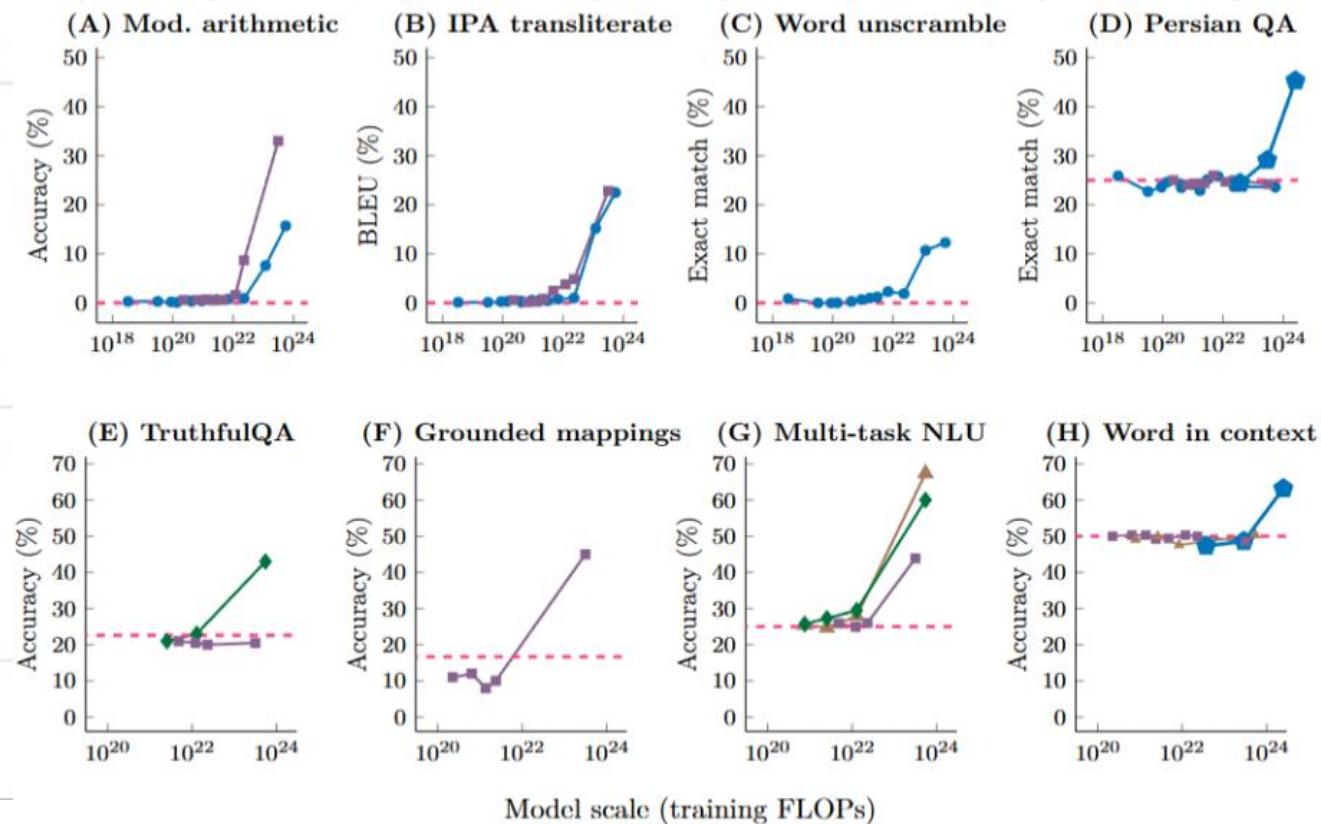
Scaling Laws：最优算力分配

- 核心问题：给定算力预算，如何分配N和D？
- OpenAI观点：优先增加参数 ($N \uparrow 5.5x$, $D \uparrow 1.8x$)
- DeepMind观点：两者同等重要 ($N \uparrow 3.16x$, $D \uparrow 3.16x$)
- 工程意义：从“赌博”转变为可预测的工程规划



模型的涌现：量变引起质变

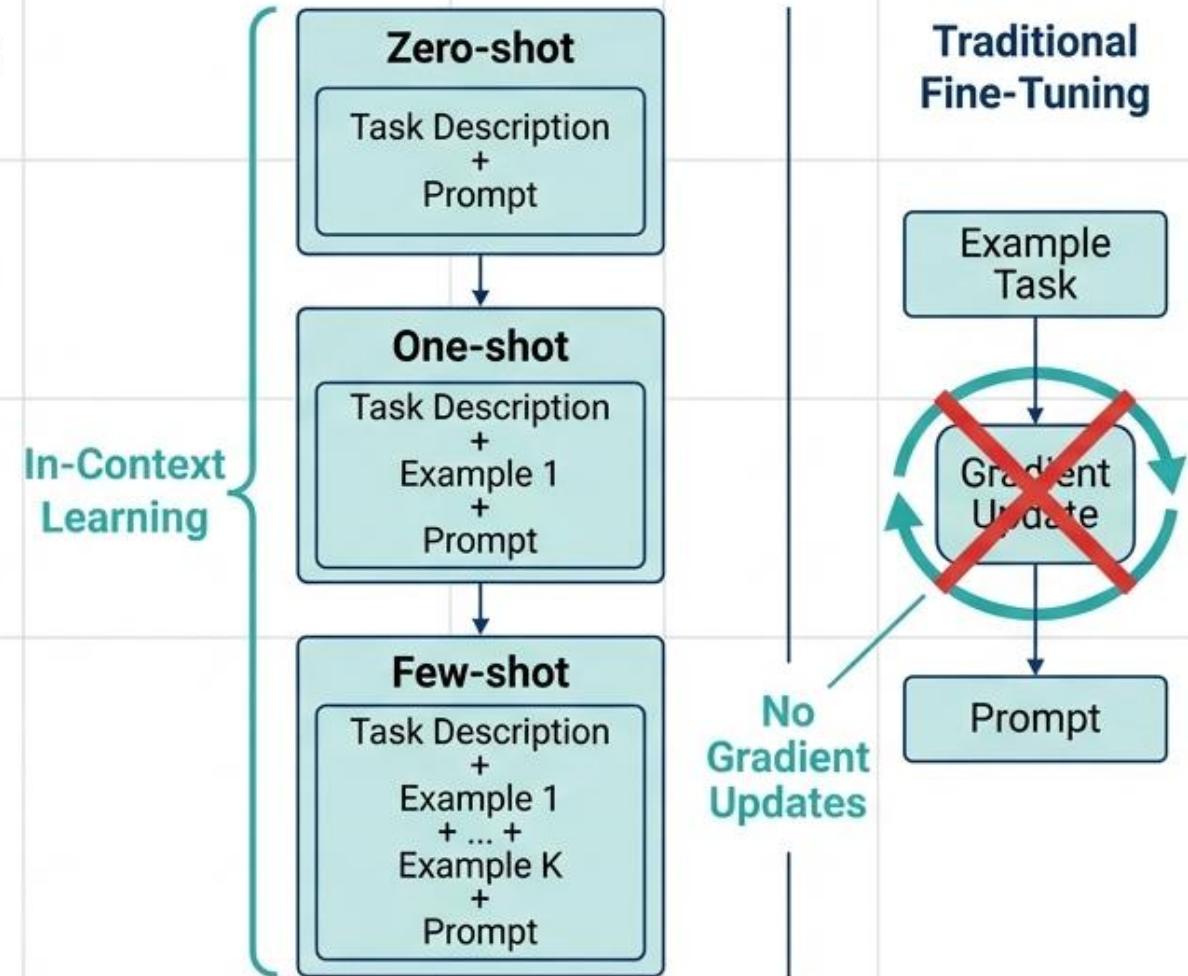
- **现象定义：**小模型接近随机，大模型突然解决
- **非线性：**Loss平滑下降 ≠ 能力平滑提升
- **阈值效应：**算术、代码等复杂任务表现明显
- **规模验证：**验证了100B+参数存在的必要性



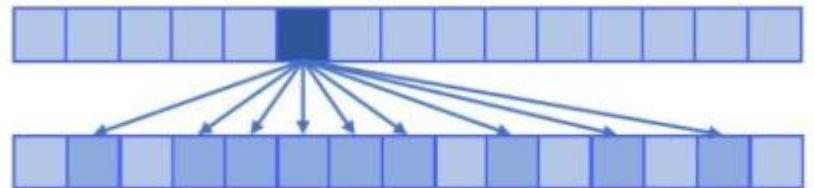
GPT-3: 情境学习 (In-Context Learning)

《Language Models are Few-Shot Learners》

- **核心定义:** Few-shot是Zero-shot的折中与改进
- **工作机制:** 在Prompt中提供10-100个示例
- **本质区别:** 推理时不进行梯度更新
(No Gradient Updates)
- **解决痛点:** 无需微调海量数据, 提升泛化性



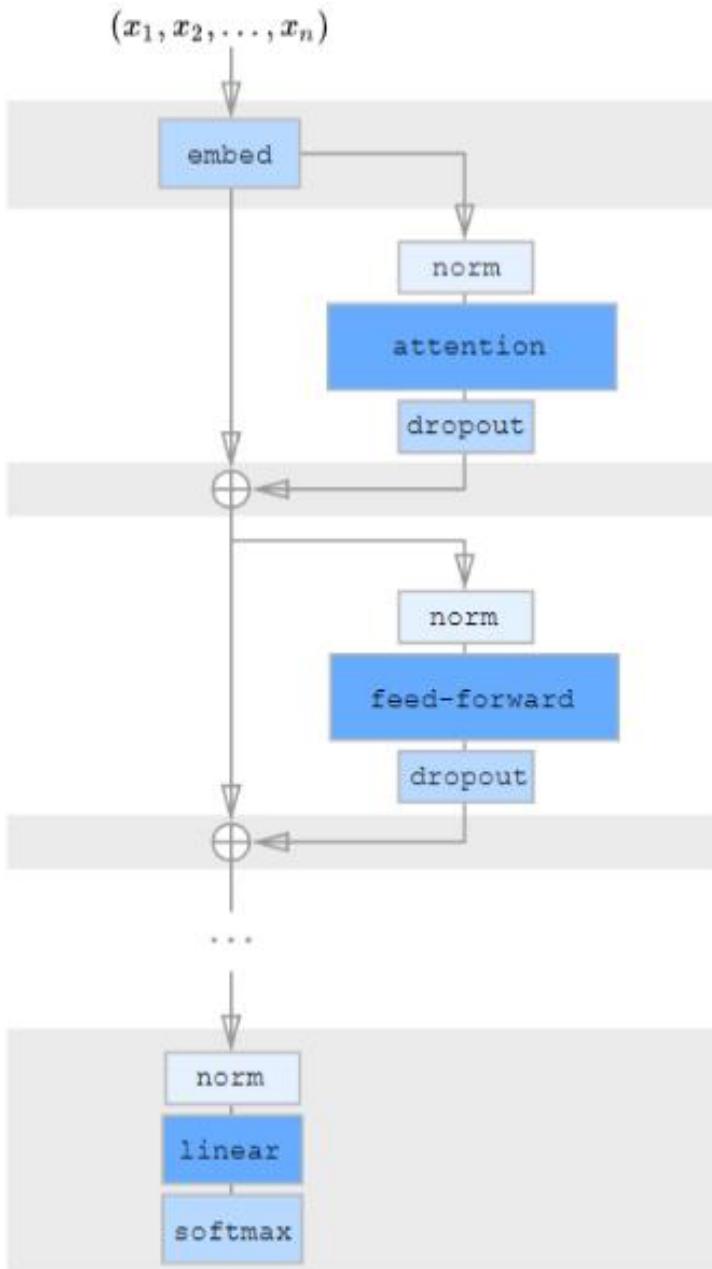
GPT-3架构：稀疏注意力机制



sparse attention ($k = 2$) 初手 @AaronWu

- 复杂度优化：Dense $O(n^2)$ → Sparse $O(n \log n)$
- 关注模式：固定步长 + 局部Token关注
- 上下文：支持更长的窗口（2048 tokens）
- 参数规模：175B参数，96层Transformer

Model Name	n_{params}	n_{layers}	d_{model}	n_{heads}	d_{head}	Batch Size	Learning Rate
GPT-3 Small	125M	12	768	12	64	0.5M	6.0×10^{-4}
GPT-3 Medium	350M	24	1024	16	64	0.5M	3.0×10^{-4}
GPT-3 Large	760M	24	1536	16	96	0.5M	2.5×10^{-4}
GPT-3 XL	1.3B	24	2048	24	128	1M	2.0×10^{-4}
GPT-3 2.7B	2.7B	32	2560	32	80	1M	1.6×10^{-4}
GPT-3 6.7B	6.7B	32	4096	32	128	2M	1.2×10^{-4}
GPT-3 13B	13.0B	40	5140	40	128	2M	1.0×10^{-4}
GPT-3 175B or “GPT-3”	175.0B	96	12288	96	128	3.2M	0.6×10^{-4}



GPT-3：大规模数据清洗与采样

- 总规模：3000亿 Tokens 训练数据
- 过滤机制：基于参考语料筛选
Common Crawl
- 去重：文档级模糊去重，防止测试集污染
- 加权采样：高质量数据（如Wiki）
被高频采样

Dataset	Quantity (Tokens)	Weight in Mix	Epochs
Common Crawl	410B	60%	0.44
WebText2	19B	22%	2.9
Books1	12B	8%	1.9
Books2	55B	8%	0.43
Wikipedia	3B	3%	3.4

GPT-3 模型评估

The three settings we explore for in-context learning

Zero-shot

The model predicts the answer given only a natural language description of the task. No gradient updates are performed.



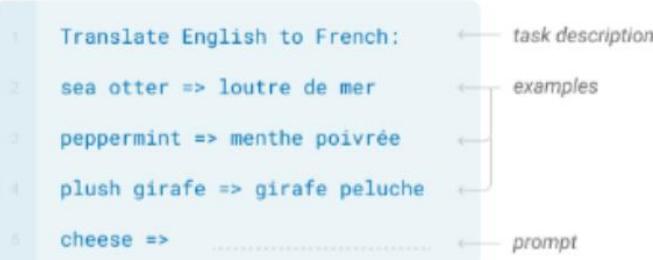
One-shot

In addition to the task description, the model sees a single example of the task. No gradient updates are performed.



Few-shot

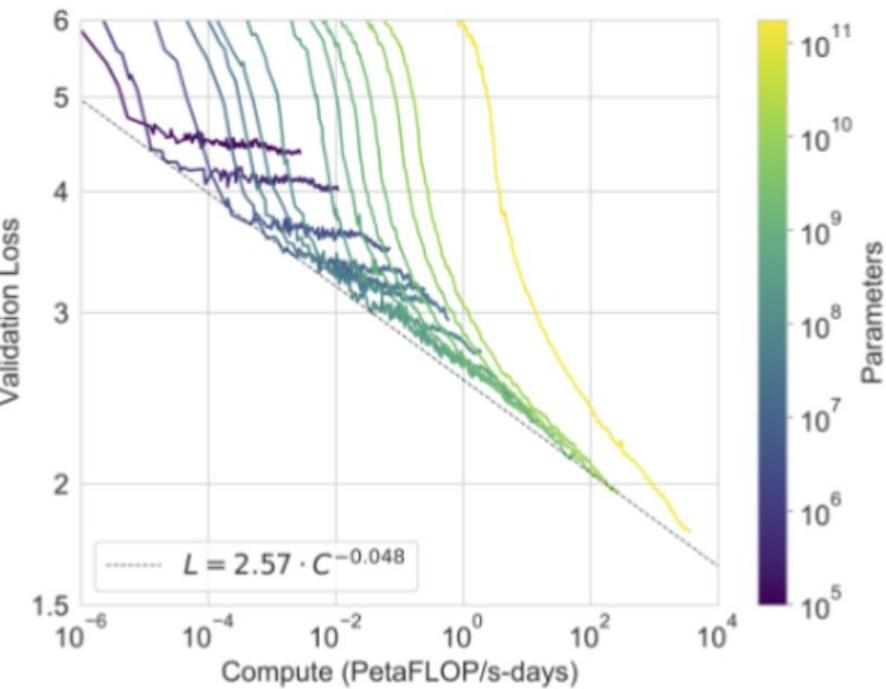
In addition to the task description, the model sees a few examples of the task. No gradient updates are performed.



Traditional fine-tuning (not used for GPT-3)

Fine-tuning

The model is trained via repeated gradient updates using a large corpus of example tasks.



与 GPT-2 的区别

整体来看，GPT-3 相比于 GPT-2 有如下几点区别：

1. 效果上，超出 GPT-2 非常多，能生成人类难以区分的新闻文章；
2. 主推 few-shot，相比于 GPT-2 的 zero-shot，具有很强的创新性；
3. 模型结构略微变化，采用 sparse attention 模块；
4. 海量训练语料 45TB（清洗后 570GB），相比于 GPT-2 的 40GB；
5. 海量模型参数，最大模型为 1750 亿，GPT-2 最大为 15 亿参数；

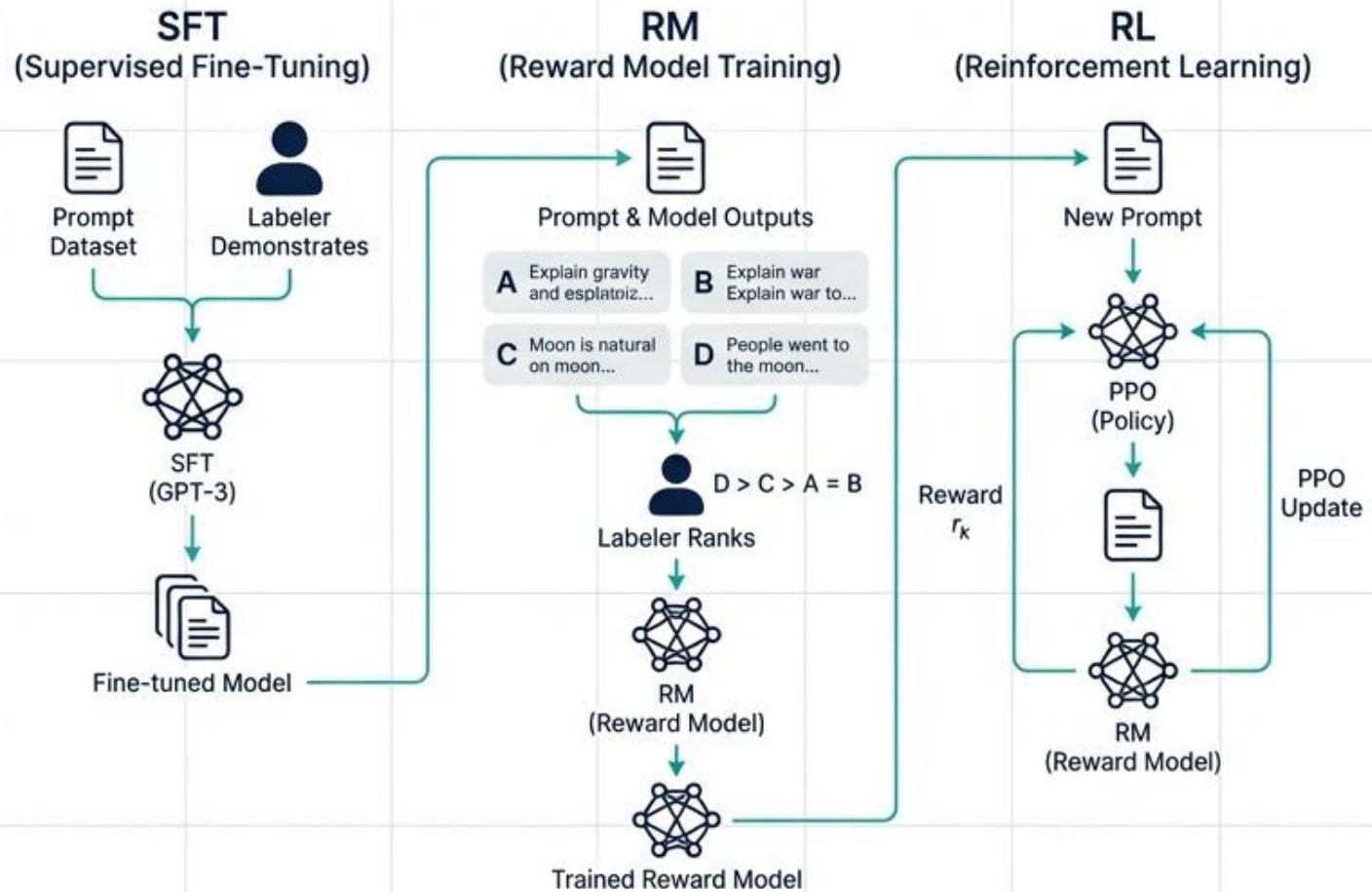
InstructGPT：模型与人类意图对齐

- **核心痛点：**GPT-3存在幻觉、偏见、不遵循指令
- **目标定义：**Helpful (有用) 、 Honest (诚实) 、 Harmless (无害)
- **指令微调：**消除“预测下个词”与“遵循指令”的Gap
- **局限性：**对“偏见”改善不明显，但安全性提升



InstructGPT: RLHF 三阶段训练法

- Step 1 SFT: 人工撰写答案，监督微调GPT-3
- Step 2 RM: 人工对输出排序，训练奖励模型
- Step 3 PPO: 利用RM反馈，通过PPO优化策略
- 核心思想：将不可微的人类偏好转化为数学目标



InstructGPT：奖励模型(RM)的训练细节

- 排序机制: D > C > A = B,
优于直接打分
- 损失函数: 最大化好回答
(y_w)与坏回答(y_l)的分差
- 数据来源: 消除分布Gap,
使用真实用户Prompt
- 结果: 1.3B InstructGPT >
175B GPT-3 (人类评估)

$$\text{loss}(\theta) =$$

$$-\mathbb{E} \left[\log(\sigma(r_\theta(x, y_w) - r_\theta(x, y_l))) \right]$$

x = Prompt

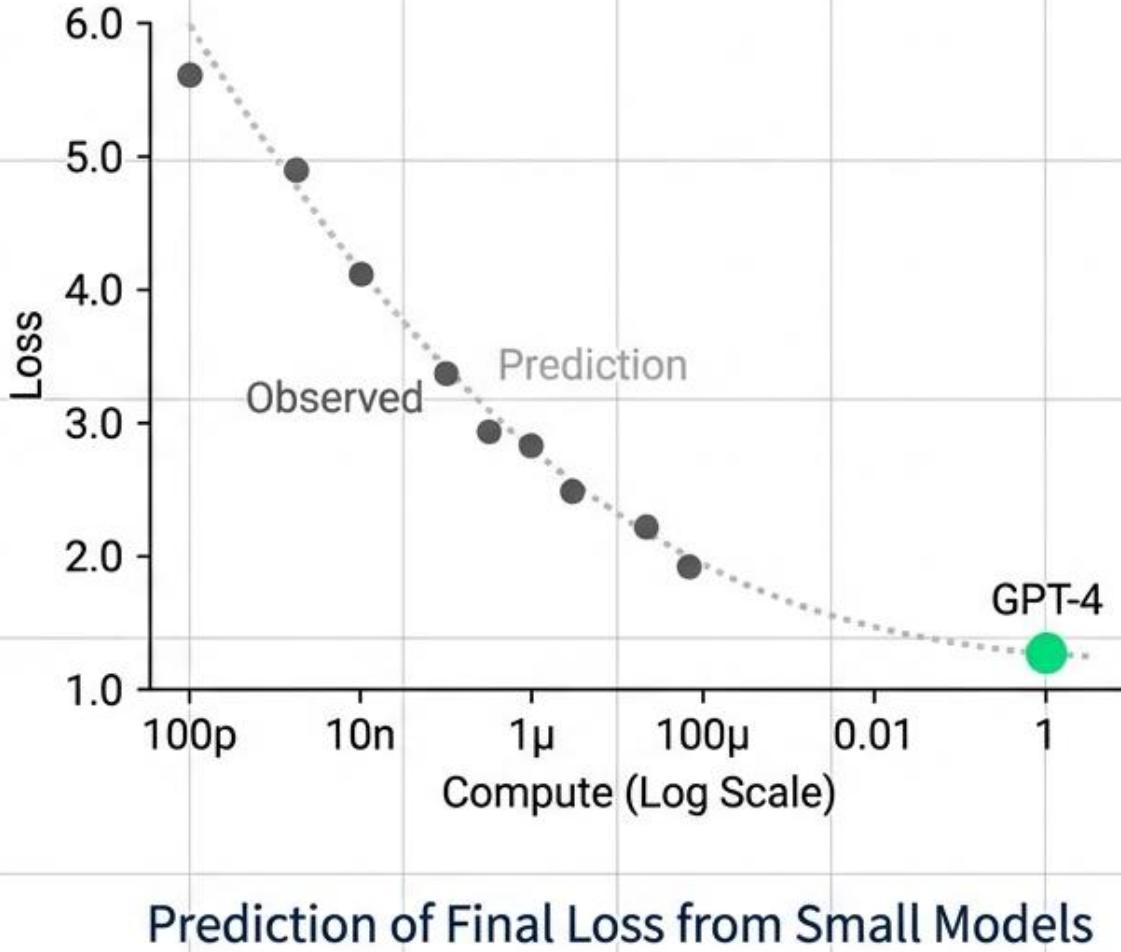
y_w = Winning Response

y_l = Losing Response

r_θ = Reward Model Score

GPT-4：可预测的扩展 (Predictable Scaling)

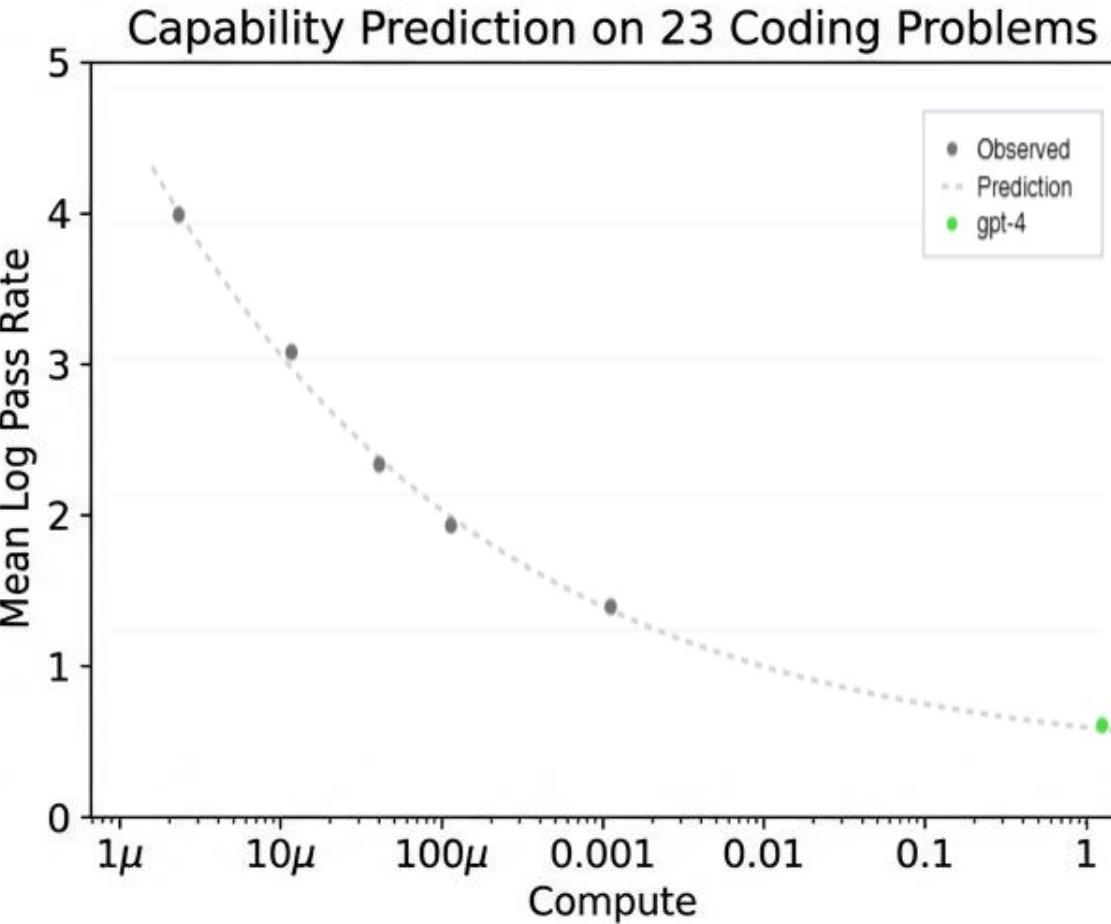
- 工程重点：构建可预测的深度学习堆栈
- Loss预测：用1/10,000计算量预测最终Loss
- 拟合公式： $L(C) = aC^b + c$
- 意义：降低训练高昂模型时的试错风险



GPT-4：具体任务能力的预测

- 挑战：将Loss预测转化为具体任务指标
- 实验：HumanEval Python编程任务
- 方法：按难度分类，拟合Pass Rate与Compute关系
- 结论：准确预测了GPT-4的代码生成能力

$$-\mathbb{E}_P[\log(\text{pass_rate}(C))] = \alpha * C^{-k}$$



Capability Prediction on 23 Coding Problems

总结：从PLM到LLM的演进逻辑

GPT-2	Scaling Laws	GPT-3	InstructGPT/GPT-4
 无监督多任务 (Zero-shot) + 架构深度优化	 规模 (N, D, C) 决定性能上限 + 涌现现象	 情境学习 (In-Context Learning) + 大数据工程	 人类对齐 (RLHF) + 可预测工程 (Predictable Scaling)

LLM的基础架构及优化

从Transformer范式到稀疏混合专家模型的演进路径



架构演化：从统计
模型到通用求解器



组件解析：分词、
位置编码与注意力



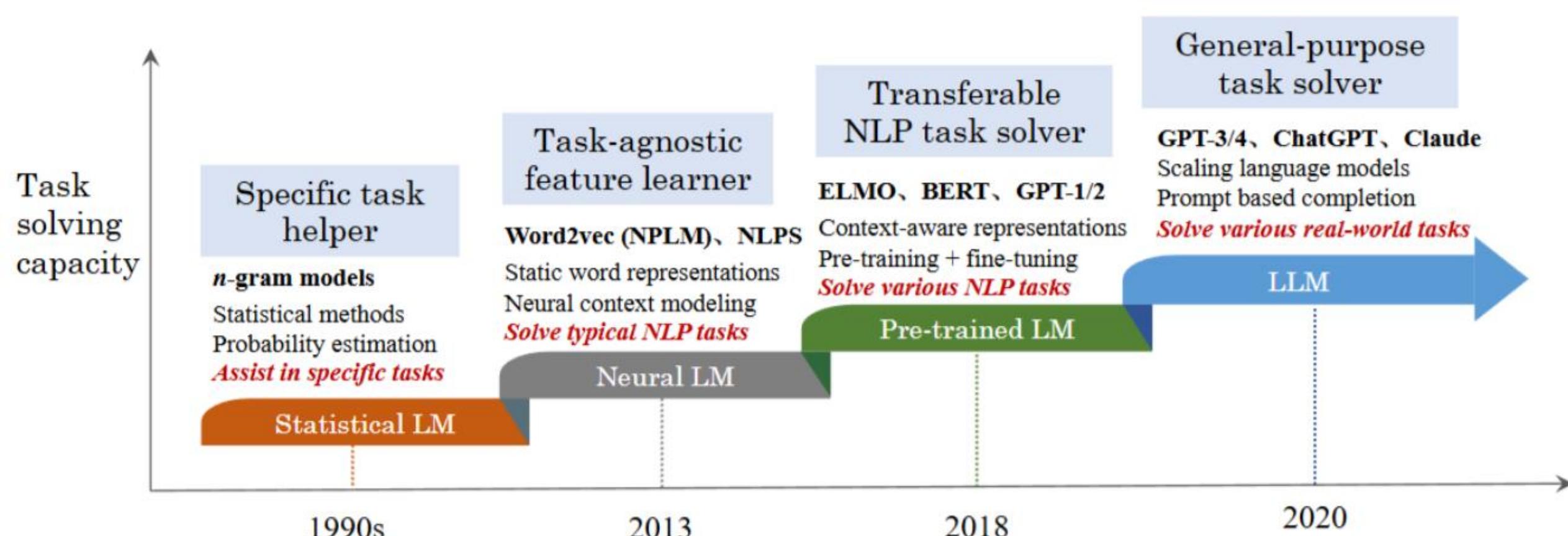
模型族系：Llama，
Qwen 与
DeepSeek



前沿趋势：MoE架
构与推理策略

范式演化：从统计语言模型到通用任务求解器

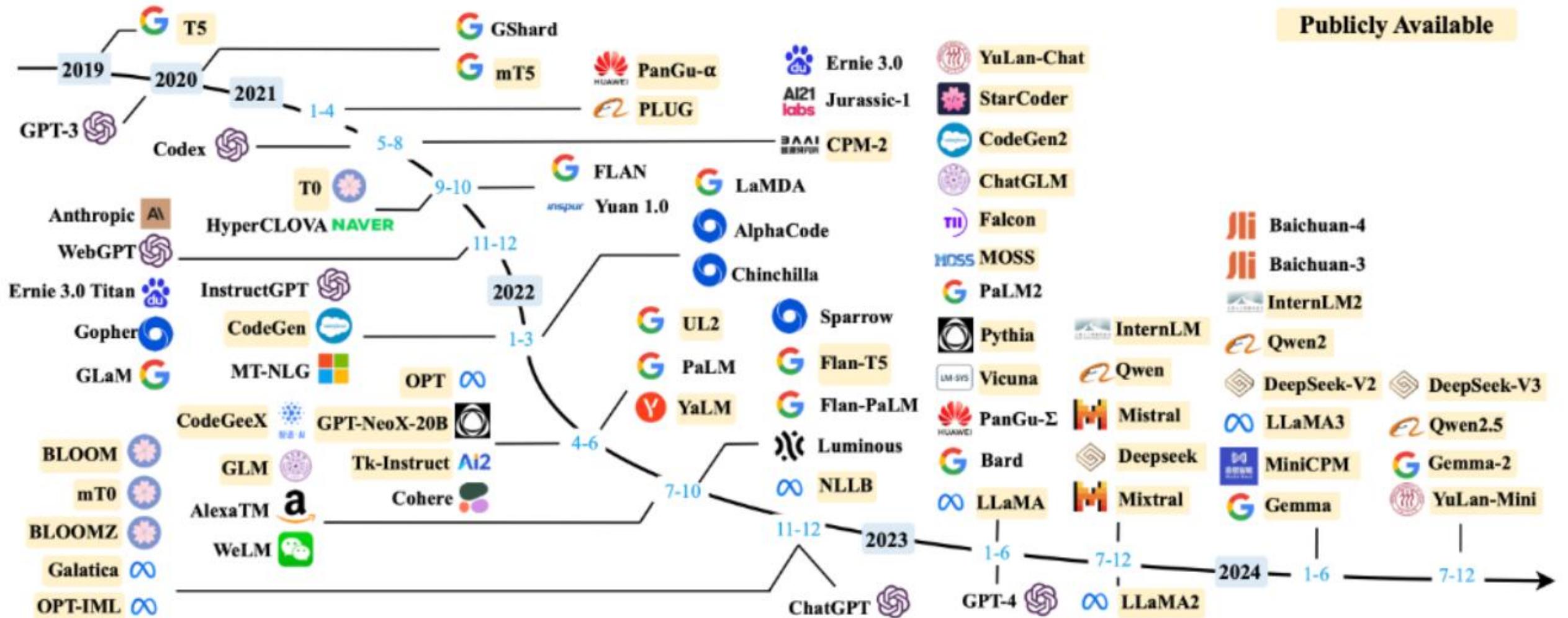
模型范式从专用任务辅助逐步演化为基于Prompt的通用任务求解器。



- N-gram: 统计概率预测
- Neural LM: 静态词向量表征

- Pre-trained LM: 通用表征 + 微调
- BERT/GPT-1/2: 解决特定NLP任务

- LLM: GPT-3/4, Scaling Laws
- 核心转变: 权重微调 → In-context Learning



Tokenization——LLM inputs

分词的目的：将输入文本切分为 token，每个token用token id表示，作为 embedding 与后续模型的输入

Tokenization :(

Tokenization is at the heart of much weirdness of LLMs. Do not brush it off.

- Why can't LLM spell words? **Tokenization**.
- Why can't LLM do super simple string processing tasks like reversing a string? **Tokenization**.
- Why is LLM worse at non-English languages (e.g. Japanese)? **Tokenization**.
- Why is LLM bad at simple arithmetic? **Tokenization**.
- Why did GPT-2 have more than necessary trouble coding in Python? **Tokenization**.
- Why did my LLM abruptly halt when it sees the string "<|endoftext|>"? **Tokenization**.
- What is this weird warning I get about a "trailing whitespace"? **Tokenization**.
- Why does the LLM break if I ask it about "SolidGoldMagikarp"? **Tokenization**.
- Why should I prefer to use YAML over JSON with LLMs? **Tokenization**.
- Why is LLM not actually end-to-end language modeling? **Tokenization**.
- What is the real root of suffering? **Tokenization**.

输入层解析：分词粒度与策略权衡

Subword粒度在词表规模、覆盖率与泛化能力之间取得了最佳平衡。

粒度对比 (Granularity Comparison)

粒度	优点 (Pros)	缺点 (Cons)
Word (词)	语义直观	词表巨大, OOV严重
Char (字符)	无OOV	序列过长, 语义破碎
Subword (子词)	平衡效率与语义	需预处理

Token count	got2
502	502
Large Language models (LLMs) are deep neural network s odcls that have been developed in recent years, such as the ChatGPT model provided by OpenAI. They ushered i n a new era of natural language processing(NLP).	Large Language models (LLMs) are deep neural network s odcls that have been developed in recent years, such as the ChatGPT model provided by OpenAI. They ushered i n a new era of natural language processing(NLP).
456 + 1234 = 1690	456 + 1234 = 1690
<pre>class SimpleTokenizerV2: def __init__(self, vocab): self.str_to_int = vocab self.int_to_str = {int(s): i for s, i in vocab.items() } def encode(self, text): preprocessed = re.split(r'[.,!?"\'] [\t\n\r]+ \s+ \s+', text) preprocessed = [item.strip() for item in preprocessed if len(item.strip())] preprocessed = [item for item in self.str_to_int if item in self.str_to_int] else "UNK" if item not in preprocessed else " ids = [self.str_to_int[i] for i in preprocessed] return ids def decode(self, ids): text = " ".join([self.int_to_str[i] for i in i ds]) text = re.sub(r'(\s+ [.,!?"\'])\s+', r'\1', tex t) return text</pre>	<pre>class SimpleTokenizerV2: def __init__(self, vocab): self.str_to_int = vocab self.int_to_str = {int(s): i for s, i in vocab.items() } def encode(self, text): preprocessed = re.split(r'[.,!?"\'] [\t\n\r]+ \s+ \s+', text) preprocessed = [item.strip() for item in preprocessed if len(item.strip())] preprocessed = [item for item in self.str_to_int if item in self.str_to_int] else "UNK" if item not in preprocessed else " ids = [self.str_to_int[i] for i in preprocessed] return ids def decode(self, ids): text = " ".join([self.int_to_str[i] for i in i ds]) text = re.sub(r'(\s+ [.,!?"\'])\s+', r'\1', tex t) return text</pre>

<https://tiktokenizer.vercel.app/>

结论：现代LLM普遍采用Subword，常见词保留整体，罕见词拆分。

主流分词算法：BPE，WordPiece 与 Unigram

三种主流Subword算法在合并策略与优化目标上存在本质差异。

BPE (Byte Pair Encoding)

基于频次统计

- 迭代合并相邻高频字符对
- 简单高效，结果稳定
- GPT系列，Llama

WordPiece

基于似然概率

- 合并能最大化训练数据似然的对
- 考虑互信息，贴合语言模型
- BERT

Unigram LM

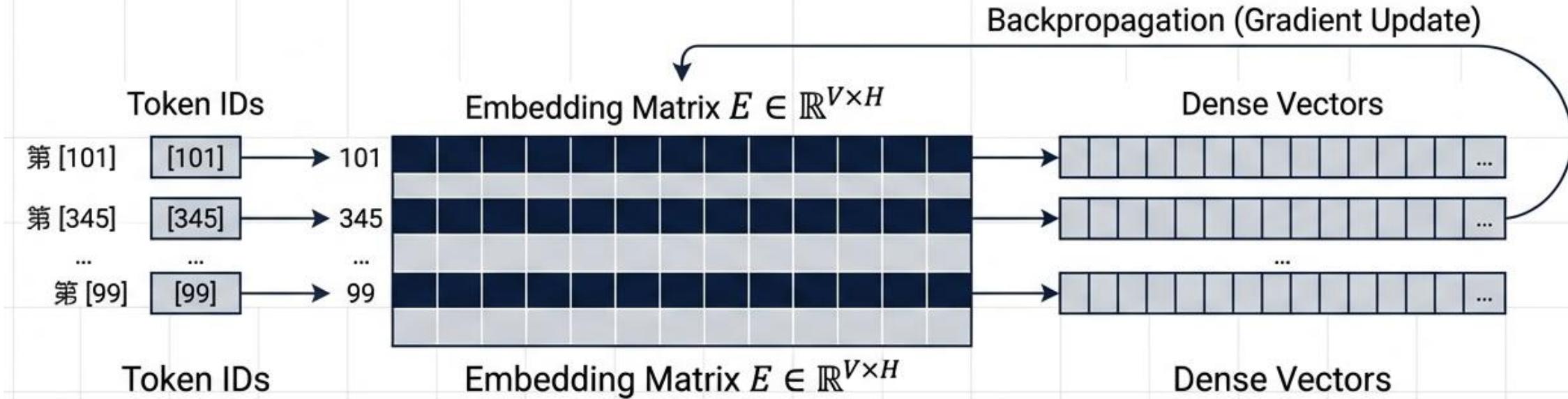
基于概率剪枝

- 从大词表逐步剪枝低概率子词
- 减法思维，支持概率化切分
- T5，ALBERT

核心差异：BPE追求高频合并（简单）；WordPiece追求信息增益（精准）；Unigram追求概率最优（灵活）。

Embedding层：参数化映射与训练机制

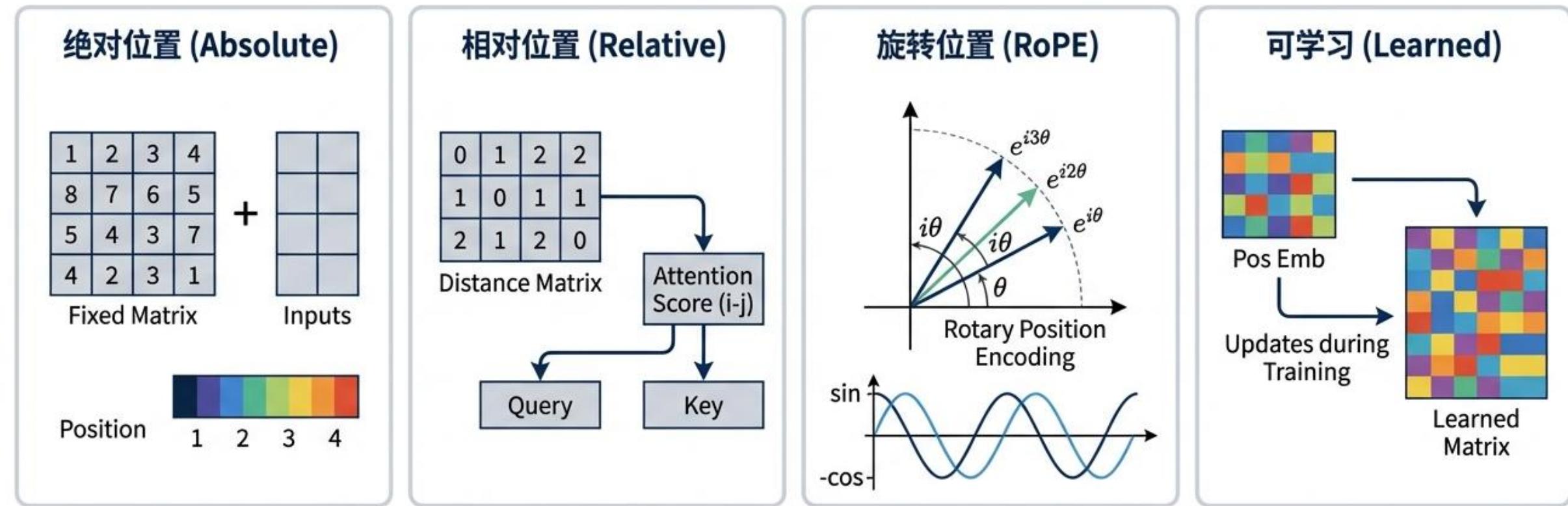
Embedding层作为可学习矩阵，常与输出层共享参数以提升效率。



- **实现机制:** Lookup查表，前向取行，反向更新
- **初始化:** 通常用小方差正态分布（如 $\mathcal{N}(0, 0.02)$ ）
- **Tied Embeddings:** 输入Embedding与输出LM Head共享参数 ($W_{\text{out}} = E^T$)
- **优势与权衡:** 共享节省显存(V 很大时显著)，独立提升灵活性(Llama部分版本)

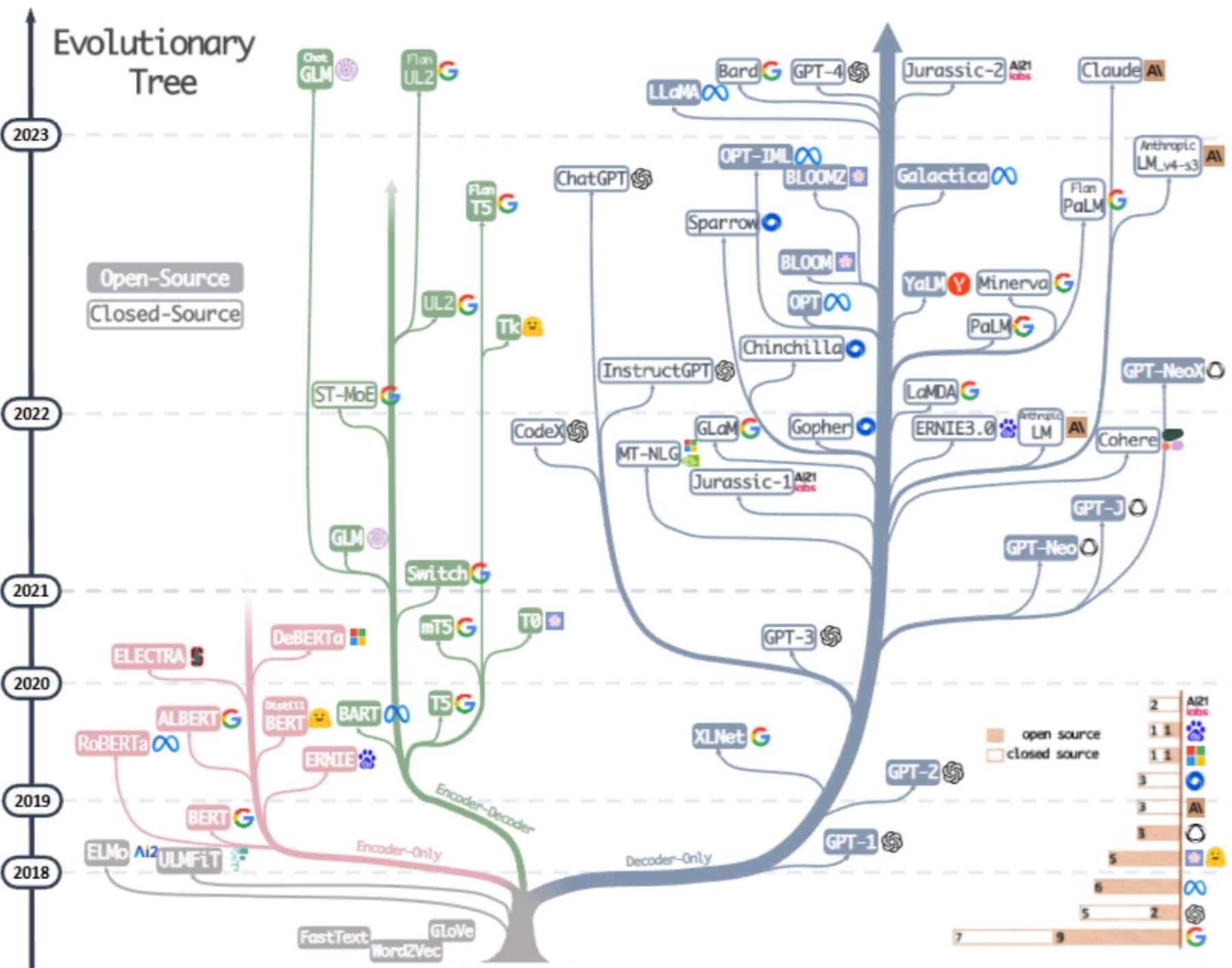
位置编码演化：从绝对位置到旋转位置(RoPE)

位置编码技术已从绝对位置正弦演进为不仅能处理长序列且具有相对位置特性的RoPE。



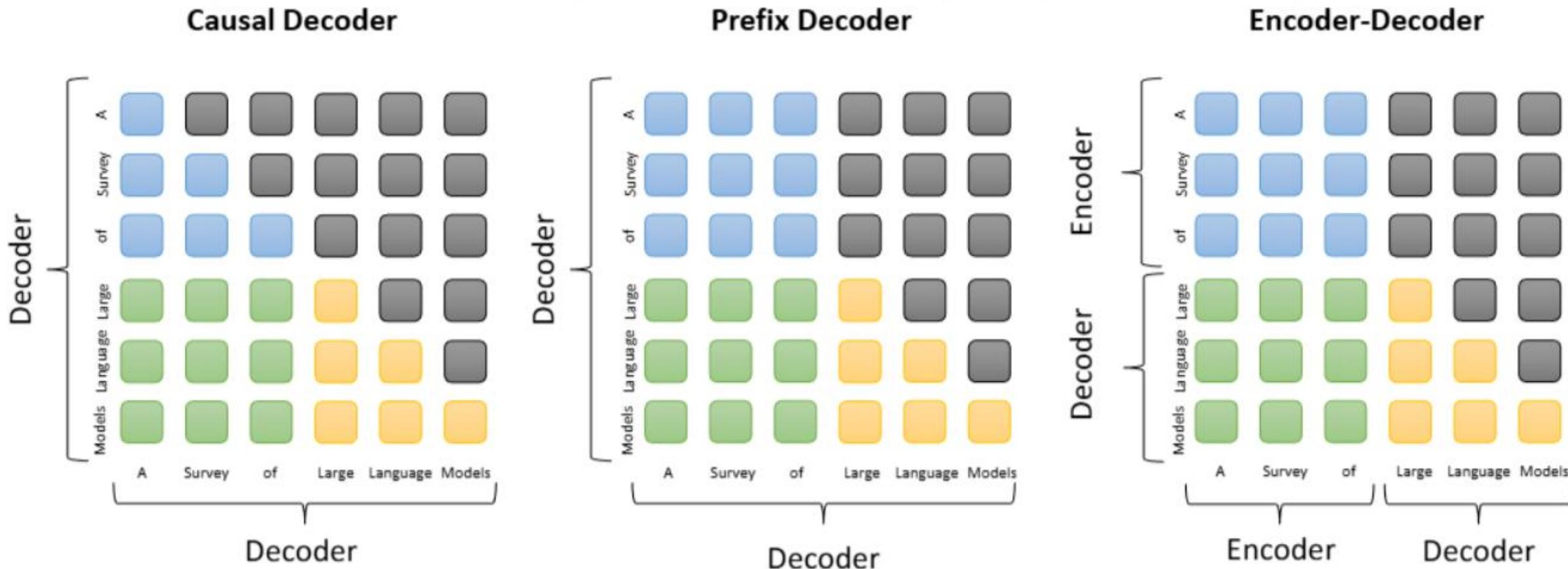
- 正弦绝对PE：无参数，缺乏深度交互
- 可学习PE：适应性强，但受限于最大长度
- RoPE (Llama标配)：通过旋转变换注入相对位置， $\text{Attention}(Q, K) = \text{Softmax}\left(\frac{QK^T}{\sqrt{d_k}} + P\right)$
- 优势：无最大长度限制，外推性极佳

Evolutionary Tree



宏观架构概览：Mask决定模型类型

Attention Mask的设计决定了模型是Encoder、Decoder还是前缀模型。



- **Encoder-only:** 双向可见，适合理解 (BERT)
- **Decoder-only:** 因果Mask，严格预测Next Token (GPT/Llama)
- **Prefix LM:** 前缀部分双向，生成部分单向 (UniLM)
- **Enc-Dec:** 独立编码器与解码器，通过Cross-Attention连接 (T5)

模型演化案例 A：Llama系列（1-4）

Llama系列确立了开源基座标准，并逐步引入GQA、MoE与原生多模态。



关键趋势：上下文扩展（2K→256K） | 架构稀疏化（Dense→MoE） | 多模态融合

代际	预训练数据规模	上下文长度 (预训练/推理支持)	Tokenizer	相对上一代的关键改动	代表模型 (base / chat-instruct)
LLaMA 1	7B/13B: 1.0T tokens; 33B/65B: 1.4T tokens (ar5iv)	2K (2048) (ar5iv)	SentencePiece BPE, 32K vocab (ar5iv)	作为基线: decoder-only Transformer (预归一化 RMSNorm、SwiGLU、RoPE 等) (ar5iv)	LLaMA-7B/13B/33B/65B (预训练基座)
Llama 2	2.0T tokens; 数据混合更新, 整体 token 数比 LLaMA1 +40% (ar5iv)	4K (相对 LLaMA1 翻倍) (ar5iv)	同 LLaMA1: SentencePiece BPE, 32K vocab (ar5iv)	更长上下文 (2K→4K) ; 大模型引入 GQA (提升推理可扩展性) ; 更强数据清洗/配比推出 Llama 2-Chat (SFT+RLHF) (ar5iv)	Llama-2 / Llama-2-Chat (ar5iv)
Llama 3	15T+ tokens (公开在线数据新配比) (Hugging Face)	8K (Hugging Face)	新 tokenizer: 128,256 vocab (从 32K 扩到 128K) (Hugging Face)	训练数据量大幅提升; GQA 在 8B/70B 都使用; 更大词表带来更高 token 效率与多语潜力 (也导致小模型参数从 7B→8B) (Hugging Face)	Llama-3 / Llama-3-Instruct (Hugging Face)
Llama 4	最高 40T tokens, 预训练 256K; Instruct: 覆盖 200 种语言 (Hugging Face)	预训练 256K; Instruct: Scout 10M、Maverick 1M (Hugging Face)	(文章未在同一段落给出词表数字; 重点升级在 MoE + 超长上下文 + 多模态) (Hugging Face)	从 Dense → MoE; 原生多模态 (early fusion, 支持文+图) ; 为超长上下文引入 NoPE/iRoPE、chunked attention 等设计 (Hugging Face)	Llama-4-Scout / Llama-4-Maverick (base / instruct) (Hugging Face)

模型演化案例 B: Qwen系列

Qwen系列强调多语言（中英）能力，并在后续版本全面拥抱MoE与长窗口。

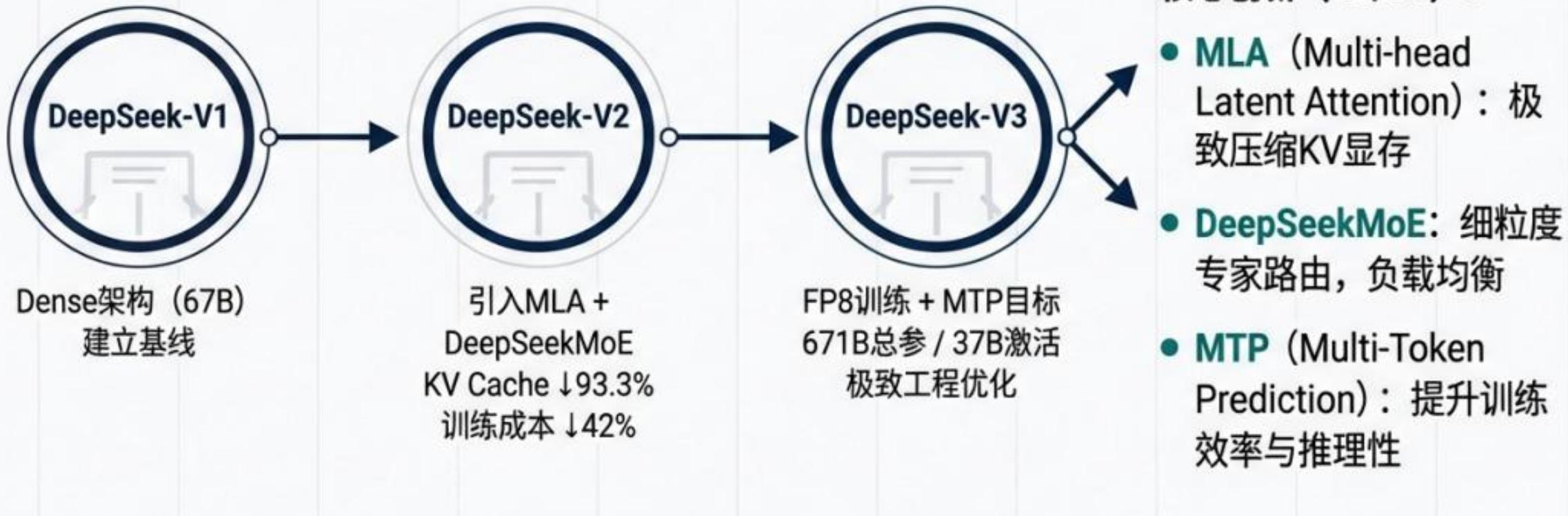
特性 (Feature)	Qwen 1	Qwen 2	Qwen 3
多语言 (Languages)	专注中英	约30种	119种 (多方言)
架构 (Architecture)	Dense Transformer	Dense + MoE (57B)	细粒度 MoE (Fine-grained)
上下文 (Context)	8K/32K	32K (预训练) -> 131K	128K (全系标配)
推理模式 (Inference)	Tool-use	SFT+DPO	Thinking Mode Budget (动态计算)

- **Qwen3亮点：**在统一框架内集成了类似R1的思维模式 (Thinking Mode)
- **MoE策略：**从Qwen2开始引入，Qwen3采用更细粒度专家切分

维度	Qwen1	Qwen2	Qwen3
代表性开源模型	1.8B / 7B / 14B / 72B (含 Chat 版本) (GitHub)	0.5B / 1.5B / 7B / 72B + 57B-A14B (MoE) (ar5iv)	Dense: 0.6B–32B; MoE: 30B-A3B、235B-A22B(arXiv)
训练数据量级	“最多到 3T tokens”的多语料 (偏中英) (GitHub)	预训练数据 “over 7T tokens” (ar5iv)	约 36T tokens (并显著扩充数据合成与标注体系) (ar5iv)
多语言覆盖	重点覆盖中文/英文 (官方表述强调中英与多领域覆盖) (GitHub)	约 30 种语言(ar5iv)	从 29 扩到 119 种语言与方言(arXiv)
长上下文能力	开源模型卡/仓库给出最大长度多款到 32K (如 1.8B/7B/72B) 14B 为 8K(GitHub)	预训练阶段将上下文从 4,096 提到 32,768；并通过 YARN+DCA 推到推理时可处理至 131,072(ar5iv)	多数模型 128K (小模型 32K)；并明确给出各型号上下文长度表(ar5iv)
注意力/推理效率关键改动	基础 Transformer + (强调工具使用/规划能力的 Chat 版本能力路线) (arXiv)	Dense 侧引入 GQA (更省 KV cache、吞吐更高)；长上下文用 DCA + YARN(ar5iv)	在系列层面强化“效率 + 推理”并给出更系统的长上下文与稀疏 MoE 设计(ar5iv)
MoE (稀疏化)	首代本身以 Dense 为主 (后续 1.5 才开始系统化 MoE)	57B-A14B MoE: 64 routed experts、每 token 激活 8，并有 shared experts(ar5iv)	MoE: 总 128 experts、每 token 激活 8，取消 shared experts，并引入更细粒度专家切分与负载均衡损失(ar5iv)
对齐/推理模式	Chat 侧强调 tool-use / planning (可做 agent/代码解释器类任务) (arXiv)	后训练: SFT + DPO (并包含 RLHF 章节)，面向 chat/agent 用途(ar5iv)	统一框架内集成 thinking mode 与 non-thinking mode，并提供 thinking budget 机制 (在同一模型内动态权衡延迟/效果) (arXiv)

模型演化案例 C: DeepSeek系列

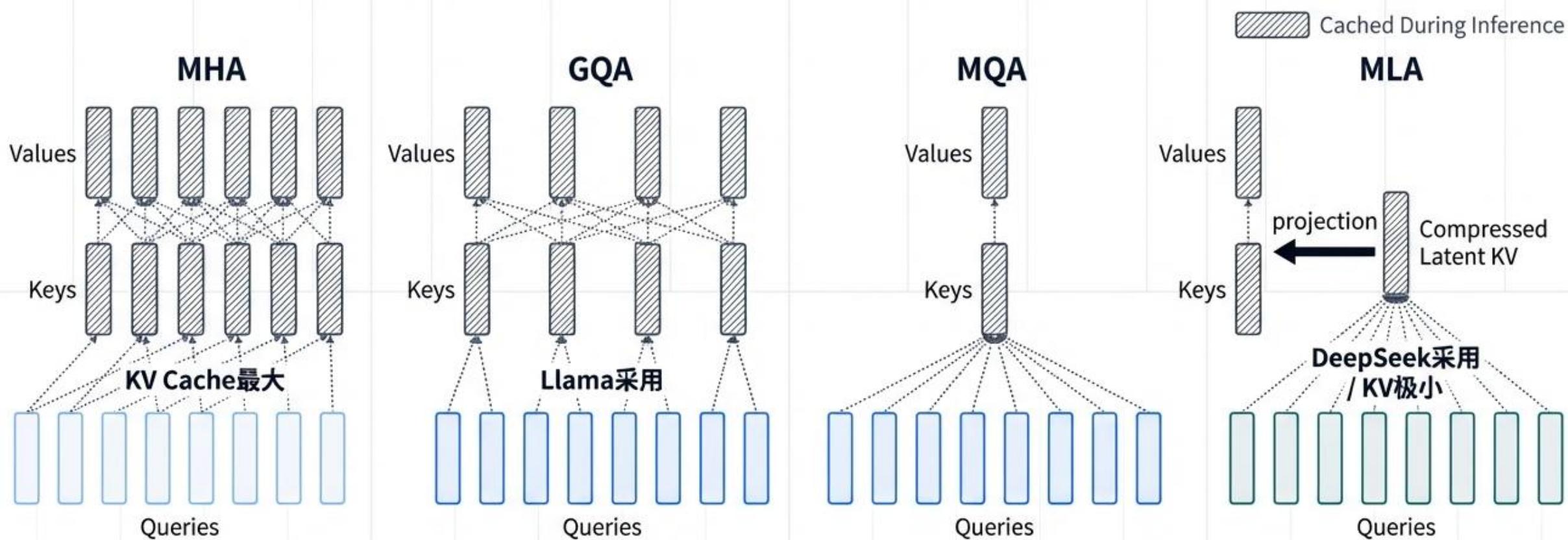
DeepSeek通过MLA与极致的MoE架构优化，显著降低了训练与推理成本。



版本	关键改动 (相对上一代)	架构/训练要点 (关键参数)	典型收益 (能力/成本/推理)
V1	作为基线：从零预训练 + Chat 对齐 (SFT、DPO)	Dense Transformer (整体设计跟 LLaMA 系一致)，预训练数据 2T tokens；上下文 4K；67B 版本用 GQA 降推理成本 (arXiv)	建立后续迭代基线：在 code/math/reasoning 等评测上对标同级开源模型，并通过 DPO 改善对齐 (arXiv)
V2	从 Dense → MoE；引入 MLA (KV 压缩) 与 DeepSeekMoE (稀疏计算)	236B 总参 / 21B 激活；上下文 128K；预训练 8.1T tokens；后训练含 SFT + RL (arXiv)	重点是“更省更快”：相对 67B 基线，训练成本 ↓42.5%、KV cache ↓93.3%、吞吐 ↑5.76×，同时能力更强 (arXiv)
V3	在 V2 架构上继续堆性能与训练效率：去辅助损失的负载均衡 + MTP (多 token 预测) 训练目标；并强化训练系统 (如 FP8)	671B 总参 / 37B 激活；预训练 14.8T tokens；全程训练约 2.788M H800 GPU-hours 且训练稳定 (arXiv)；模型上下文 128K (GitHub)	更强的综合能力 (并保持 MoE 推理效率)；MTP 还可用于推理加速 (如 speculative decoding) (GitHub)
R1	从“通用模型”转向“推理优先”的后训练：先做 R1-Zero (不经过 SFT，直接 RL) 探索推理，再用多阶段训练解决可读性/语言混杂等问题 (rejection sampling + RL + SFT)	明确写到：在 DeepSeek-V3-Base 上，用 GRPO 做 RL；奖励主要基于最终答案正确性；R1-Zero 会自发产生更长的推理、反思与多路径探索，但可读性与语言混杂有问题 (arXiv)	推理能力显著增强 (长 CoT、反思、验证等行为被强化)，并通过后续对齐让它更“可用” (写作/开放域等更稳) (arXiv)

注意力机制改进：MHA, GQA 与 MLA

从MHA到MLA，核心驱动力是在保持性能的同时极致压缩KV Cache。



- **MHA:** 独立KV，显存占用大，效果最强

- **MQA:** 共享一组KV，推理快，效果微损

- **GQA:** 分组共享KV，平衡速度与质量 (Llama常用)

- **MLA:** 低维Latent压缩KV，极致省显存 (DeepSeek)

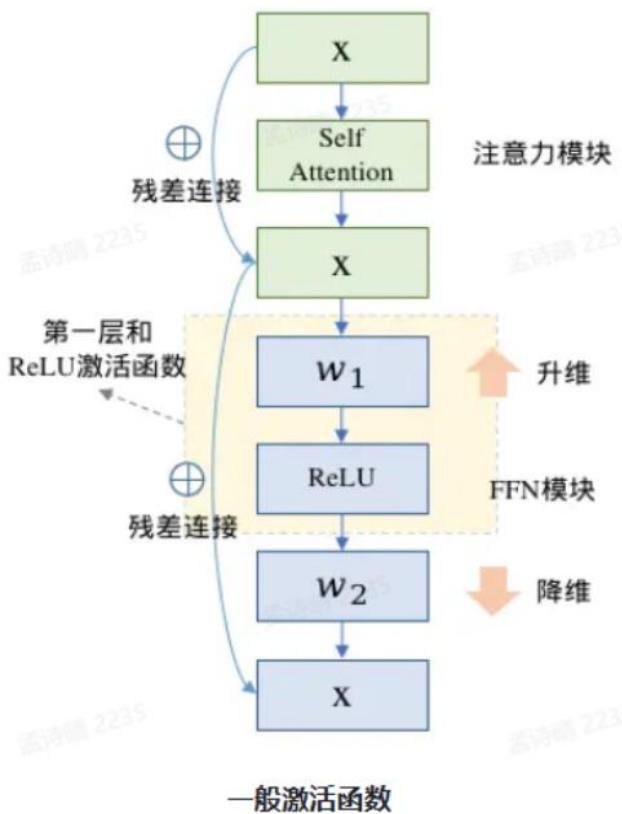
FFN层改进：SwiGLU激活函数

$$FFN(x) = \text{ReLU}(xW_1 + b_1)W_2 + b_2$$

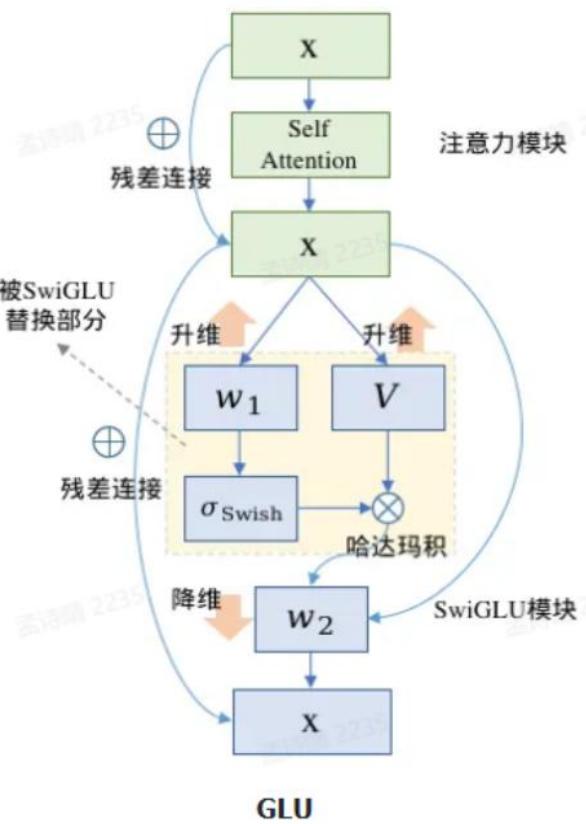
$$GLU(x) = xV \cdot \sigma(xW + b) \quad FFN_{GLU} = (xV \cdot \sigma(xW_1 + b))W_2$$

SwiGLU通过引入门控机制增加了非线性表达能力，成为大模型标配。

Standard FFN (标准FFN)



GLU/SwiGLU (门控线性单元)



C

SwiGLU Mechanism

$$FFN_{SwiGLU}(x) = (\text{Swish}(xW) \odot xV)W_2$$

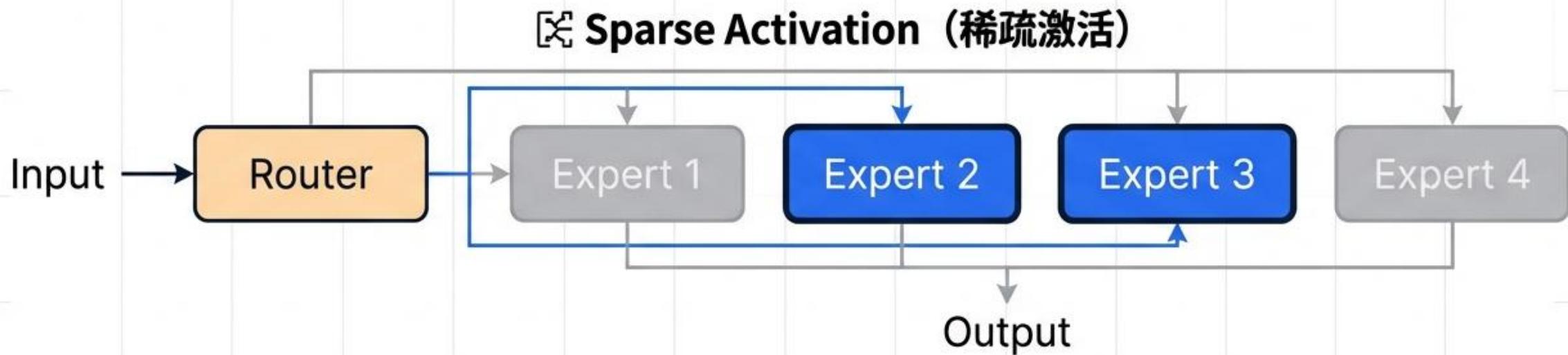
- **结构变化:** 增加门控路径 (Gating)，参数量通常通过缩小中间维度平衡 (如 2/3)
- **优势:** 增强表达能力，更有效地筛选信息流

$$\text{Swish}(x) = x \times \text{sigmoid}(\beta * x)$$

$$\text{GeLU}(x) \approx 0.5x \left(1 + \tanh \left(\sqrt{\frac{2}{\pi}} (x + 0.044715x^3) \right) \right)$$

MoE (混合专家模型) 基础架构

MoE通过Router动态选择专家，在扩大总参数量的同时保持低推理成本。



C

- 核心组件: Router (门控网络) + 多个 Expert (FFN)
- 稀疏激活: 每Token仅激活Top-K个专家 (如 DeepSeek V3: 激活37B/总671B)
- 优势: 极大提升参数容量 (Knowledge Capacity) , 推理FLOPs不增
- 挑战: 训练稳定性与负载均衡 (Load Balancing)

MoE专家的功能分化：Encoder vs Decoder

Encoder专家表现出明显的语义/语法分工，而Decoder专家分工相对模糊。

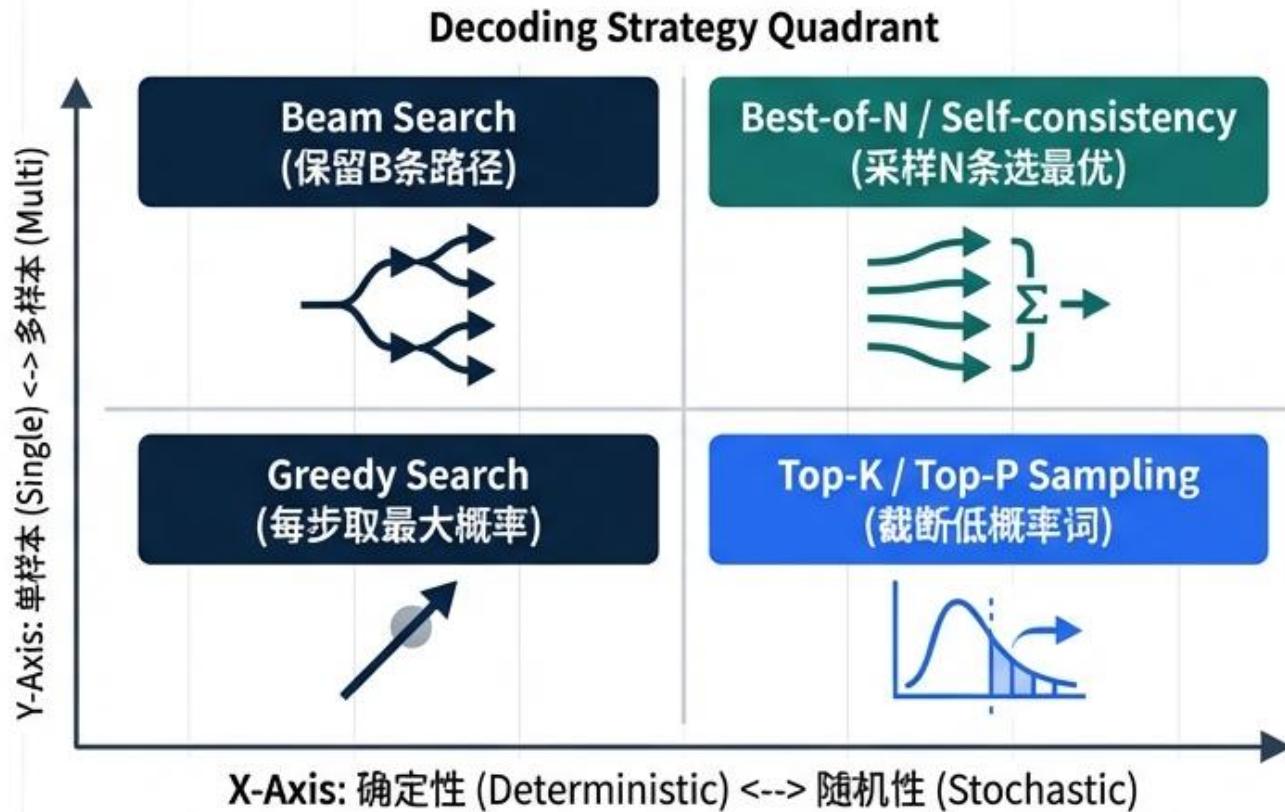
Expert Specialization	Expert Position	Routed Tokens	C 分析
Punctuation	Layer 2, 6	, . ; & ?	<ul style="list-style-type: none">Encoder表现：专家分化明显，各司其职（动词、名词、符号）
Verbs	Layer 1	said, read, miss	<ul style="list-style-type: none">Decoder表现：语义分工较少，路由趋于均匀
Visual Desc	Layer 0	blue, yellow, dark	<ul style="list-style-type: none">Decoder逻辑：倾向于按句法结构或上下文相似性聚类（如 Python `self` 与 English `Question` 同路）
Proper Names	Layer 1	Mart, Ken, Sam	

[1] ST-MoE: Designing Stable and Transferable Sparse Expert Models

[2] Mixtral of Experts

解码与采样策略（一）：确定性与随机性

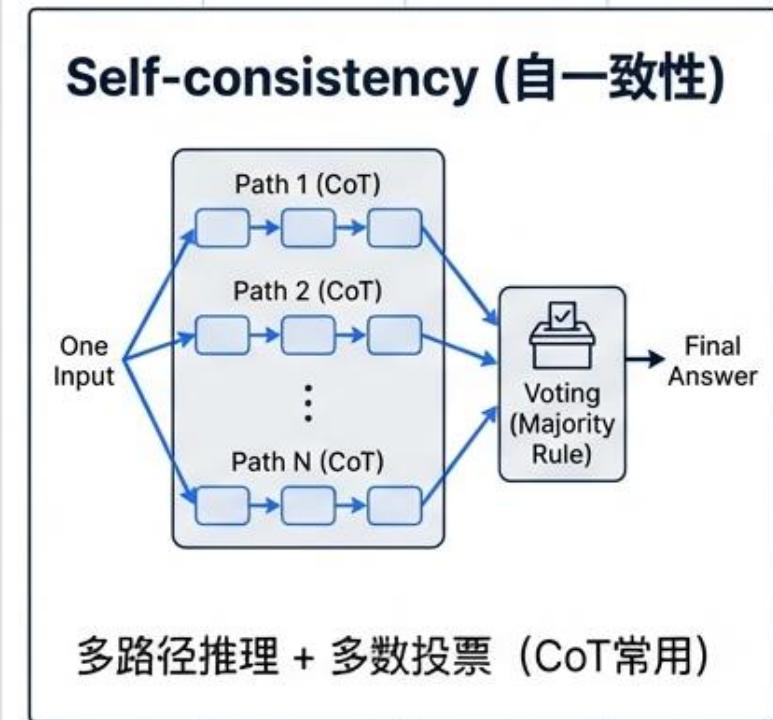
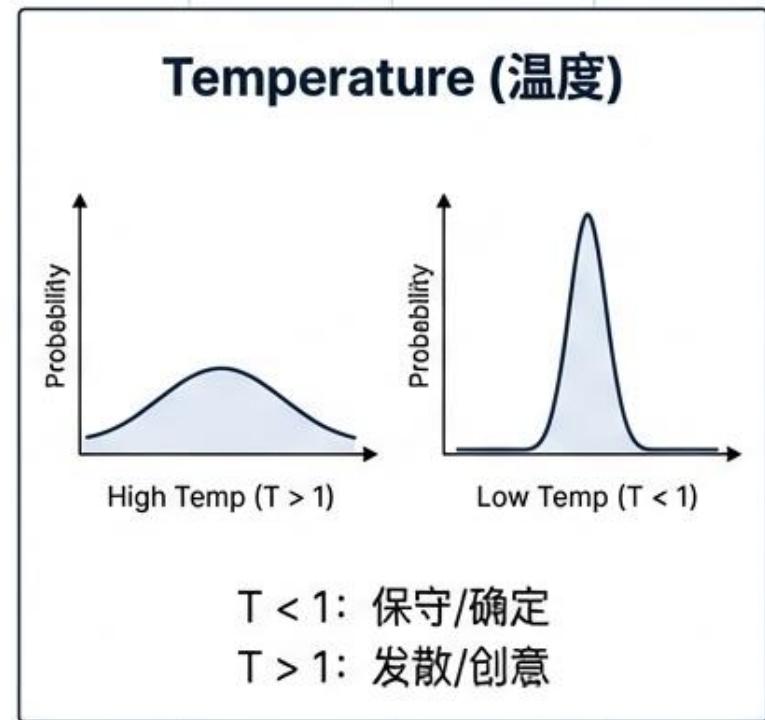
解码策略决定了生成的确定性与多样性，需根据任务场景权衡。



- **Greedy:** 稳定但易局部最优（适合数学/翻译）
- **Top-P (Nucleus):** 动态截断尾部概率，平衡逻辑与创意（最常用）
- **Beam Search:** 搜索空间更大，适合高质量短生成

解码与采样策略（二）：高级策略

温度采样调节分布平滑度，而Best-of-N与自一致性通过“多选一”提升效果。



应用场景：对于DeepSeek-R1等推理模型，Self-consistency与Best-of-N是提升数学/逻辑任务准确率的关键。