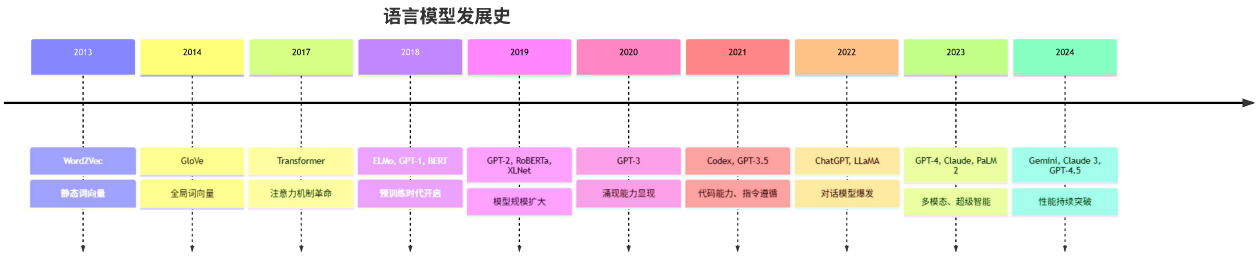
# 第3章 预训练语言模型发展史

> 从词向量到千亿参数大模型，AI语言理解的进化之路

## 3.1 发展时间线

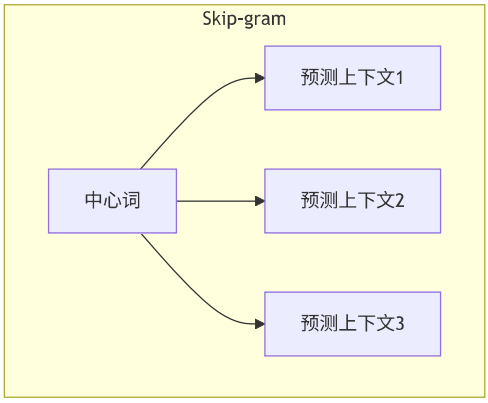


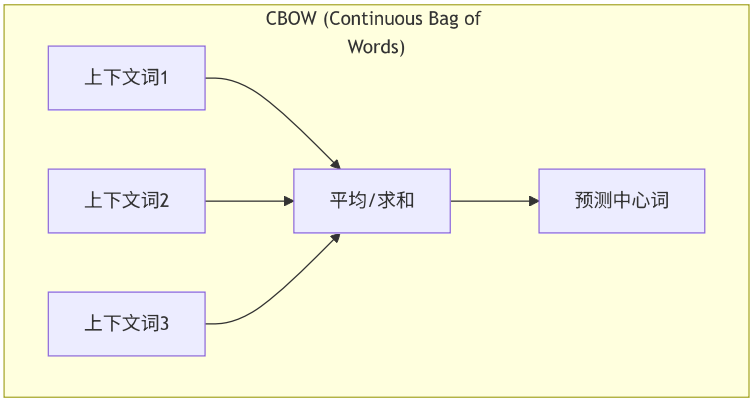

## 3.2 词向量时代（2013-2017）

### 3.2.1 Word2Vec（2013）

**核心思想：** "You shall know a word by the company it keeps"（根据上下文理解词义）

**两种架构：**




**简化实现：**

```
import numpy as np

class Word2Vec:
    def __init__(self, vocab_size, embedding_dim=100):
        # 词嵌入矩阵
        self.W1 = np.random.randn(vocab_size, embedding_dim) * 0.01
        # 输出矩阵
        self.W2 = np.random.randn(embedding_dim, vocab_size) * 0.01

    def skip_gram_forward(self, center_word_idx):
        """
```

```
        Skip-gram前向传播
        """
        # 中心词的embedding
        hidden = self.W1[center_word_idx]  # (embedding_dim,)

        # 预测上下文词的得分
        scores = np.dot(hidden, self.W2)  # (vocab_size,)

        # Softmax
        probs = np.exp(scores) / np.sum(np.exp(scores))

        return hidden, probs
```

**创新点：**

- [x] 从one-hot到稠密向量
- [x] 捕捉语义相似性
- [x] 训练高效（负采样优化）

**局限性：**

- [ ] 一词一向量，无法处理多义词
- [ ] 没有考虑词序
- [ ] 无法处理OOV（Out of Vocabulary）

### 3.2.2 GloVe（2014）

**Global Vectors for Word Representation**

结合全局统计信息（共现矩阵）和局部上下文。

**目标函数：**

$$J = \sum_{i,j=1}^{V} f(X_{ij})(w_i^T \tilde{w}_j + b_i + \tilde{b}_j - \log X_{ij})^2$$

其中：

- $X_{ij}$：词$i$和词$j$的共现次数
- $f(X_{ij})$：加权函数

### 3.2.3 词向量的经典应用

**类比推理：**

```
king - man + woman ≈ queen
Paris - France + Germany ≈ Berlin
```

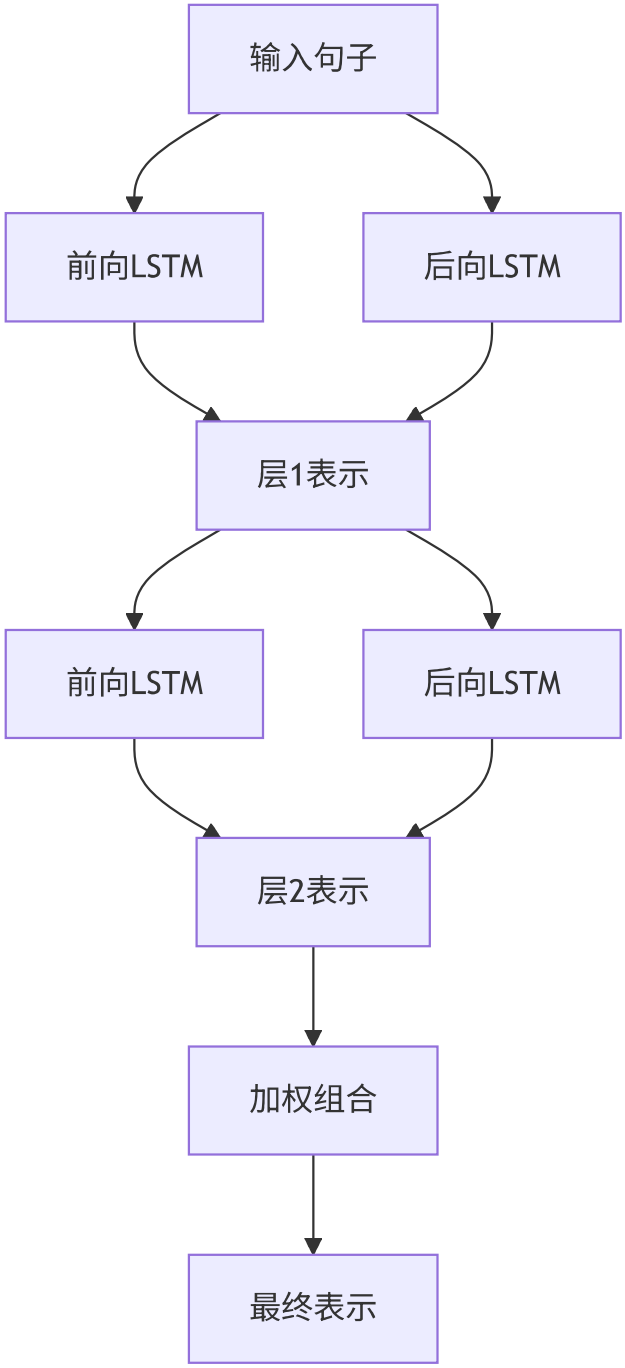
```
def word_analogy(word_vectors, a, b, c):
    """
    解决类比问题：a - b + c ≈ ?
    """
    # 计算目标向量
    target = word_vectors[a] - word_vectors[b] + word_vectors[c]

    # 找最相似的词
    similarities = {}
    for word, vec in word_vectors.items():
        if word not in [a, b, c]:
            sim = cosine_similarity(target, vec)
            similarities[word] = sim

    return max(similarities, key=similarities.get)

def cosine_similarity(v1, v2):
    return np.dot(v1, v2) / (np.linalg.norm(v1) * np.linalg.norm(v2))
```

## 3.3 预训练时代的开端（2018）

### 3.3.1 ELMo（2018.2）

**Embeddings from Language Models**

**关键创新：** 上下文相关的词向量（Contextualized Word Embeddings）




**特点：**

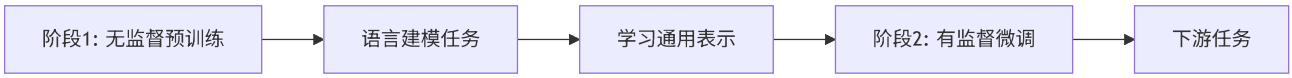
- [x] 动态词向量（同一个词在不同语境有不同表示）
- [x] 双向建模
- [ ] 基于LSTM，不能并行训练
- [ ] 需要针对下游任务微调权重

### 3.3.2 GPT-1（2018.6）

**Improving Language Understanding by Generative Pre-Training**

**架构：** Transformer Decoder-only

**两阶段范式：**




**模型规格：**

- 参数量：117M
- 层数：12层
- 隐藏维度：768
- 注意力头数：12

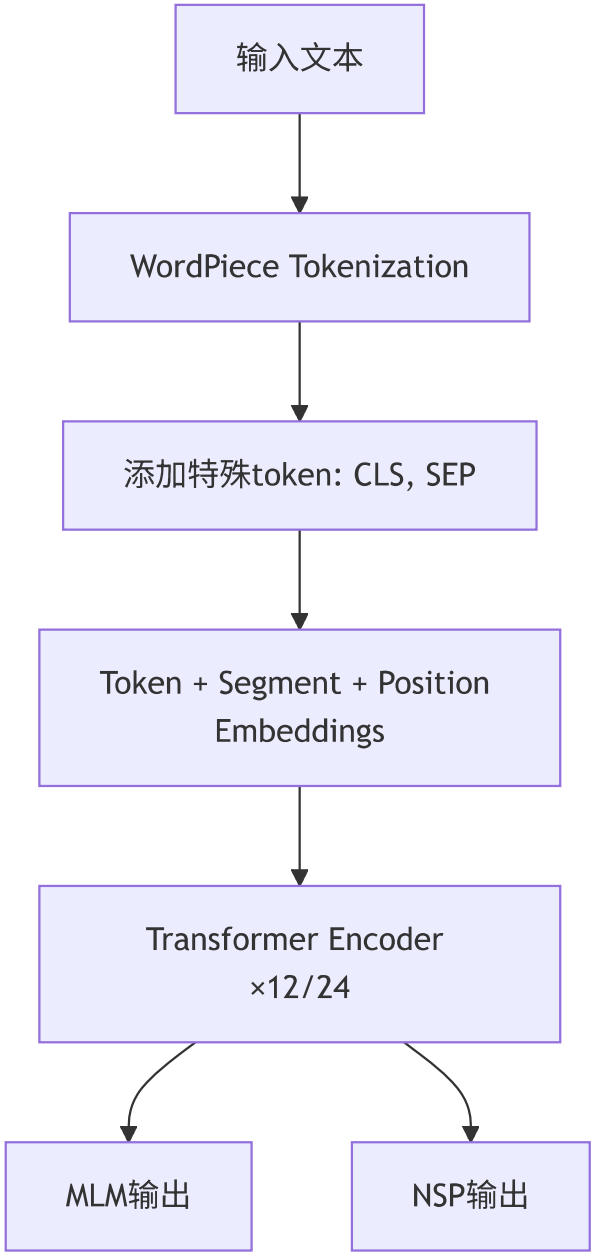
**创新点：**

- [x] 证明了Transformer的生成能力
- [x] "预训练+微调"范式
- [x] 单向语言模型

### 3.3.3 BERT（2018.10）

**Bidirectional Encoder Representations from Transformers**

**架构：** Transformer Encoder-only




**两大预训练任务：**

**1. Masked Language Model (MLM)：**

```
输入: "我 [MASK] 吃 [MASK]"
目标: 预测 "爱" 和 "饭"
```

```python
def create_masked_lm_data(tokens, mask_prob=0.15):
    """
    创建MLM训练数据
    """
    masked_tokens = tokens.copy()
    labels = [-100] * len(tokens)  # -100表示不计算损失

    for i, token in enumerate(tokens):
        if random.random() < mask_prob:
```

```
            prob = random.random()
            if prob < 0.8:
                masked_tokens[i] = "[MASK]"  # 80%替换为[MASK]
            elif prob < 0.9:
                masked_tokens[i] = random_token()  # 10%替换为随机token
            # 10%保持不变

            labels[i] = token  # 记录原始token用于计算损失

    return masked_tokens, labels
```

**2. Next Sentence Prediction (NSP)：**

```
输入：[CLS] 句子A [SEP] 句子B [SEP]
目标：判断B是否是A的下一句
```

**BERT的影响：**

- [x] 双向上下文建模
- [x] 在NLU任务上SOTA
- [x] 引发预训练模型热潮
- [ ] 不适合生成任务

### 3.3.4 三者对比

| 模型 | 架构 | 预训练任务 | 优势领域 | 局限性 |
|---|---|---|---|---|
| **ELMo** | BiLSTM | 语言建模 | 双向理解 | 不能并行，速度慢 |
| **GPT-1** | Decoder | 语言建模（单向） | 生成任务 | 单向，理解能力有限 |
| **BERT** | Encoder | MLM + NSP | 理解任务 | **不适合生成** |

## 3.4 模型规模化时代（2019-2020）

### 3.4.1 GPT-2（2019.2）

**Language Models are Unsupervised Multitask Learners**

**关键突破：** 零样本学习（Zero-shot Learning）

**模型规格对比：**

| 版本 | 参数量 | 层数 | 隐藏维度 | 头数 |
|---|---|---|---|---|
| GPT-2 Small | 117M | 12 | 768 | 12 |
| GPT-2 Medium | 345M | 24 | 1024 | 16 |

| 版本 | 参数量 | 层数 | 隐藏维度 | 头数 |
|---|---|---|---|---|
| GPT-2 Large | 762M | 36 | 1280 | 20 |
| **GPT-2 XL** | **1.5B** | **48** | **1600** | **25** |

**创新点：**

- [x] 规模扩大10倍
- [x] 更大更多样的训练数据（WebText）
- [x] 展示零样本能力
- [x] 生成质量大幅提升

**争议：** 最初因"安全考虑"未完全开源，引发讨论。

### 3.4.2 RoBERTa（2019.7）

**Robustly Optimized BERT Approach**

对BERT的优化版本：

**改进点：**

- [ ] 移除NSP任务（发现无用）
- [x] 动态masking（每次epoch不同mask）
- [x] 更大的batch size
- [x] 更多训练数据
- [x] 更长训练时间

**结果：** 性能全面超越BERT

### 3.4.3 其他重要模型

**XLNet（2019.6）**

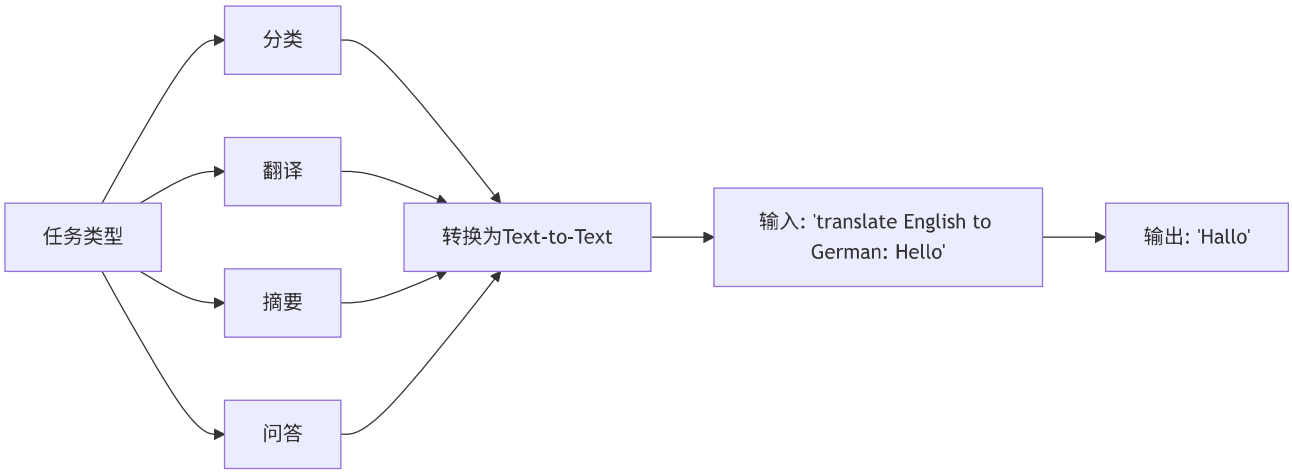
- 结合自回归和自编码
- Permutation Language Modeling

**ALBERT（2019.9）**

- 参数共享，减少参数量
- SOP（Sentence Order Prediction）任务

**T5（2019.10）**

- Text-to-Text Transfer Transformer
- 所有任务统一为seq2seq

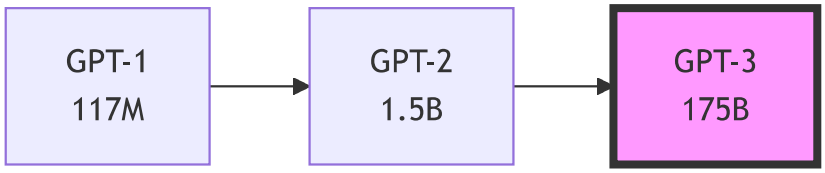



## 3.5 大模型涌现时代（2020-2023）

### 3.5.1 GPT-3（2020.5）

**Language Models are Few-Shot Learners**

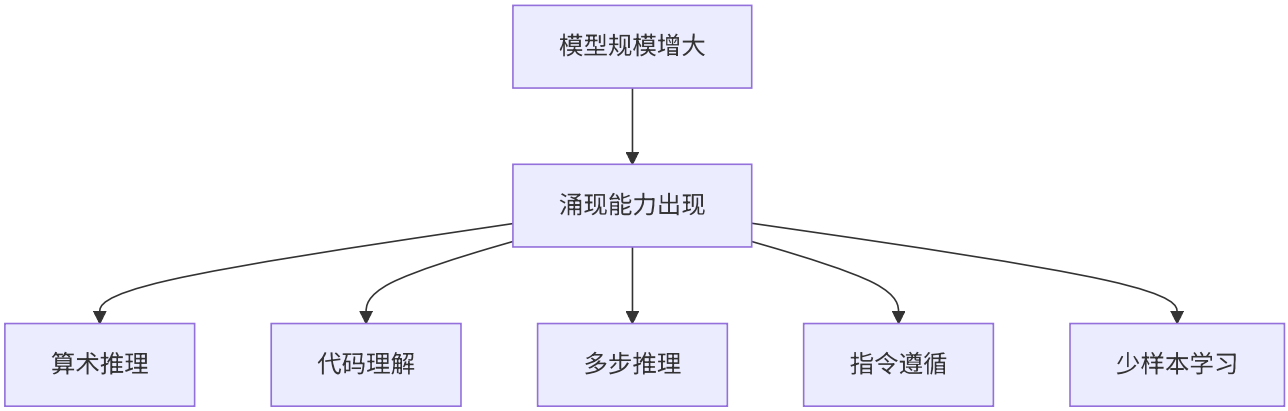
**规模跃升：**




**模型规格：**

| 版本 | 参数量 | 层数 | 隐藏维度 | 头数 | 上下文长度 |
|---|---|---|---|---|---|
| GPT-3 Small | 125M | 12 | 768 | 12 | 2048 |
| GPT-3 Medium | 350M | 24 | 1024 | 16 | 2048 |
| GPT-3 Large | 760M | 24 | 1536 | 16 | 2048 |
| GPT-3 XL | 1.3B | 24 | 2048 | 24 | 2048 |
| GPT-3 2.7B | 2.7B | 32 | 2560 | 32 | 2048 |
| GPT-3 6.7B | 6.7B | 32 | 4096 | 32 | 2048 |
| GPT-3 13B | 13B | 40 | 5140 | 40 | 2048 |
| **GPT-3 175B** | **175B** | **96** | **12288** | **96** | **2048** |

**涌现能力（Emergent Abilities）：**




**In-Context Learning示例：**

```
任务：情感分类

# Few-shot示例
输入：这个电影太棒了！
输出：正面

输入：服务态度很差。
输出：负面

输入：价格有点贵。
输出：负面

# 实际查询
输入：今天天气真好！
输出：<模型预测>
```

**关键发现：**

- [x] Few-shot性能随规模显著提升
- [x] 无需微调即可完成多种任务
- [x] 展示通用人工智能（AGI）的曙光

### 3.5.2 Codex（2021.8）

**基于GPT-3，针对代码训练**

**训练数据：**

- GitHub公开代码
- 多种编程语言

**应用：**

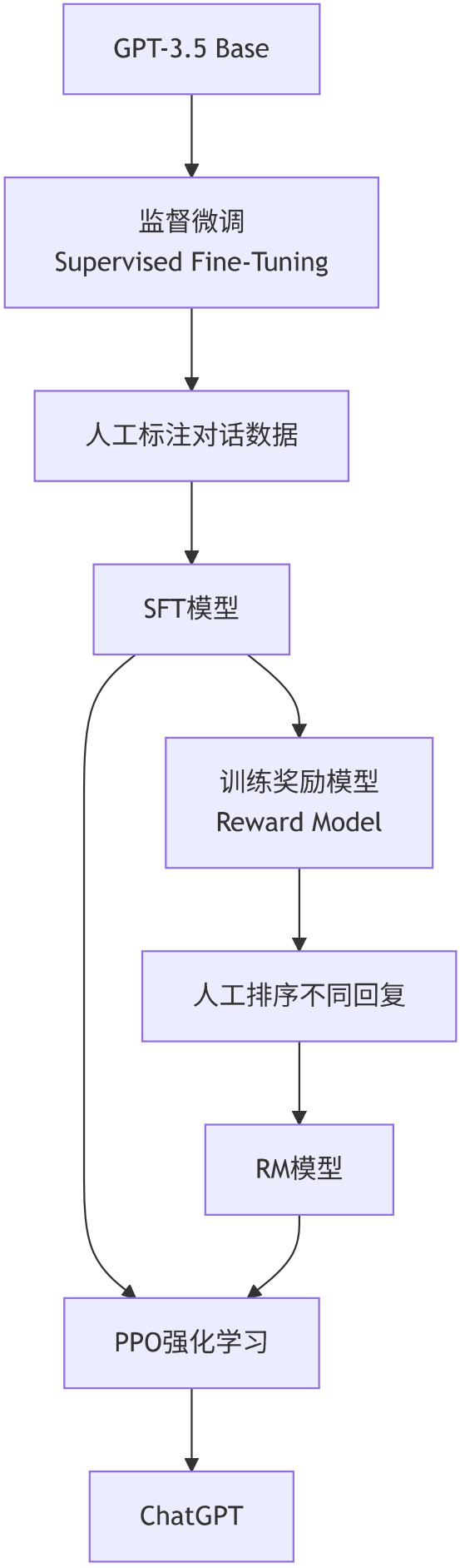
- GitHub Copilot
- OpenAI API code-davinci模型

**能力示例：**

```
# 提示：编写一个函数，计算列表中所有偶数的和

def sum_even_numbers(numbers):
    """
    计算列表中所有偶数的和

    Args:
        numbers: 整数列表

    Returns:
        所有偶数的和
    """
    return sum(num for num in numbers if num % 2 == 0)
```

### 3.5.3 ChatGPT（2022.11）

**基于GPT-3.5 (InstructGPT) + RLHF**




**RLHF三阶段：**

**阶段1: Supervised Fine-Tuning (SFT)**

输入：用户指令
输出：高质量人工回复

数据量：约13,000条对话

**阶段2: Reward Model (RM) Training**

给定一个指令和多个回复，人工排序：
回复A > 回复B > 回复C > 回复D

训练RM预测人类偏好分数

**阶段3: Proximal Policy Optimization (PPO)**

使用RM作为奖励信号，通过强化学习优化模型
目标：最大化奖励，同时不偏离SFT模型太远

**影响：**

- 🔥 发布2个月用户破1亿
- 🔥 引爆AI应用热潮
- 🔥 证明对话是最佳交互方式

### 3.5.4 GPT-4（2023.3）

**多模态大模型**

**能力提升：**

| 维度 | GPT-3.5 | GPT-4 | 提升 |
| --- | --- | --- | --- |
| 编程能力 | 通过70%简单题 | 通过85%困难题 | ⬆️ |
| 多语言 | 中等 | 接近母语者 | ⬆️⬆️ |
| 推理能力 | 基础 | 复杂多步推理 | ⬆️⬆️ |
| 幻觉率 | 较高 | 显著降低 | ⬆️⬆️ |
| 上下文 | 8K | 32K/128K | ⬆️⬆️⬆️ |
| 视觉理解 | ❌ | ✅ | 新能力 |

**基准测试表现：**

| 考试/测试 | GPT-3.5 | GPT-4 |
| --- | --- | --- |
| SAT阅读 | 69% | 93% |

| | | |
|---|---|---|
| SAT数学 | 70% | 89% |
| 律师资格考试 | 40% | 90% |
| GRE写作 | 54% | 99% |
| 医学知识测试 | 53% | 86% |
| 编程竞赛 | 约11% | 约89% |

## 3.6 开源大模型崛起（2023-至今）

### 3.6.1 LLaMA系列（Meta）

**LLaMA-1（2023.2）**

**模型规格：**

| 模型 | 参数量 | 层数 | 隐藏维度 | 头数 | 学习率 |
|---|---|---|---|---|---|
| LLaMA-7B | 7B | 32 | 4096 | 32 | 3e-4 |
| LLaMA-13B | 13B | 40 | 5120 | 40 | 3e-4 |
| LLaMA-33B | 33B | 60 | 6656 | 52 | 1.5e-4 |
| LLaMA-65B | 65B | 80 | 8192 | 64 | 1.5e-4 |

**技术特点：**

- [x] 完全开源（权重）
- [x] 使用RoPE位置编码
- [x] Pre-normalization（提升稳定性）
- [x] SwiGLU激活函数
- [x] 训练更长时间（1.4T tokens）

**架构改进：**

```
class LLaMABlock(nn.Module):
    """
    LLaMA的Transformer Block
    """
    def forward(self, x):
        # Pre-Norm + RMSNorm
        h = x + self.attention(self.attention_norm(x))
        out = h + self.ffn(self.ffn_norm(h))
        return out

class SwiGLU(nn.Module):
    """
    SwiGLU激活函数: Swish-Gated Linear Unit
    """
    def forward(self, x):
```

```
        x, gate = x.chunk(2, dim=-1)
        return F.silu(gate) * x
```

**LLaMA-2（2023.7）**

改进：

- [x] 商用许可
- [x] 上下文长度4096
- [x] 更多训练数据（2T tokens）
- [x] Chat版本（经过RLHF）

**LLaMA-3（2024）**

进一步提升性能，成为开源SOTA。

### 3.6.2 其他重要开源模型

**GLM系列（智谱AI）**


特点：

- [x] 中英双语
- [x] 轻量化（6B可在消费级GPU运行）
- [x] 国内应用广泛

**Bloom（2022）**

- 176B参数
- 多语言（46种语言）
- BigScience开源项目

**Falcon（2023）**

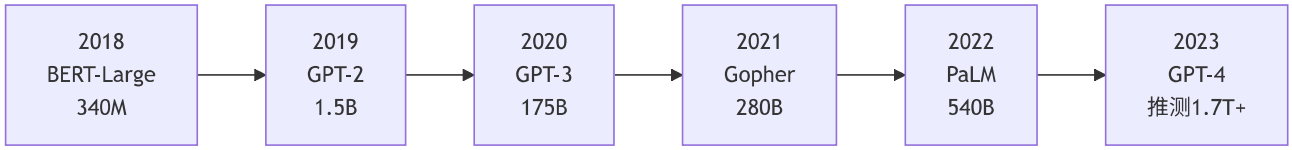
- 180B参数
- RefinedWeb数据集
- 开放商用

**Mistral 7B（2023.9）**

- 仅7B参数
- 性能超越LLaMA-13B
- Sliding Window Attention

## 3.7 模型规模对比

### 3.7.1 参数量演进




### 3.7.2 主流模型参数量对比表

| 模型 | 参数量 | 发布时间 | 开源状态 | 特点 |
|---|---|---|---|---|
| BERT-Large | 340M | 2018.10 | ✓ | 经典Encoder |
| GPT-2 | 1.5B | 2019.02 | ✓ | 首个GPT扩展 |
| T5-11B | 11B | 2019.10 | ✓ | Text-to-Text |
| GPT-3 | 175B | 2020.05 | ✗ API | 涌现能力 |
| GLM-130B | 130B | 2022.08 | ✓ | 中英双语 |
| BLOOM | 176B | 2022.07 | ✓ | 多语言 |
| LLaMA-65B | 65B | 2023.02 | ✓ | 开源标杆 |
| GPT-4 | ~1.7T? | 2023.03 | ✗ | 多模态 |
| PaLM 2 | ? | 2023.05 | ✗ | Google最强 |
| Claude 3 | ? | 2024.03 | ✗ | Anthropic |

## 3.8 Scaling Law（规模法则）

### 3.8.1 核心发现

**Kaplan et al. (2020) - OpenAI的Scaling Laws**

模型性能主要由三个因素决定：

1. **N**: 模型参数量
2. **D**: 训练数据量
3. **C**: 计算量（FLOPs）

**关键公式：**

$$L(N) \propto N^{-\alpha}$$

其中 $\alpha \approx 0.076$，即性能随参数量幂律提升。

**最优分配：**

给定计算预算C，最优策略：

- 模型参数 $N \propto C^{0.73}$
- 训练数据 $D \propto C^{0.27}$

### 3.8.2 Chinchilla Scaling Law

**DeepMind (2022) 重新审视规模法则**

**关键发现：** 大多数大模型训练不足（undertrained）！

**Chinchilla结论：**

对于相同计算预算，应该：

- 减小模型规模
- 增加训练数据

```
GPT-3风格：175B参数 × 300B tokens
Chinchilla风格：70B参数 × 1.4T tokens
结果：Chinchilla性能更好！
```

**新的最优配比：**

$$N \propto C^{0.50}, \quad D \propto C^{0.50}$$

即参数量和数据量应该等比例增长。

### 3.8.3 实际影响

**LLaMA的成功：**

- LLaMA-13B（13B × 1T tokens） > GPT-3（175B × 300B tokens）
- 验证了Chinchilla的理论

## 3.9 面试高频问题

### Q1: BERT和GPT的核心区别？

| 维度 | BERT | GPT |
| --- | --- | --- |
| **架构** | Encoder-only | Decoder-only |
| **注意力** | 双向（看到全部上下文） | 单向（只看左侧） |
| **预训练任务** | MLM + NSP | 语言建模（预测下一个词） |
| **适用场景** | 理解任务（分类、NER、问答） | **生成任务（对话、写作）** |
| **训练目标** | 填空（完形填空） | 续写（自回归） |
| **现代趋势** | 较少使用 | **主流** |

### Q2: 为什么现代大模型都用Decoder-only架构？

**答案要点：**

1. **统一范式**：

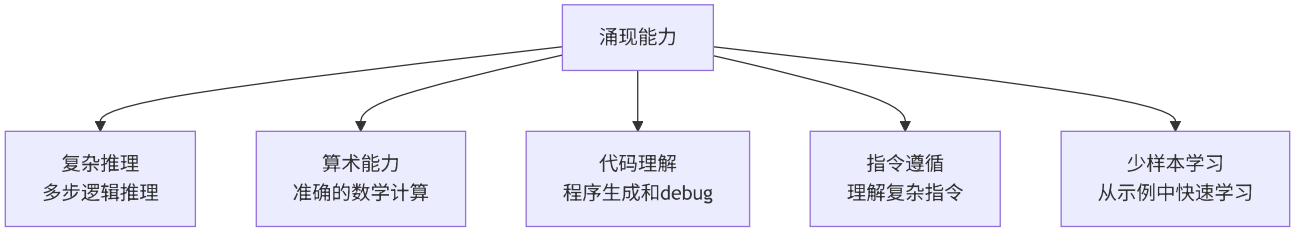
- 所有任务都能表示为文本生成
- 不需要为不同任务设计不同架构

2. **扩展性**：
   - 架构简单，易于扩展到千亿参数
   - 训练和推理流程统一
3. **少样本学习**：
   - 预训练和推理形式一致
   - In-context learning能力强
4. **实验验证**：
   - GPT-3证明了强大的零样本能力
   - 规模扩大带来涌现能力

### Q3: 什么是涌现能力（Emergent Abilities）？

**定义：** 当模型规模超过某个阈值时，突然出现的在小模型上没有的能力。

**典型涌现能力：**




**示例：**

| 任务 | GPT-2 (1.5B) | GPT-3 (175B) | 涌现? |
|---|---|---|---|
| 简单QA | 40% | 75% | 渐进 |
| 多步推理 | 随机 | 65% | ✓ 涌现 |
| 代码生成 | 几乎不能 | 能用 | ✓ 涌现 |

### Q4: In-Context Learning是如何工作的?

**现象：** 无需微调，只需在prompt中提供示例，模型就能学习新任务。

**示例：**

```
# Zero-shot
输入："Translate to French: Hello"
输出："Bonjour"

# One-shot
```

```
输入："Translate to French:
Good morning -> Bonjour
Hello"
输出："Bonjour"

# Few-shot
输入："Translate to French:
Good morning -> Bonjour
Thank you -> Merci
Goodbye -> Au revoir
Hello"
输出："Bonjour"
```

**机制（假说）：**

1. **模式匹配**：识别输入输出的对应关系
2. **任务推断**：从示例推断任务类型
3. **知识激活**：激活预训练时学到的相关知识

### Q5: 为什么ChatGPT需要RLHF?

**问题：** Base模型虽然强大，但：

- [ ] 不遵循指令
- [ ] 可能输出有害内容
- [ ] 啰嗦或不够helpful

**RLHF的作用：**


**效果对比：**

| 特性 | Base Model | After RLHF |
|---|---|---|
| 指令遵循 | 差 | [x] 好 |
| 有害内容 | 可能产生 | [x] 拒绝 |
| 回复质量 | 不稳定 | [x] 稳定 |
| 用户满意度 | 低 | [x] 高 |

## 3.10 本章小结

本章回顾了从Word2Vec到GPT-4的语言模型发展历史：

[x] **词向量时代**：Word2Vec、GloVe奠定基础 [x] **预训练时代**：ELMo、GPT、BERT开启新范式 [x] **规模化时代**：GPT-3展示涌现能力 [x] **对齐时代**：ChatGPT证明RLHF的重要性 [x] **开源崛起**：LLaMA等推动技术民主

化

**关键趋势：**

1. Decoder-only成为主流架构
2. 模型规模持续增大
3. 数据质量比数量更重要（Chinchilla Scaling Law）
4. 对齐（Alignment）是产品化的关键

---

**下一章预告：** 第4章将深入讲解预训练技术的细节，包括数据处理、训练策略等。