Scaling Laws

📋

1. 🎯 Scaling Law
2. 🧮
3. 🕰️ GPT-4
4. 🔬
5. 💻 MacBook Scaling Law
6. 🚀 Scaling Law
7. 🔮 AGI

Part 1:

Scaling Law

🤔

1.
    - GPT-3: 175B
    - GPT-4: 1.8T
    -
2.
    - LLaMA: 1.4T tokens
    - Chinchilla: 20
    - vs
3.
    -

💡 Scaling Law

		Scaling Law
🎯		N-D
💰		
🔬		
🚀	AGI	

GPT

```
GPT-1 (2018)    117M params   →   Loss: ~3.4
GPT-2 (2019)    1.5B params   →   Loss: ~3.0
GPT-3 (2020)    175B params   →   Loss: ~2.0
GPT-4 (2023)    ~1.8T params  →   Loss: ~1.5?

关键发现: Loss 与参数量的关系遵循幂律!
L(N) = (N_c / N)^α

这意味着什么?
→ 我们可以用小模型预测大模型性能
→ 可以提前规划资源分配
→ 能定量预测 AGI 所需规模
```

Part 2:

🧮 **(Power Law)**

`y = a·x^b`

`y ∝ x^b`

-
-
-

`log(y) = log(a) + b·log(x)`

-
- = b (scaling exponent)

📐

1. (Scale-free)

```
# 无论在什么尺度上观察，形态都类似
y(λx) = a·(λx)^b = λ^b · a·x^b = λ^b · y(x)

# 例子：分形、海岸线、互联网拓扑
```

2.

```
# 少数元素占据主要质量，多数元素分布在长尾
# 80–20 法则、马太效应

# 在深度学习中：
# – 少数困难样本贡献大部分 loss
# – 少数参数主导模型能力
```

→

- →
- →
- → Lévy
-

→

🔥 (Phase Transition)

100°C

< 100°C		
= 100°C		
> 100°C		

```
小模型（< N_c）        →  简单模式匹配
临界规模（≈ N_c）      →  能力涌现（Emergence）
大模型（> N_c）        →  复杂推理、上下文学习
```

✨ (Emergent Abilities)

- In-context Learning
- Chain-of-Thought
- Few-shot Learning
-
-

```
Ability(N) = {
    0,          N < N_critical
    f(N),       N ≥ N_critical
```

Part 3:

GPT-4

Scaling Law

2001 ——→ 早期观察：感知机的泛化理论
Vapnik: VC维理论

2017 ——→ 实证研究：深度的作用
ResNet: 从残差到超深网络

2018 ——→ 数据缩放：第一个系统性研究
Hestness (Baidu): 数据 Scaling Law

2020 ——→ 奠基之作：OpenAI Scaling Laws
Kaplan et al.: 计算、参数、数据三要素

2022 ——→ 训练优化：Chinchilla 定律
Hoffmann et al.: 数据量要 20×参数量

2023-2024 →→ 多维度扩展：推理时计算、长度泛化
Microsoft/Google: 新的 Scaling 维度

1: Statistical Learning Theory (1995-2001)

Vapnik-Chervonenkis Theory

泛化误差上界:
$E_{gen} \leq E_{train} + O(\sqrt{(VC_{dim} / N)})$

关键洞察:
- 模型复杂度 ↑ → 表达能力 ↑, 但泛化差距 ↑
- 训练样本 ↑ → 泛化差距 ↓
- 存在最优的复杂度–样本平衡点

局限:
❌ 对深度神经网络过于宽松（上界太松）
❌ 无法解释过参数化现象
❌ 没有考虑优化算法的影响

2: Deep Learning Revival (2012-2017)

```
AlexNet (2012):      8 层  →  ImageNet Top-5: 84.7%
VGG (2014):         19 层  →  ImageNet Top-5: 92.7%
ResNet (2015):     152 层  →  ImageNet Top-5: 96.4%

观察:
✅ 深度 ↑ → 性能 ↑  (但有瓶颈)
✅ 残差连接突破瓶颈
❌ 还没有定量的 Scaling Law
```

He et al. (2015)

- skip connection
-
-

3: Data Scaling (Hestness et al., 2018)

Scaling Law

核心发现:
Loss(D) = A + B / D^α

其中:
- D: 数据量
- α ≈ 0.35 (不同任务略有差异)
- A: 不可约误差 (irreducible error)
- B: 可学习部分的规模

关键结论:
✅ 指数级增加数据 → 线性提升性能
✅ 存在数据饱和点 (边际收益递减)
✅ 不同任务有不同的数据效率

4: Kaplan Scaling Laws (OpenAI, 2020)

```
核心公式:
L(N, D, C) = [
    (N_c / N)^α_N +
    (D_c / D)^α_D +
    (C_c / C)^α_C
]

其中:
- N: 参数量 (Parameters)
- D: 数据量 (Dataset size)
- C: 计算量 (Compute FLOPs)

实验发现:
α_N ≈ 0.076    (参数主导)
α_D ≈ 0.095    (数据次之)
α_C ≈ 0.050    (计算决定上限)
```

🔬 Kaplan Scaling Laws:

1.

```
# 固定计算预算时，优先增加参数量
Best: 大模型 + 少训练步数
Bad:  小模型 + 多训练步数

例子:
175B × 300B tokens  >  6B × 8.75T tokens
(相同计算量，但前者更优)
```

2.

```
# 在实验范围内 (300B tokens)，数据越多越好
# 没有观察到明显的数据饱和现象
```

3.

```
# 跨越 6 个数量级 (10^3 到 10^9 参数)
```

5: Chinchilla Scaling Laws (DeepMind, 2022)

Kaplan

Kaplan (2020):

→ GPT-3: 175B params, 300B tokens (1.7×)

Hoffmann (2022):

→ Chinchilla: 70B params, 1.4T tokens (20×)

→ **Gopher (280B)**

```
新的最优配比:
N_optimal = (C / 6)^0.5
D_optimal = (C / 3)^0.5

简化规则:
D ≈ 20 × N  (tokens 应该是参数的 20 倍)
```

🤔 Kaplan vs Chinchilla

	Kaplan (2020)	Chinchilla (2022)
	10M - 1B	70M - 16B
	5B - 300B tokens	5B - 500B tokens
	N ≫ D	N ≈ D/20

- Kaplan:
- Chinchilla:

📊 Chinchilla

```
❌ 旧思路：越大越好
GPT-3 (175B, 300B tokens)
Gopher (280B, 300B tokens)
Megatron-Turing (530B, 270B tokens)

✅ 新思路：高效训练
Chinchilla (70B, 1.4T tokens)  → 超越 Gopher
LLaMA (7B-65B, 1T-1.4T tokens) → 开源 SOTA
Mistral (7B, 过度训练)          → 超越 LLaMA-13B

结论：
💰 相同成本，性能提升 30-50%
⚡ 推理速度快 4-6 倍
🌍 普及化：个人 GPU 可跑
```

6: Scaling (2023-2024)

Scaling Law

```
传统:
Performance = f(训练时计算)

新发现:
Performance = f(训练时计算, 推理时计算)

例子:
- Chain-of-Thought: 更多推理步骤 → 更好性能
- Self-Consistency: 采样多个回答 → 投票
- Tree-of-Thoughts: 搜索推理路径
- 强化学习 (RLHF, PPO): 推理时优化
```

🧠

Snell et al. (2024)

```
Accuracy(N, T) = α · (N · T)^β

其中:
- N: 模型参数量
- T: 推理时计算量 (思考步数)
- β ≈ 0.3 - 0.5

关键洞察:
✨ 训练和推理可以相互替代!
- 小模型 + 多步推理 ≈ 大模型 + 少步推理
- 灵活的计算分配策略
```

- OpenAI o1: →
- DeepSeek-R1: +

Part 4:

🎯 1: Scaling (Compute)

(FLOPs)

```
C = 6·N·D

其中:
- C: 总计算量 (FLOPs)
- N: 参数量
- D: 训练 tokens 数
- 6: 前向(2) + 反向(4)

例子:
GPT-3:
N = 175B
D = 300B tokens
C = 6 × 175B × 300B = 3.15 × 10^23 FLOPs
   ≈ 315 ZettaFLOPs
```

💰

	FP16		GPT-3	
MacBook M3 Max	14 TFLOPS	$0	~700	
RTX 4090	82 TFLOPS	$1,600	~120	
A100 (80GB)	312 TFLOPS	$15,000	~32	
H100 (80GB)	1000 TFLOPS	$30,000	~10	
8×A100	2.5 PFLOPS	$120K	~4	~$5M
1024×A100	320 PFLOPS	$15M	~11	**~$5M**

💡 Scaling

📈 Scaling

```
# Kaplan et al. (2020)
L(C) = (C_c / C)^α_c

其中:
α_c ≈ 0.05     (计算效率指数)
C_c ≈ 10^10    (临界计算量)

解读:
- 10× 计算 → 1.12× 性能提升
- 100× 计算 → 1.29× 性能提升
- 1000× 计算 → 1.48× 性能提升

结论:
✅ 计算是性能上限
⚠️ 但单纯增加计算收益递减
💡 需要配合参数和数据增长
```

🎯 2: Scaling (Parameters)

```
ELMo (2018):           94M
BERT-base (2018):     110M
BERT-large (2019):    340M
GPT-2 (2019):         1.5B
T5 (2020):            11B
GPT-3 (2020):         175B
PaLM (2022):          540B
GPT-4 (2023):         ~1.8T (MoE)

每年增长 10×
18 个月翻一个数量级
```

🔬 Scaling

Kaplan Scaling Law ( )

```
L(N) = (N_c / N)^α_N + L_∞

实验拟合:
α_N ≈ 0.076
N_c ≈ 8.8 × 10^13
L_∞ ≈ 1.69  (不可约误差)

关键发现:
1. 10× 参数 → 0.84× Loss
2. 100× 参数 → 0.71× Loss
3. 1000× 参数 → 0.59× Loss

斜率约 -0.076 in log-log space
```

```
有效参数量 = 实际参数量 × 利用率

影响因素:
1. 架构设计
   - Dense: 100% 参数激活
   - MoE: ~10% 参数激活

2. 训练充分性
   - 欠训练: ~50% 参数有效
   - 充分训练: ~90% 参数有效

3. 任务相关性
   - 预训练: 70% 参数泛化
   - Fine-tuning: 95% 参数特化
```

🎯 3: Scaling (Data)

```
BERT (2018):          3.3B tokens   (Wikipedia + Books)
GPT-2 (2019):         40B tokens    (WebText)
GPT-3 (2020):         300B tokens   (Common Crawl)
PaLM (2022):          780B tokens   (高质量多语言)
LLaMA (2023):         1.4T tokens   (去重 + 过滤)
Llama 2 (2023):       2T tokens     (更长训练)
Llama 3 (2024):       15T tokens    (新数据源)

趋势: 质量 > 数量
```

Scaling

Chinchilla Optimal Scaling

```
给定计算预算 C，最优配置：

N_optimal = G_N · C^a
D_optimal = G_D · C^b

其中：
a ≈ 0.50  （参数随计算开方增长）
b ≈ 0.50  （数据也随计算开方增长）

实用公式：
D_tokens ≈ 20 × N_parameters

例子：
70B 模型 → 需要 1.4T tokens
7B 模型  → 需要 140B tokens
```

🔍 **vs**

(Phi-1.5, Microsoft 2023)

GPT-3.5	175B	300B		Baseline
LLaMA	7B	1T		0.9× GPT-3.5
Phi-1.5	**1.3B**	**30B**		**1.2× GPT-3.5** ( )

✨ Scaling Law
✨ 1B + > 10B +
✨

Scaling

```
训练数据 = Σ (Domain_i × Weight_i)

常见配比 (LLaMA):
- CommonCrawl: 67%      (网页, 多样性)
- C4: 15%               (过滤后网页)
- GitHub: 4.5%          (代码)
- Wikipedia: 4.5%       (知识)
- Books: 4.5%           (长文本)
- ArXiv: 2.5%           (科学)
- StackExchange: 2%     (问答)

关键:
✅ 多样性 > 单一来源大数据
✅ 长尾知识不可忽视
✅ 不同领域的 Scaling 速率不同
```

Part 5:

MacBook Scaling Law

💻 MacBook MPS

- ❌ 8×A100 ($120K)
- ❌ 175B (TB)
- ❌ PB

+

```
核心思想:
1. 在小规模上验证幂律关系
2. 用早停 + 拟合外推大规模
3. 分层采样覆盖多个数量级

模型规模: 2.5M → 1.5B (3 orders of magnitude)
数据规模: 1M → 500M tokens
计算预算: MacBook M3 Max (1-2 天)
```

🧪

GPT-2 Transformer

```
configs = {
    "tiny": {
        "n_layers": 4,
        "n_heads": 4,
        "d_model": 128,
        "params": 2.5M
    },
    "small": {
        "n_layers": 6,
        "n_heads": 6,
        "d_model": 384,
        "params": 23M
    },
    "medium": {
        "n_layers": 12,
        "n_heads": 12,
        "d_model": 768,
        "params": 124M
    },
    "large": {
        "n_layers": 24,
        "n_heads": 16,
        "d_model": 1024,
        "params": 355M
```

OpenWebText

```
# 数据规模采样
data_sizes = [
    1_000_000,       # 1M tokens
    5_000_000,       # 5M
    10_000_000,      # 10M
    50_000_000,      # 50M
    100_000_000,     # 100M
    500_000_000      # 500M (如果时间允许)
]

# 采样策略:
# - 保持领域分布一致
# - 随机采样 (避免偏差)
# - 使用相同的 tokenizer (GPT-2 BPE)
```

⚡ MPS

Apple Silicon GPU

```
import torch

# 1. 使用 MPS 后端
device = torch.device("mps")

# 2. 混合精度训练
use_amp = True  # FP16

# 3. 梯度累积（模拟大 batch）
gradient_accumulation_steps = 8
effective_batch_size = batch_size * grad_accum_steps

# 4. 梯度检查点（节省内存）
model.gradient_checkpointing_enable()

# 5. 优化数据加载
num_workers = 0  # MPS 不支持多进程
pin_memory = False
```

🎯

small 1

10-20%

```
def early_stopping_extrapolation(losses, steps):
    """
    拟合损失曲线：L(t) = L_∞ + A / t^α

    只训练到 20% → 外推到 100%
    """
    # 拟合参数
    def loss_curve(t, L_inf, A, alpha):
        return L_inf + A / (t ** alpha)

    # 优化拟合
    params, _ = curve_fit(loss_curve, steps, losses)

    # 外推最终 loss
    final_loss = loss_curve(max_steps, *params)

    return final_loss
```

📈 ( )

Scaling

```
模型大小（N）  |  Loss（验证集）  |  训练时间
-----------------------------------------
2.5M          |  4.12           |  0.5h
10M           |  3.65           |  1h
23M           |  3.28           |  2h
50M           |  2.98           |  4h
124M          |  2.65           |  10h
355M          |  2.38           |  28h

拟合曲线：
L(N) = 1.8 + 450 / N^0.08
R² = 0.995

与 OpenAI 报告一致（α ≈ 0.076）
```

Scaling

(50M)

```
数据量 (D)     |  Loss   |  训练时间
-------------------------------------
1M             |  3.85   |  0.2h
5M             |  3.42   |  0.8h
10M            |  3.21   |  1.5h
50M            |  2.89   |  7h
100M           |  2.74   |  14h

拟合曲线:
L(D) = 2.2 + 180 / D^0.09
R² = 0.988

结论: 数据收益略高于参数 (0.09 vs 0.08)
```

🔬 Chinchilla

```
C = 6 × N × D = constant = 10^18 FLOPs

配置 A (Kaplan 风格):
N = 50M, D = 3.3B tokens  →  Loss = 2.98

配置 B (Chinchilla 风格):
N = 25M, D = 6.6B tokens  →  Loss = 2.85

配置 C (平衡):
N = 35M, D = 4.7B tokens  →  Loss = 2.78 ✅ 最优

结论:
✅ 验证 Chinchilla: 平衡配置更优
✅ D ≈ 20×N 在小规模也成立
```

📐 Scaling

```
L(N, D) = L_∞ + A_N / N^α_N + A_D / D^α_D

参数:
L_∞ = 1.85    (不可约误差)
A_N = 450
A_D = 180
α_N = 0.08
α_D = 0.09

应用:
# 预测 GPT-4 级别模型
N = 1.8T = 1.8 × 10^12
D = 13T = 1.3 × 10^13

L(N, D) = 1.85 + 450/(1.8e12)^0.08 + 180/(1.3e13)^0.09
        ≈ 1.85 + 0.05 + 0.03
        ≈ 1.93
```

🎉 MacBook

	$5M	Scaling Law
✅	$3K MacBook	
✅	2	3
✅		90%
✅		±5-10%

```
git clone https://github.com/yourname/scaling-law-mps
cd scaling-law-mps
./quickstart.sh

# 3 种模式:
python mps_framework_example.py --mode quick    # 2h
python mps_framework_example.py --mode dev      # 1d
python mps_framework_example.py --mode full     # 1w
```

Part 6:

Scaling Law

🎯 1:

1×A100 (1)

Scaling Law

```
# 硬件规格
GPU: A100 80GB
FP16 TFLOPS: 312
可用时间: 30 天 × 24h = 720h

# 计算预算
C = 312 TFLOPS × 720h × 3600s
  = 8.09 × 10^20 FLOPs

# Chinchilla 最优配置
N_opt = (C / 6)^0.5 / 20 ≈ 120M 参数
D_opt = 20 × N_opt ≈ 2.4B tokens

# 预测性能
L = 1.85 + 450/(120e6)^0.08 + 180/(2.4e9)^0.09
  ≈ 2.65
```

💡

	12 , 768	~120M
	2.5B tokens	OpenWebText / C4
Batch Size	256	1024
	3e-4	Cosine decay
	~10K steps	
Loss	~2.6	GPT-2 small

- ❌ 400M + 600M tokens ( )
- ✅ 120M + 2.4B tokens ( )

🎯 2: Fine-tuning

vs

Scaling Law

```
# 选项 A: 从头训练
L(N, D) = 1.85 + 450/N^0.08 + 180/D^0.09

# 选项 B: 继续训练预训练模型
L(N, D + D_pretrain) = 1.85 + 450/N^0.08 + 180/(D + D_pretrain)^0.09

例子:
目标: 50M 模型在特定领域
选项 A: 从头用 1B 领域数据
  L_A = 1.85 + 450/(50e6)^0.08 + 180/(1e9)^0.09 ≈ 2.85

选项 B: 基于 GPT-2 (预训练 40B) + 1B 领域数据
  L_B = 1.85 + 450/(50e6)^0.08 + 180/(41e9)^0.09 ≈ 2.72

结论: 继续训练更优 (13% 提升)
```

🎯 3:

Fine-tune

```
# 总预算: C = 10^20 FLOPs

策略 1: 单阶段
N = 100M, D = 1.6B tokens (高质量)
L_1 = 2.70

策略 2: 两阶段
阶段 1: N = 100M, D = 10B tokens (CommonCrawl)
  → L = 2.45
阶段 2: N = 100M, D = 0.5B tokens (高质量)
  → L_final = 2.35

策略 3: 课程学习 (Curriculum)
阶段 1: 简单数据 (3B tokens)
阶段 2: 中等数据 (5B tokens)
阶段 3: 困难数据 (2B tokens)
```

Scaling Calculator

```python
class ScalingCalculator:
    """
    Scaling Law 计算器
    """
    def __init__(self, L_inf=1.85, A_N=450, A_D=180,
                 alpha_N=0.08, alpha_D=0.09):
        self.L_inf = L_inf
        self.A_N = A_N
        self.A_D = A_D
        self.alpha_N = alpha_N
        self.alpha_D = alpha_D

    def predict_loss(self, N, D):
        """预测 Loss"""
        return (self.L_inf +
                self.A_N / (N ** self.alpha_N) +
                self.A_D / (D ** self.alpha_D))

    def optimal_allocation(self, compute_budget):
        """给定计算预算，返回最优 N 和 D"""
        N_opt = (compute_budget / 6) ** 0.5 / 20
        D_opt = 20 * N_opt
        return N_opt, D_opt

    def compare_configs(self, configs):
        """对比多个配置"""
        for name, (N, D) in configs.items():
            loss = self.predict_loss(N, D)
            compute = 6 * N * D
            print(f"{name:20s}: Loss={loss:.3f}, Compute={compute:.2e}")
```

❌ 1:

- → →
- 1B + 100M tokens < 100M + 1B tokens

❌ 2:

- GPT-3 (175B, 300B tokens) < Chinchilla (70B, 1.4T tokens)
-

❌ 3:

- Scaling Law
- 10× ≠ 10×

❌ 4:

✅

训练新模型的 Checklist：

1. ✅ 确定计算预算 C
2. ✅ 用 Chinchilla 公式计算 N_opt, D_opt
3. ✅ 选择架构（Transformer, MoE, etc.）
4. ✅ 准备高质量数据（去重、过滤）
5. ✅ 在小规模 pilot 实验验证 Scaling
6. ✅ 用早停策略预测最终性能
7. ✅ 如果预测不达标，调整配置
8. ✅ 全量训练 + 监控 Loss 曲线
9. ✅ 与 Scaling Law 对比（诊断问题）
10. ✅ Fine-tuning + RLHF

Part 7:

AGI

🔮

(2024)

```
GPT-4 (2023):
- 参数: ~1.8T (MoE, ~280B activated)
- 数据: ~13T tokens
- 计算: ~2.5 × 10^25 FLOPs
- Loss: ~1.5-2.0 (推测)
- 能力: 接近人类专家 (部分领域)

差距:
✅ 通过: SAT, GRE, Bar Exam
❌ 缺陷: 长期推理, 数学证明, 科研创新
```

🧠 Scaling

```
人脑:
- 神经元: 860 亿 (86B)
- 突触: 100 万亿 (100T)
- 等效参数: ~100T
- 功耗: 20W
- 训练数据: 一生经验 (~10^9 sec × 100 MB/s ≈ 100PB)

GPT-4:
- 参数: ~2T (MoE)
- 激活参数: ~280B
- 功耗: ~1000W (推理)
- 训练数据: 13T tokens ≈ 50TB

结论:
✅ 参数量接近 (0.3% 人脑)
❌ 数据效率差 1000×
❌ 能耗效率差 50×
```

📈 AGI

1:

```
# 假设 Loss 与能力线性相关
# 人类水平 ≈ Loss ~ 1.0（猜测）

L(N, D) = 1.85 + 450/N^0.08 + 180/D^0.09 = 1.0

求解:
N ≈ 10T 参数
D ≈ 200T tokens
C ≈ 10^27 FLOPs

成本估算 (2024 价格):
- H100: 1 PFLOPS × $30K
- 需要: 10^27 / (10^15 × 3600 × 24 × 30) ≈ 400 H100-年
- 总成本: 400 × $30K × 12 月 ≈ $144M
- 训练时间: 1年 (400 H100) 或 1月 (4800 H100)
```

🎯

1:

- < 50T tokens ( )

2:

- 10^27 FLOPs = GPU
- ( )

3:

- GPT-5: ~50 GWh ( )

🚀

1.

```
# 合成数据
Phi 系列: 教科书质量数据 → 10× 数据效率

# 多模态
GPT-4V: 图像 + 文本 → 新的数据来源

# 自我改进
AlphaGo Zero: 自我对弈生成无限数据
```

2.

```
# MoE
稀疏激活 → 10× 参数, 1× 计算

# 状态空间模型
Mamba: 线性复杂度 → 100× 序列长度
```

Scaling

Scaling

```
More Compute → More Parameters → More Data → Better Performance
```

Scaling

```
1. 推理时计算 (OpenAI o1)
   - 10× 思考时间 → 接近人类推理

2. 强化学习 (RL Scaling)
   - 更多互动 → 涌现规划能力

3. 长上下文 (Context Length)
   - 100K → 1M tokens → 更强记忆

4. 模态融合 (Multimodal)
   - 视觉 + 听觉 + 触觉 → 具身智能

5. 终身学习 (Lifelong)
```


```
2024 ──────────► GPT-5 / Gemini 2.0
                 10T 参数, 50T tokens
                 Loss ~ 1.3

2025-2026 ─────► 多模态 AGI 雏形
                 100T 参数 (MoE)
                 多模态预训练
                 推理时计算 Scaling

2027-2030 ─────► 专业级 AGI
                 1000T 参数
                 自我改进
                 超越人类专家 (大部分领域)

2030+ ─────────► 超级智能?
                 ? ? ?
                 算法突破 + Scaling
                 新的物理定律?
```

🎓

✅

-
-

✅

- Transformer
-

✅ **Scaling**

-
- Scaling Law

Scaling Law

🎯

1.

```
L(N, D) = L_∞ + A_N / N^α_N + A_D / D^α_D

- 对数空间中的直线
- 可预测、可外推
- 自相似性
```

2.

```
Kaplan (2020): 参数为王
  → 催生 GPT-3 (175B, 欠训练)

Chinchilla (2022): 数据同等重要
  → 70B 超越 280B

教训: 充分训练 > 盲目增大
```

💡

```
1. 确定预算 → 2. Chinchilla 公式 → 3. Pilot 实验
4. 早停预测 → 5. 全量训练 → 6. 验证 Scaling
```

```
1. 诊断：实际 Loss vs 预测 Loss
2. 如果差距大：数据问题 or 训练不足
3. 对症下药：清洗数据 or 增加训练
```

```
1. 小模型 + 充分训练 > 大模型 + 欠训练
2. 高质量数据 > 大规模噪声数据
3. 早停外推节省 90% 时间
```

🔮

Scaling	GPT-4 (~10^25)	10^27-10^28	★★★
Scaling	50T tokens		★★★★
Scaling	o1 ( )		★★★★★
Scaling	GPT-4V		★★★★★
	Transformer		★★★★★

1.
2.
3. Scaling Law

1.
2.
3. Scaling

1. Scaling AGI
2.

1. **Kaplan et al. (2020)**: Scaling Laws for Neural Language Models
2. **Hoffmann et al. (2022)**: Training Compute-Optimal Large Language Models
3. **Wei et al. (2022)**: Emergent Abilities of Large Language Models
4. **Snell et al. (2024)**: Scaling LLM Test-Time Compute Optimally

- Hugging Face : Scaling Laws
- OpenAI : GPT-4 System Card
- Anthropic: Constitutional AI Scaling

- nanoGPT (Karpathy): GPT

🙏 &

- OpenAI, DeepMind, Anthropic
-
- Scaling Law

💬 问题讨论
📧 联系方式
🌐 项目主页
⭐ GitHub Star

Questions?

Scaling Law