北京大学
PEKING UNIVERSITY

基于AI的数字系统设计

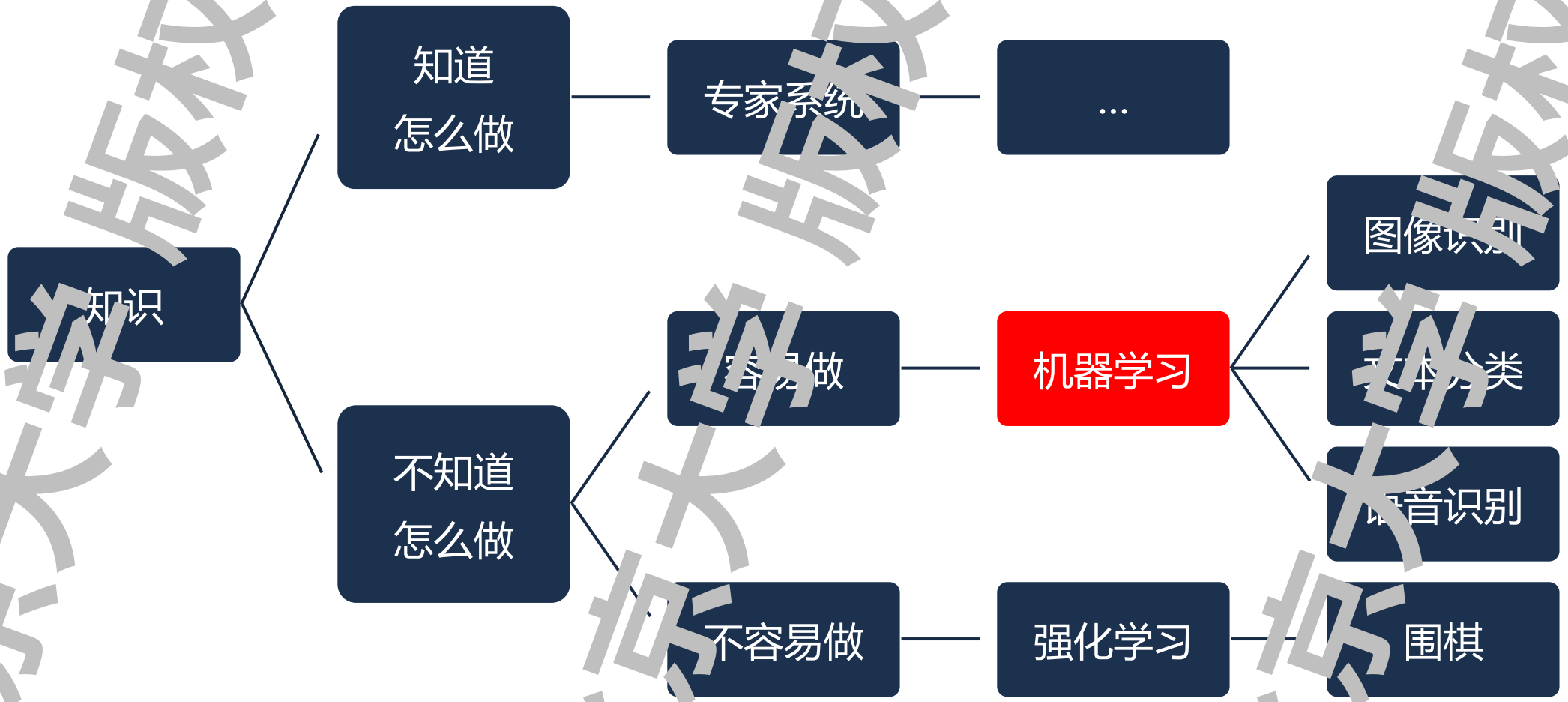
第三讲：主流AI算法简介

主讲：陶耀宇

2026年秋季

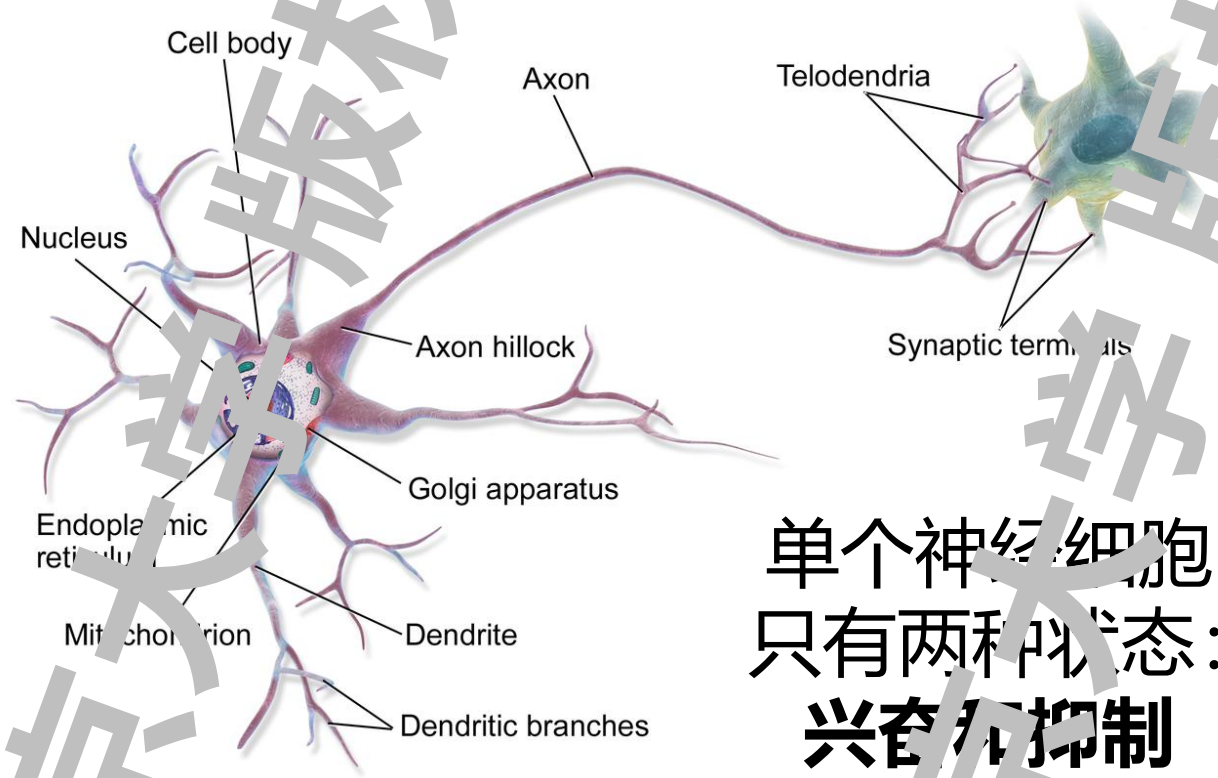
如何开发一个人工智能算法?

- 三大根本性问题：做什么、怎么做、做的好?

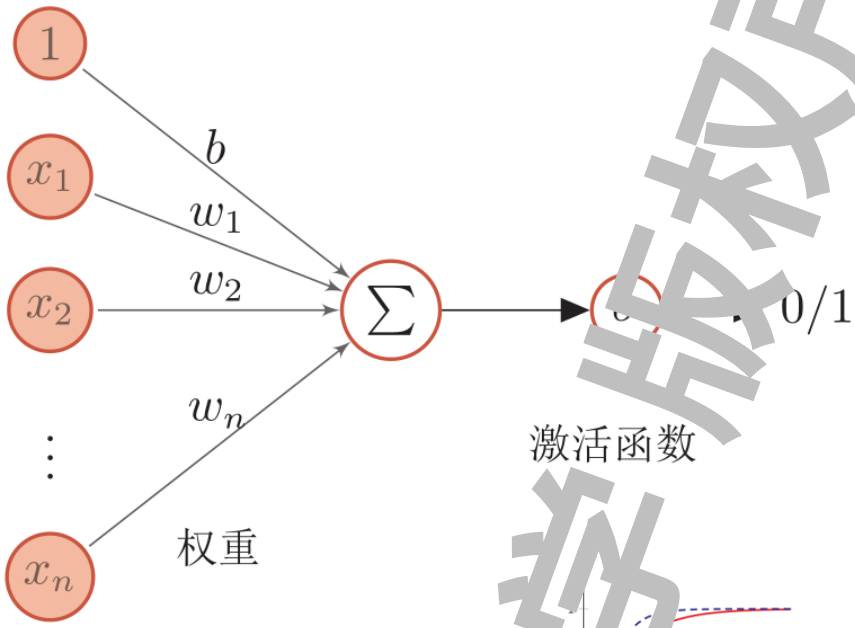


神经网络在最近十余年成为主流

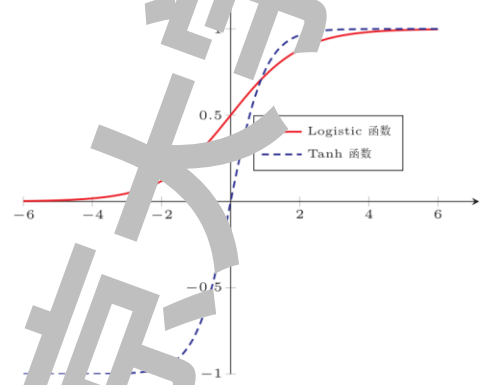
- 受到生物神经元启发



单个神经细胞只有两种状态：**兴奋和抑制**

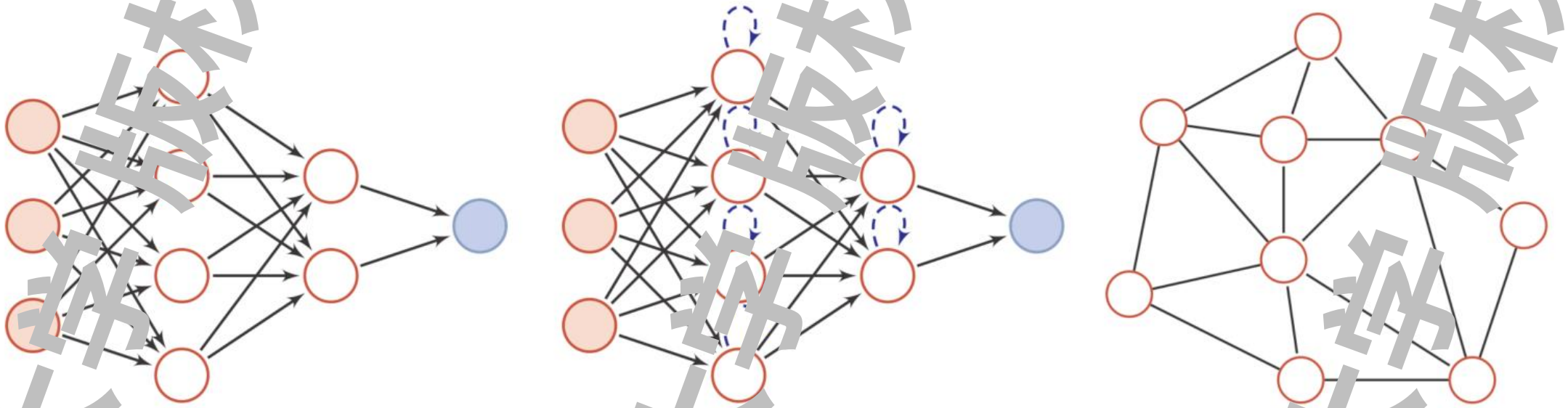


人工神经元



神经网络在最近十余年成为主流

- 人工神经网络由神经元模型构成，这种由许多神经元组成的信息处理网络具有并行分布结构。

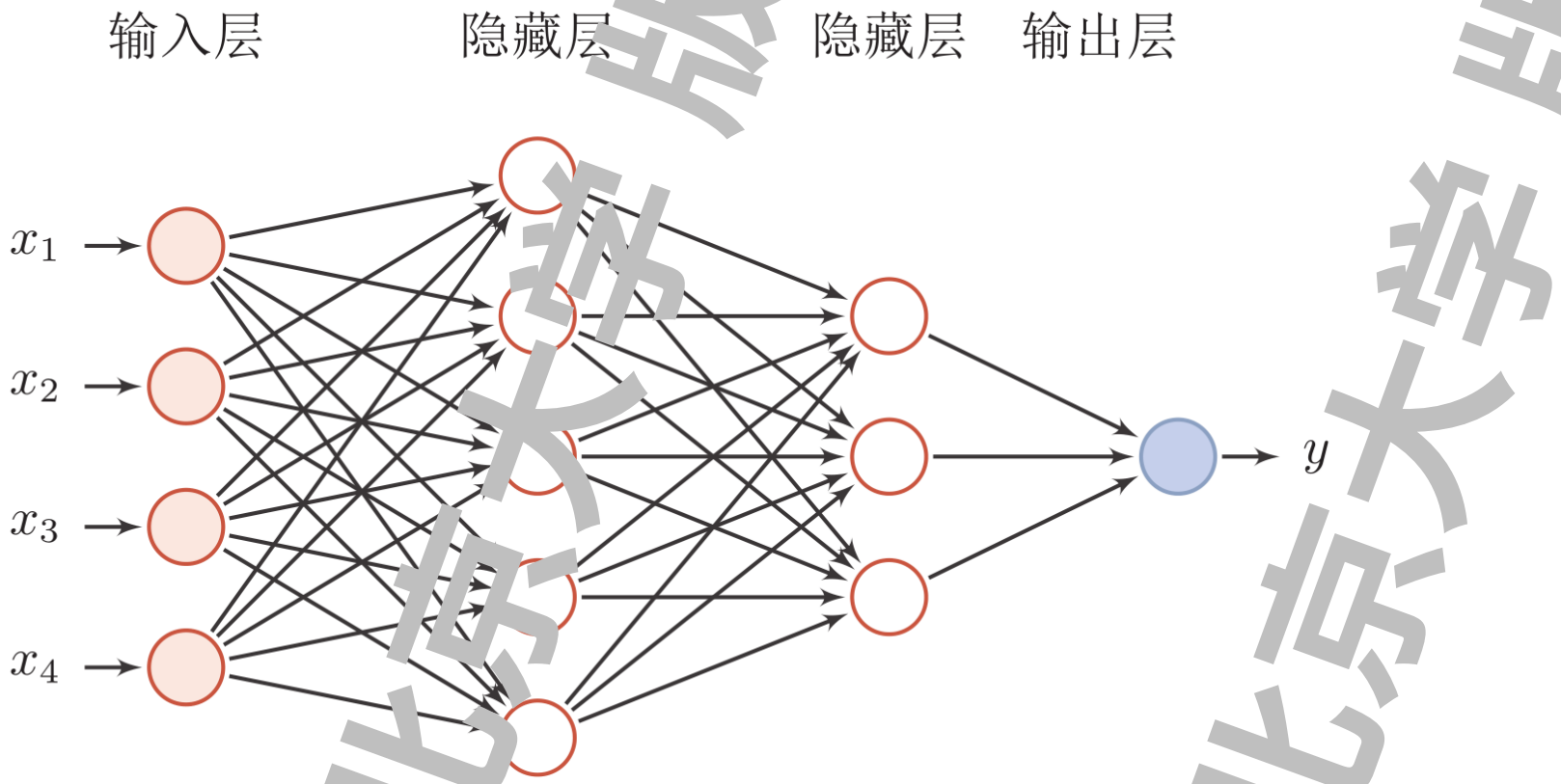


(a) 前馈网络　(b) 反馈网络　(c) 图网络

前馈神经网络

- **网络结构**

- 在前馈神经网络中，**各神经元分别属于不同的层**
- 整个网络中无反馈，信号从输入层向输出层单向传播，**可用一个有向无环图表示**



前馈神经网络

- 通用近似定理

- 只要其隐藏层神经元的数量足够，它可以以任意精度来近似任何从一个定义在实数空间中的有界闭集函数。

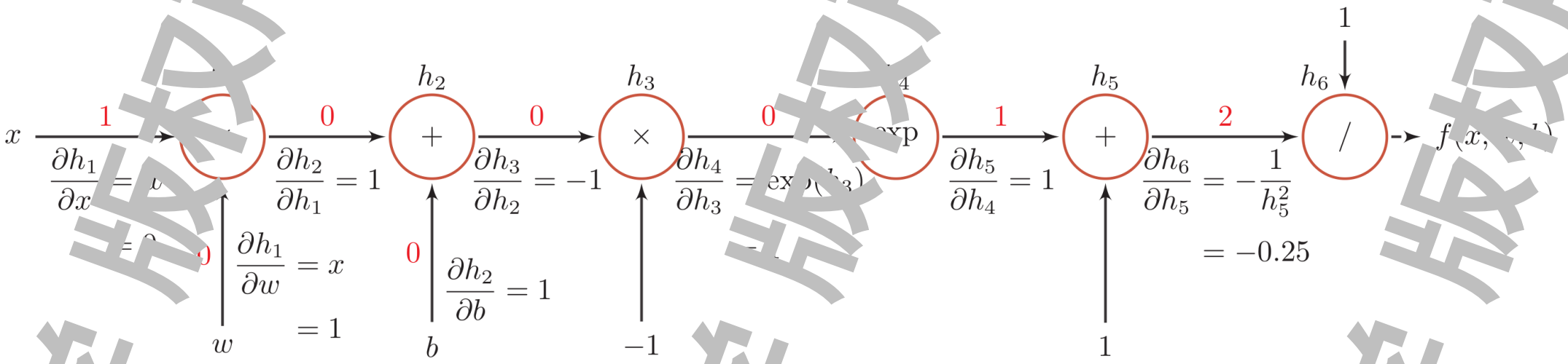
  - 模型
    - $y = f^5(f^4(f^3(f^2(f^1(x)))))$
  - 学习准则
    - $\mathrm{L}(y, y^*)$
  - 优化
    - 梯度下降

链式法则，可以自动计算！

$$\frac{\partial \mathrm{L}(y, y^*)}{\partial f^1} = \frac{\partial f^2}{\partial f^1} \times \frac{\partial f^3}{\partial f^2} \times \frac{\partial f^4}{\partial f^3} \times \frac{\partial f^5}{\partial f^4} \times \frac{\partial \mathrm{L}(y, y^*)}{\partial f^5}$$

计算图与自动微分

- 复合函数f(x; w, b) = σ(wx + b)的计算图



链式法则

$$\frac{\partial f(x;w,b)}{\partial w} = \frac{\partial f(x;w,b)}{\partial h_6}\frac{\partial h_6}{\partial h_5}\frac{\partial h_5}{\partial h_4}\frac{\partial h_4}{\partial h_3}\frac{\partial h_3}{\partial h_2}\frac{\partial h_2}{\partial h_1}\frac{\partial h_1}{\partial w}$$

反向传播算法只是自动微分的一种特殊形式

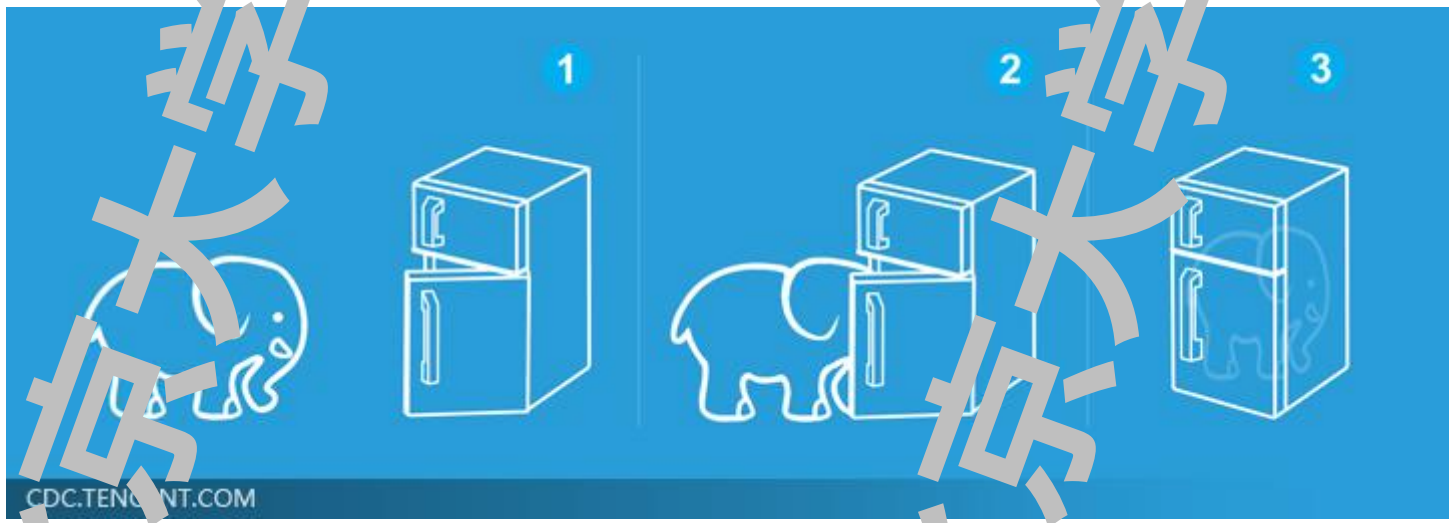
深度学习的三个步骤

1 定义网络

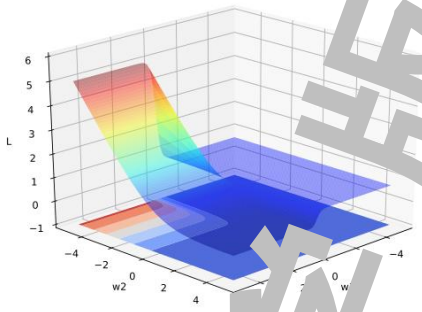
2 损失函数

3 优化

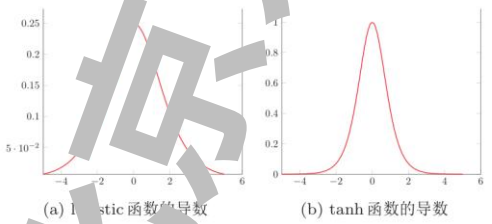
Deep Learning is so simple ……



- 非凸优化问题
  - y = σ(w 2 σ(w 1 x)) 的损失函数



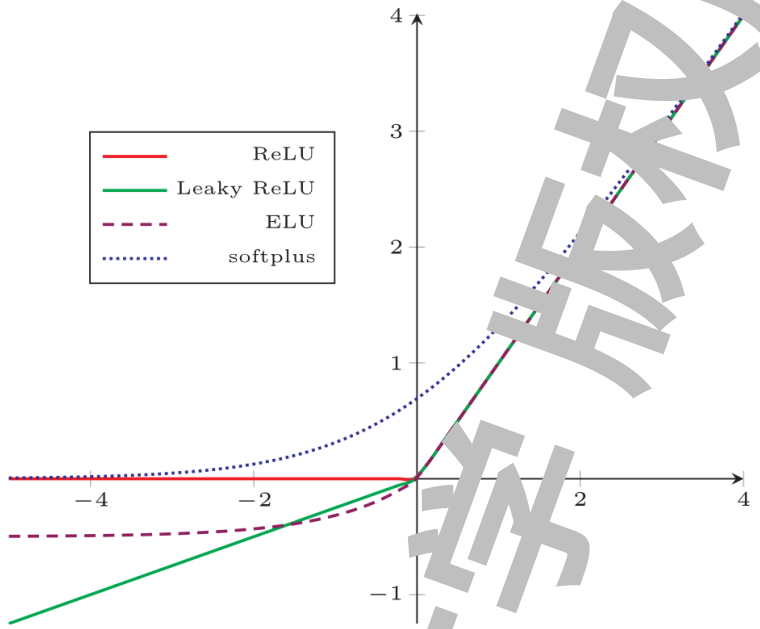
- 梯度消失问题
  - 在每一层都要乘以该层的激活函数的导数



激活函数

有效减轻梯度消失问题！

激活函数	函数	导数
Logisti 函数	$f(x)=\frac{1}{1+e^{-x}}$	$f'(x)=f(x)($ $f(x))$
Tann 函数	$f(x)=\frac{2}{1+e^{-2x}}-1$	$f'(x)=1-$ $(x)^2$
ReL	$f(x)=\max(0,x)$	$f'(x)=I(x>0)$
EL	$f(x)=\max(0,x)+$ $\min(0,\gamma(\exp(x)-1))$	$f'(x)=$ $(x>0)+$ $I(x\leq 0)\cdot\gamma\exp(x)$
o Plus 函数	$f(x)=\ln(1+e^x)$	$f'(x)=\frac{1}{1+e^{-x}}$

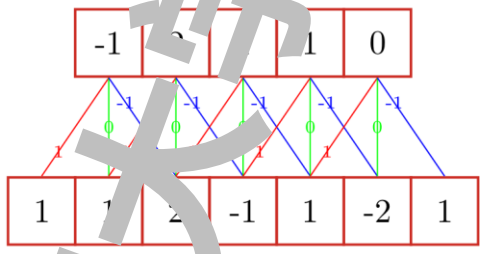


典型神经网络模型：卷积神经网络

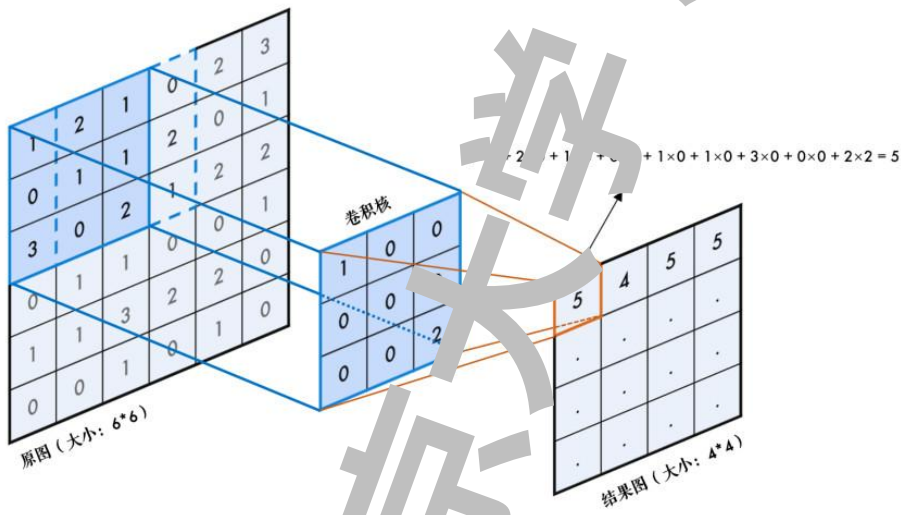
- 卷积经常用在信号处理中，用于计算信号的延迟累积。
- 假设一个信号发生器每个时刻t产生一个信号x_t，其信息的衰减率为w_k，即在k−1个时间步长后，信息为原来的w_k 倍
  - 假设$w_1 = 1, w_2 = 1/2, w_3 = 1/4$
- 时刻t收到的信号y_t 为当前时刻产生的信息和以前时刻延迟信息的叠加

$$y_t = 1 \times x_t + 1/2 \times x_{t-1} + 1/4 \times x_{t-2}$$
$$= w_1 \times x_t + w_2 \times x_{t-1} + w_3 \times x_{t-2}$$
$$= \sum_{k=1}^{3} w_k \cdot x_{t-k+1}.$$

滤波器（filter）或卷积核（convolution kernel）



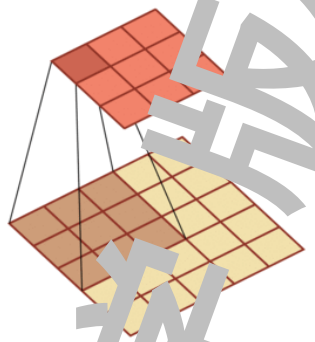
滤波器：[1, 0, -1]



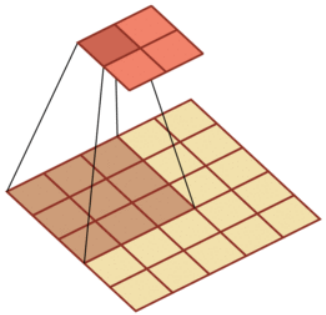
二维卷积

典型神经网络模型：卷积神经网络

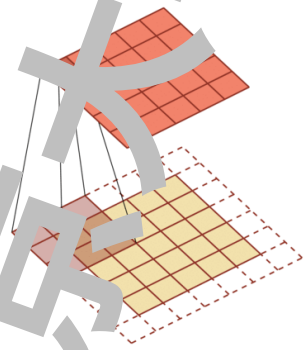
- **二维卷积：在数据的两端填充零的技术，它的主要目的是使卷积后的图像大小不变，从而方便后续的操作。当卷积核的大小大于输入图像的大小时，零填充就变得特别重要，因为它能够避免图像的尺寸减小**



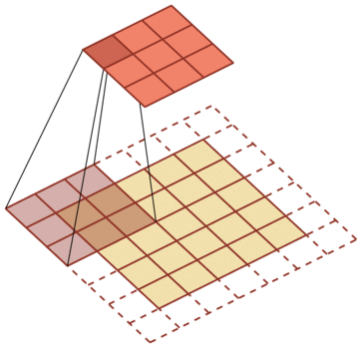
步长1，零填充0



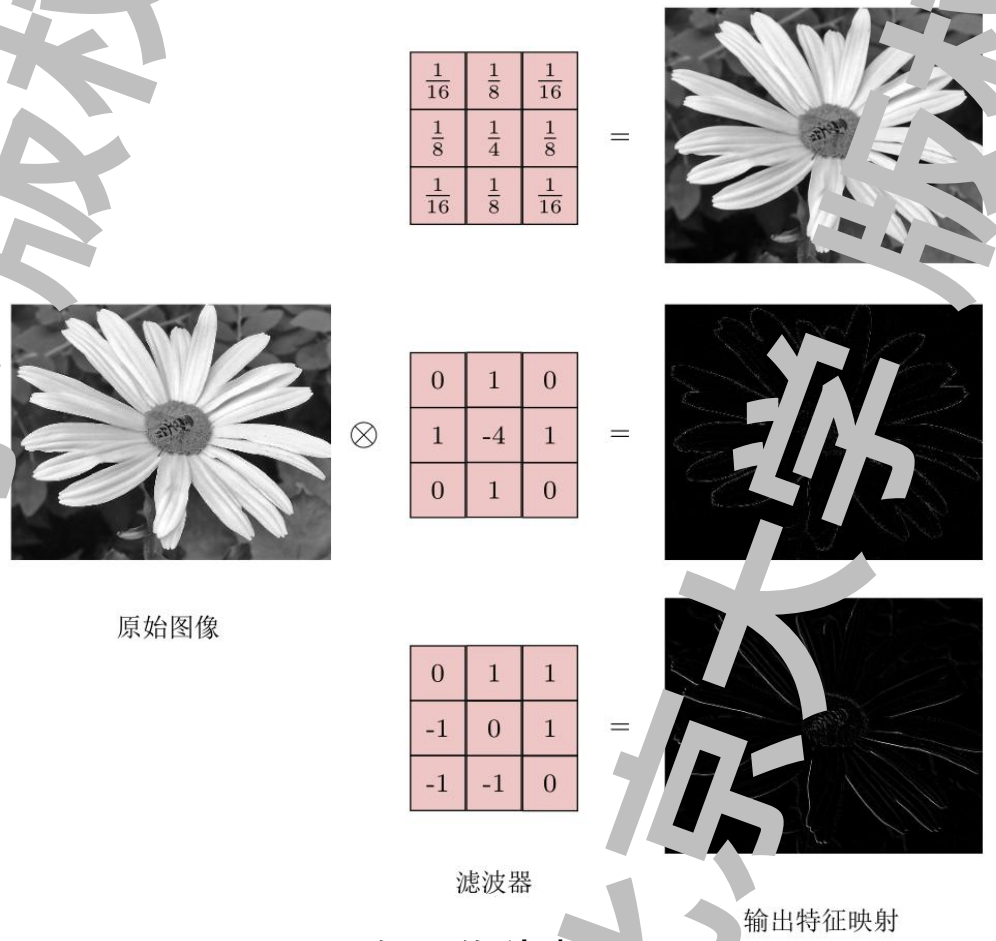
步长2，零填充0



步长1，零填充1



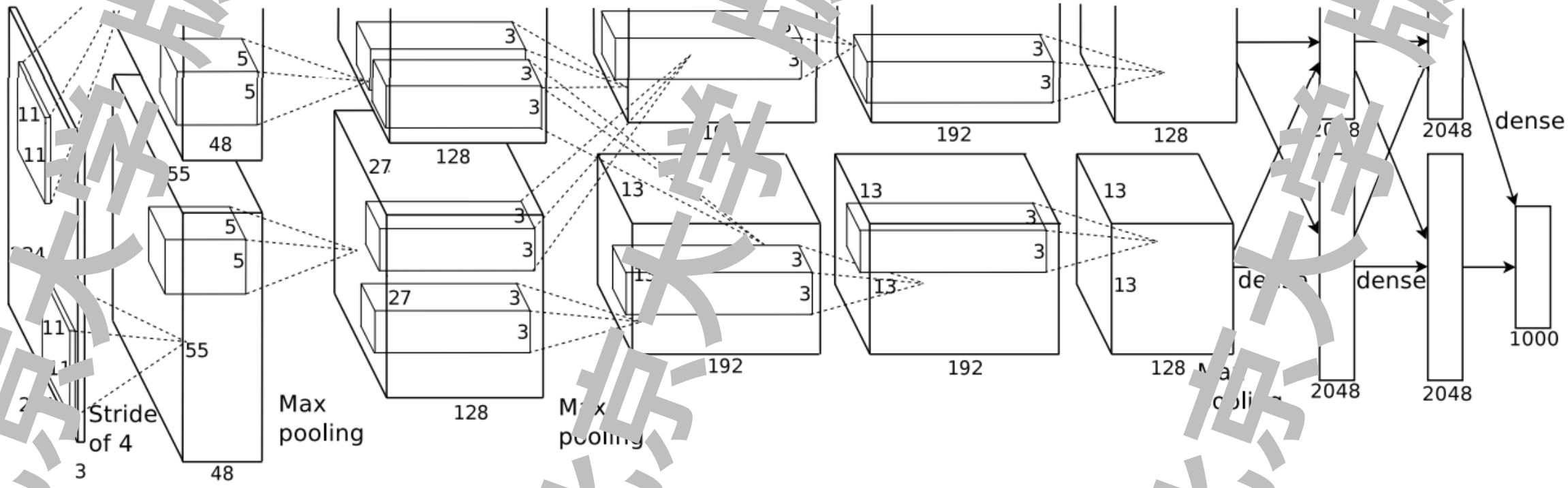
步长2，零填充1



卷积作为特征提取器

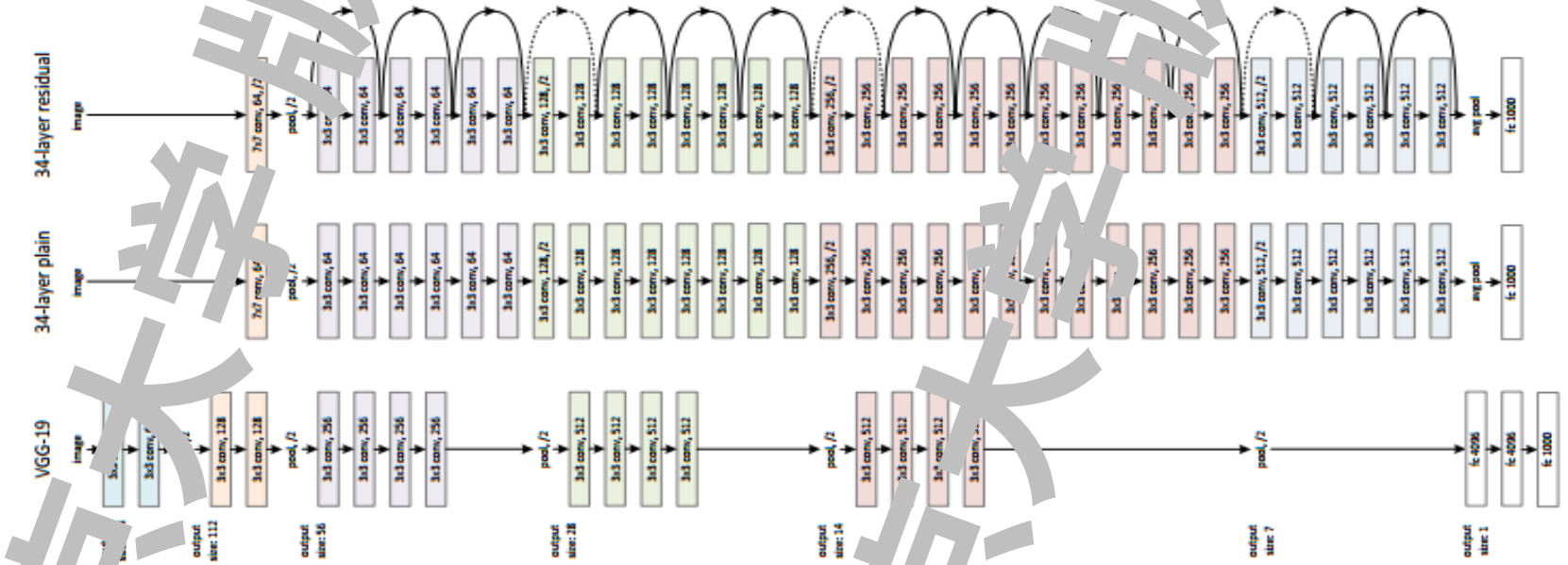
典型神经网络模型：卷积神经网络

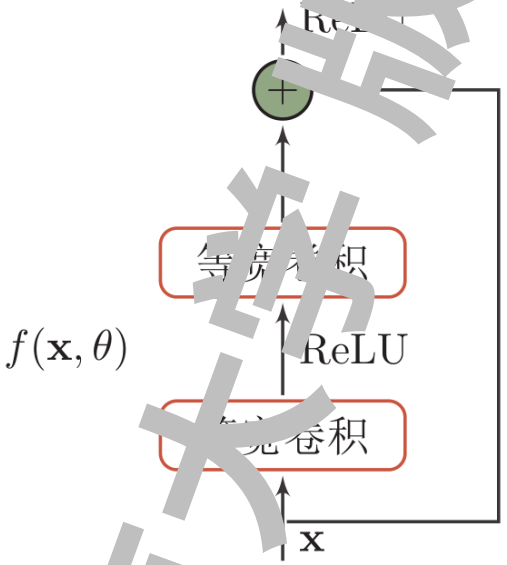
- AlexNet、2012 ILSVRC winner
- （top 5 error of 16\% compared to runner-up with 26\% error）
- 共有8层，其中前5层卷积层，后边3层全连接层



典型神经网络模型：卷积神经网络

- ResNet、2015 ILSVRC winner （152层）
- 错误率：3.57%



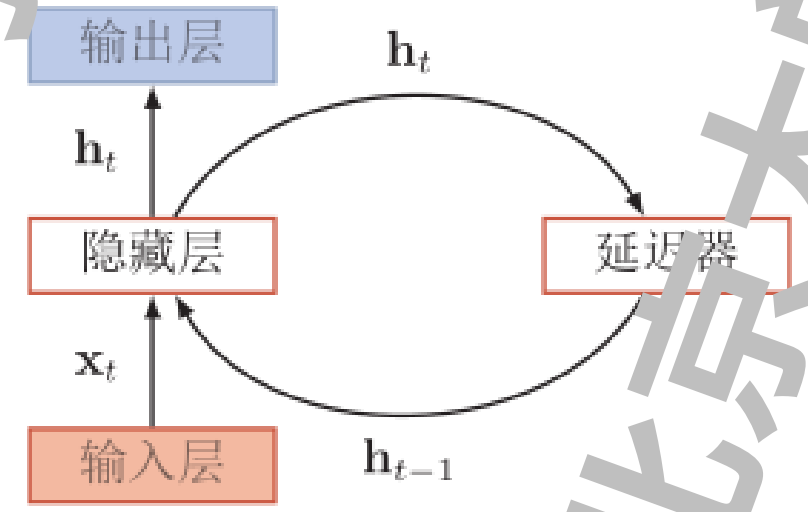


典型神经网络模型：循环神经网络

- 循环神经网络通过使用带自反馈的神经元，能够处理任意长度的序列。
- 循环神经网络比前馈神经网络更加符合生物神经网络的结构。
- 循环神经网络已经被广泛应用在语音识别、语言模型以及自然语言生成等任务上。

对于一个输入序列 $\mathbf{x}_{1:T} = (\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_t, \ldots, \mathbf{x}_T)$，循环神经网络通过下面公式更新带反馈边的隐藏层的活性值 $\mathbf{h}_t$：

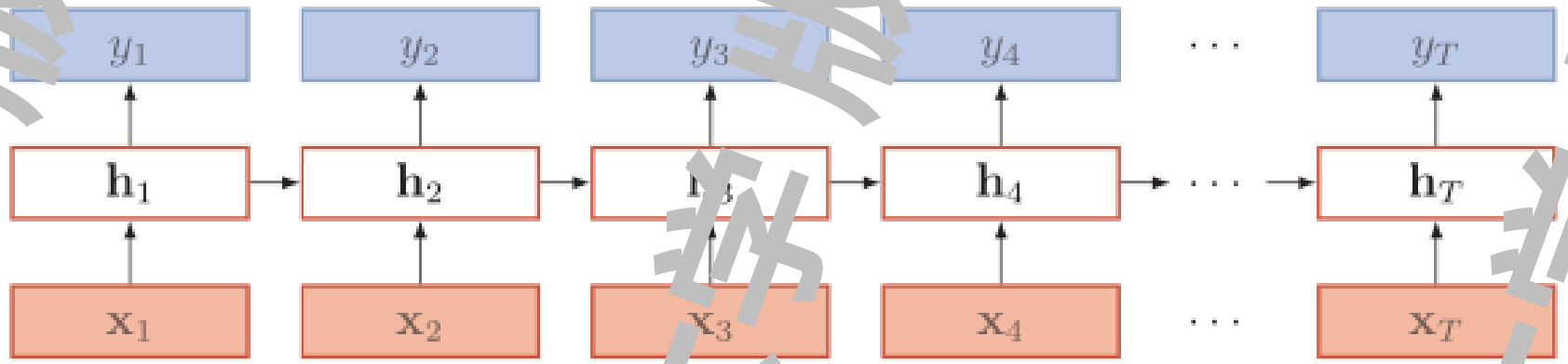
$$\mathbf{h}_t = \begin{cases} 0 & t = 0 \\ f(\mathbf{h}_{t-1}, \mathbf{x}_t) & \text{otherwise} \end{cases}$$



典型神经网络模型：循环神经网络

- **状态更新：**

$$\mathbf{h}_t = f(U\mathbf{h}_{t-1} + W\mathbf{x}_t + \mathbf{b}),$$

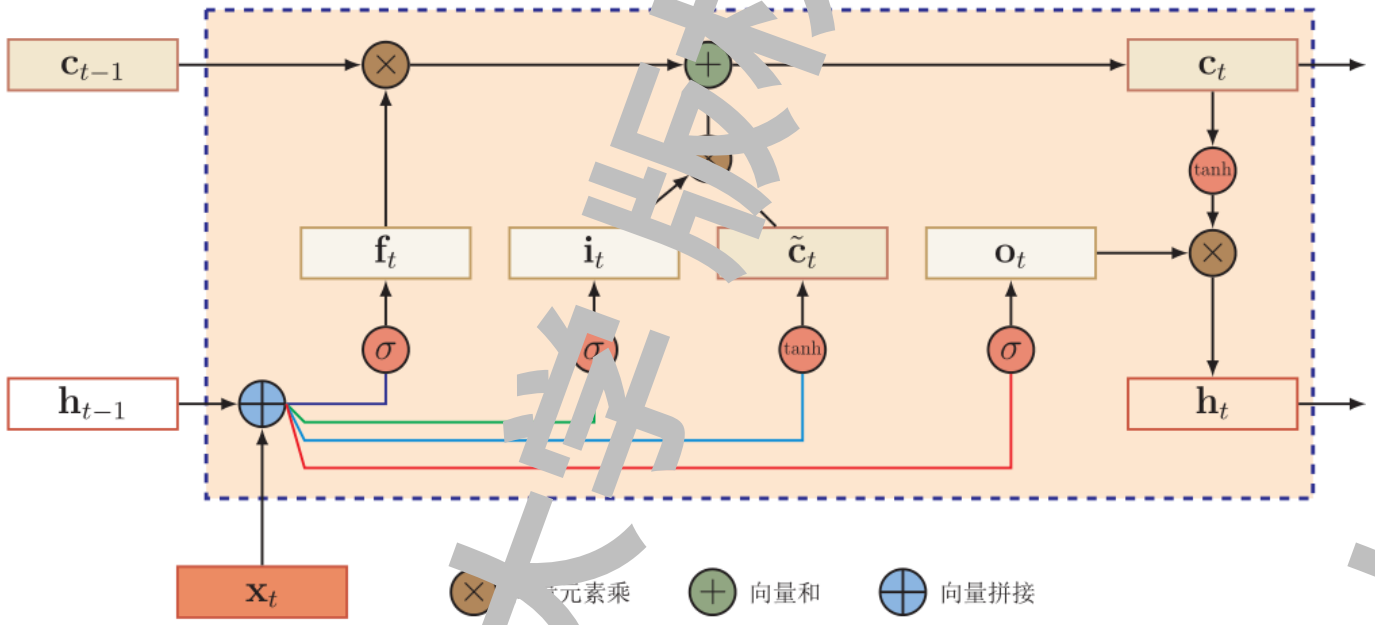


RNN是图灵完全等价的 (Siegelmann and Sontag, 1995)

FNN：模拟任何函数
RNN：模拟任何程序（计算过程）。

典型神经网络模型：长短时记忆神经网络LSTM

- 循环神经网络在时间维度上非常深！
- 梯度消失或梯度爆炸



$$\mathbf{i}_t = \sigma(W_i\mathbf{x}_t + U_i\mathbf{h}_{t-1} + \mathbf{b}_i),$$

$$\mathbf{f}_t = \sigma(W_f\mathbf{x}_t + U_f\mathbf{h}_{t-1} + \mathbf{b}_f),$$

$$\mathbf{o}_t = \sigma(W_o\mathbf{x}_t + U_o\mathbf{h}_{t-1} + \mathbf{b}_o),$$

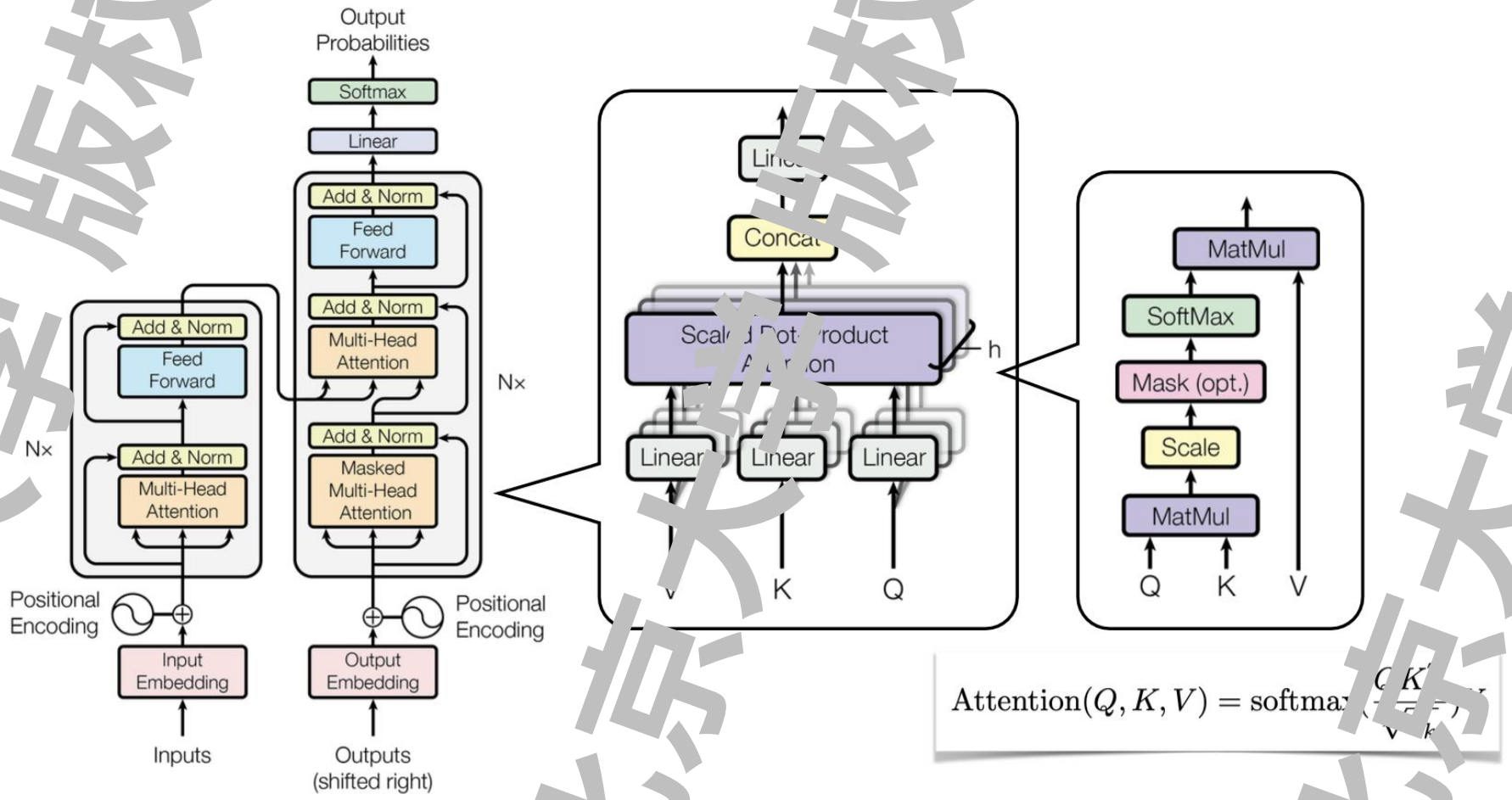
$$\tilde{\mathbf{c}}_t = \tanh(W_c\mathbf{x}_t + U_c\mathbf{h}_{t-1} + \mathbf{b}_c)$$

$$\mathbf{c}_t = \mathbf{f}_t \odot \mathbf{c}_{t-1} + \mathbf{i}_t \odot \tilde{\mathbf{c}}_t,$$

$$\mathbf{h}_t = \mathbf{o}_t \odot \tanh(\mathbf{c}_t),$$

典型神经网络模型：自注意力机制神经网络Transformer

- RNN的问题：处理长序列时，信息会丢失（多个输入序列xi ->单个向量c）
- Transformer：基于注意力机制，抛弃卷积，模仿人类视觉处理信息时的选择性关注视点

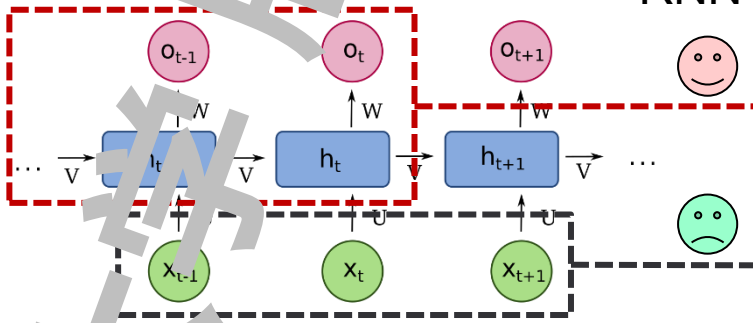


典型神经网络模型：自注意力机制神经网络Transformer

- Transformer模型的优势

指标	RNN	CNN	Transformer
数据特征	有序序列	多维网格	一组向量
长序列处理	✓	X	✓
可并行性	X	✓	✓

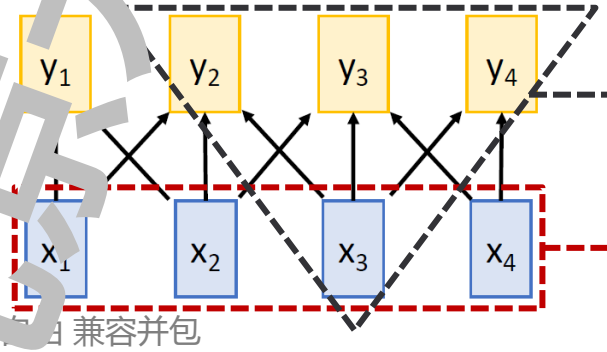
RNN



可以得到前面状态的信息

不能同步输入

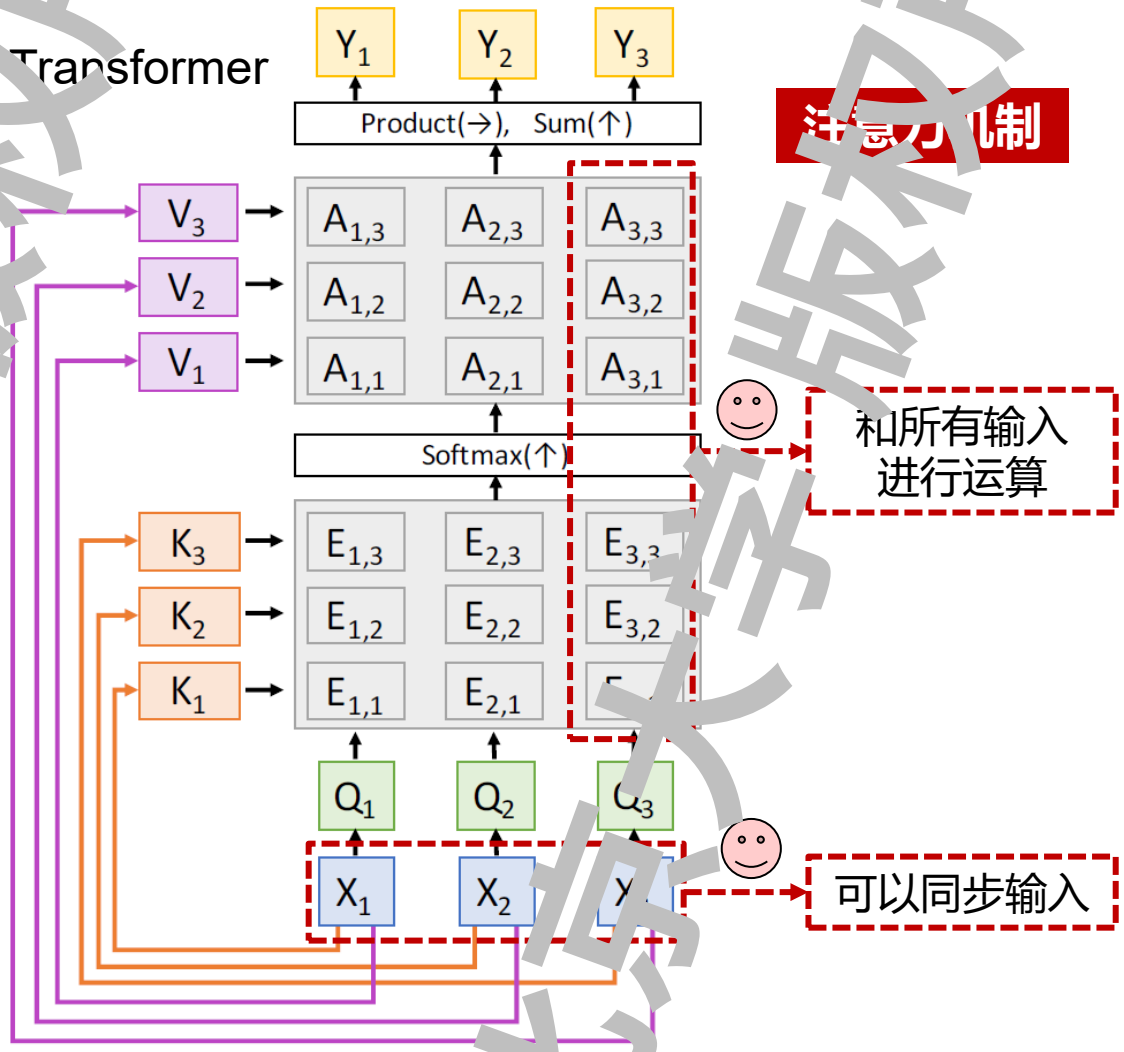
CNN



不能得到全局信息

可以同步输入

Transformer



注意力机制

和所有输入进行运算

可以同步输入

典型神经网络模型：自注意力机制神经网络Transformer

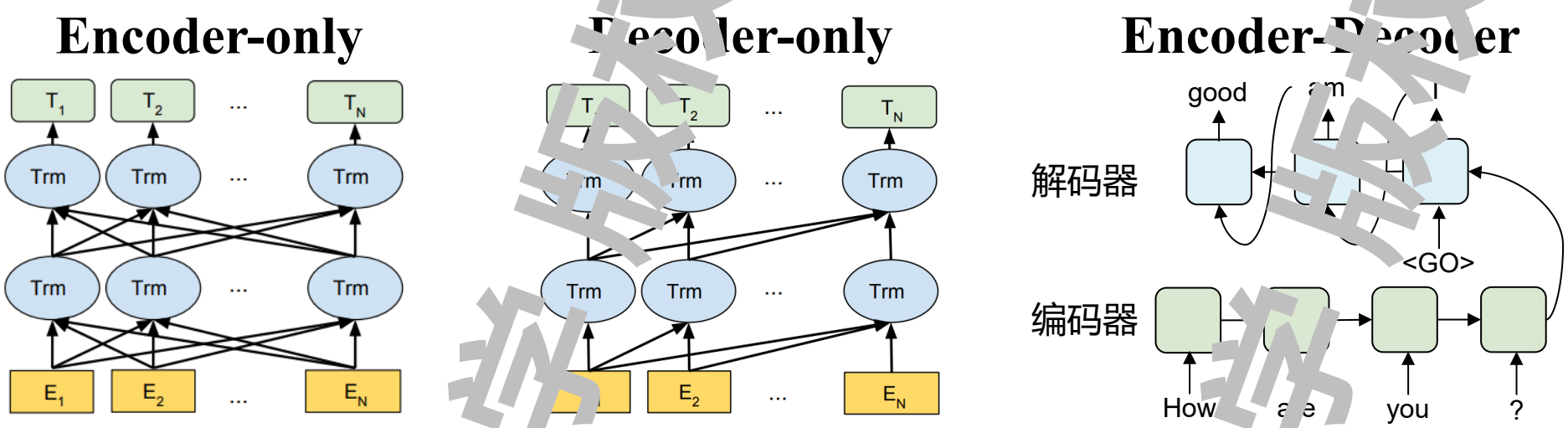
- 根据网络结构分为：Encoder-only；Decoder-only；Encoder-Decoder

低延时要求

输入依赖于上一次的输出

流水并行设计

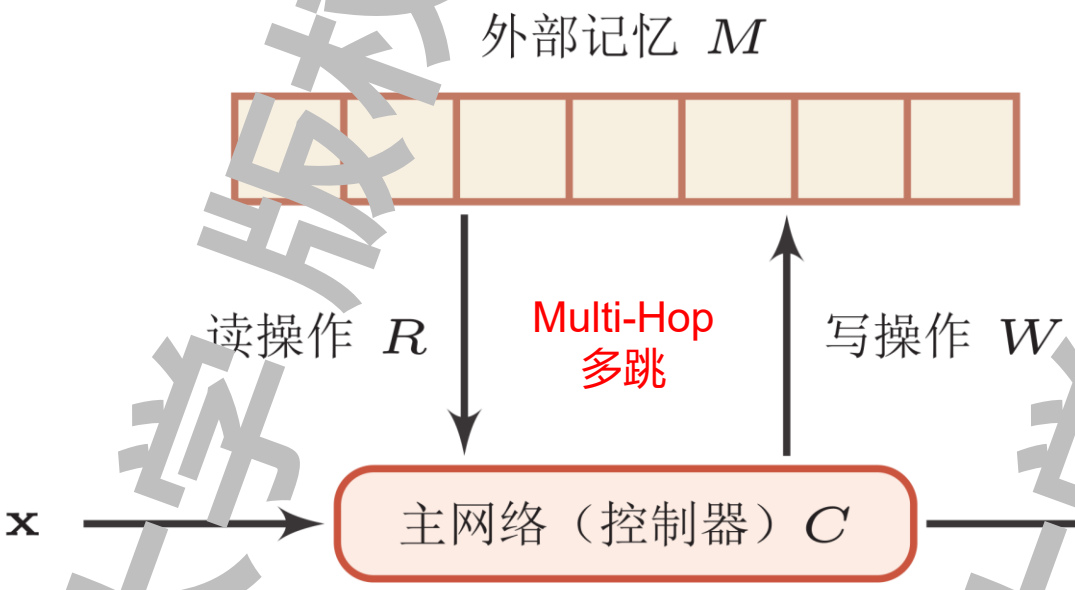
分阶段并行设计

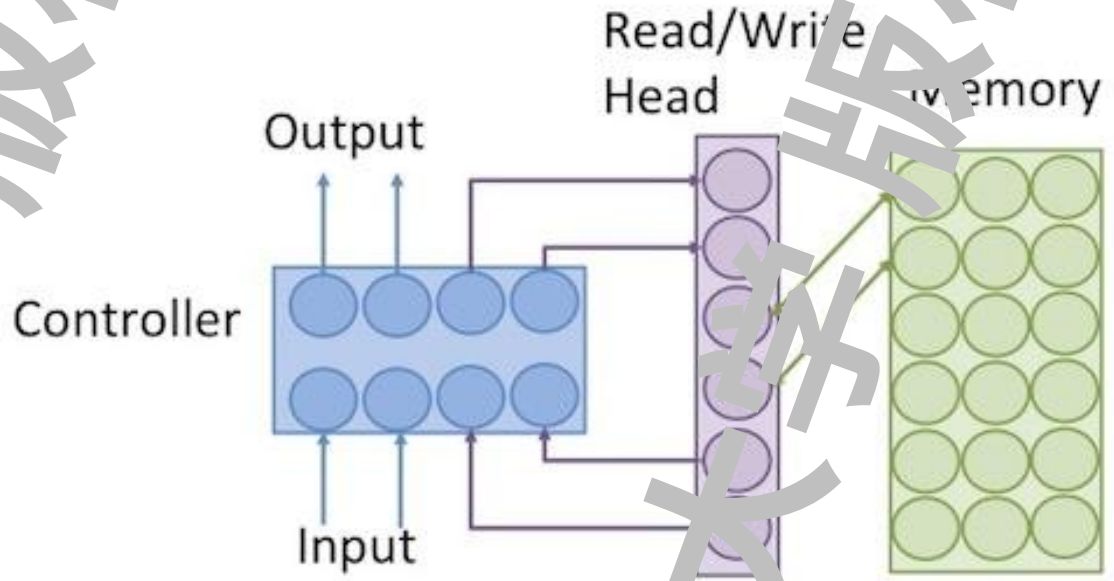


常见模型	BERT，RoBERTa	GPT，LLAMA	T5，BART
适合任务	分类任务	问答任务	翻译任务
输入输出特征	所有token并行输入	输入依赖于上一次的输出	输出强依赖于输入
适配加速器特征	**高吞吐**	**低延时**	**低延时**
适配并行设计	**流水并行设计**	**分阶段并行设计**	**分阶段并行设计**

典型神经网络模型：记忆增强神经网络NTM、DNC

- 神经图灵机、微分神经计算机等

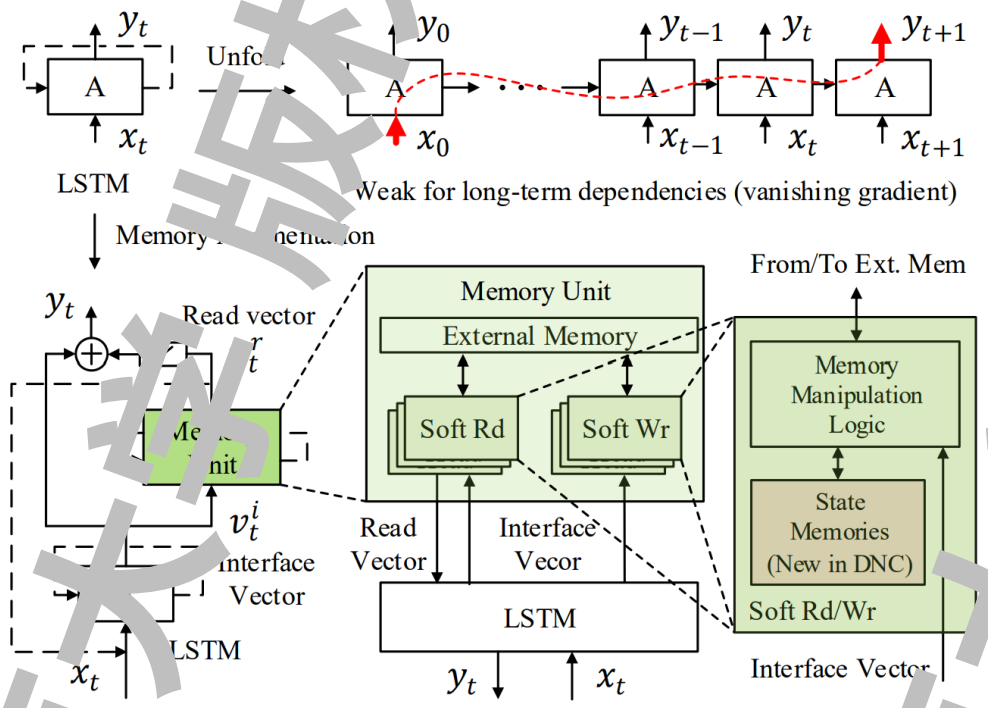


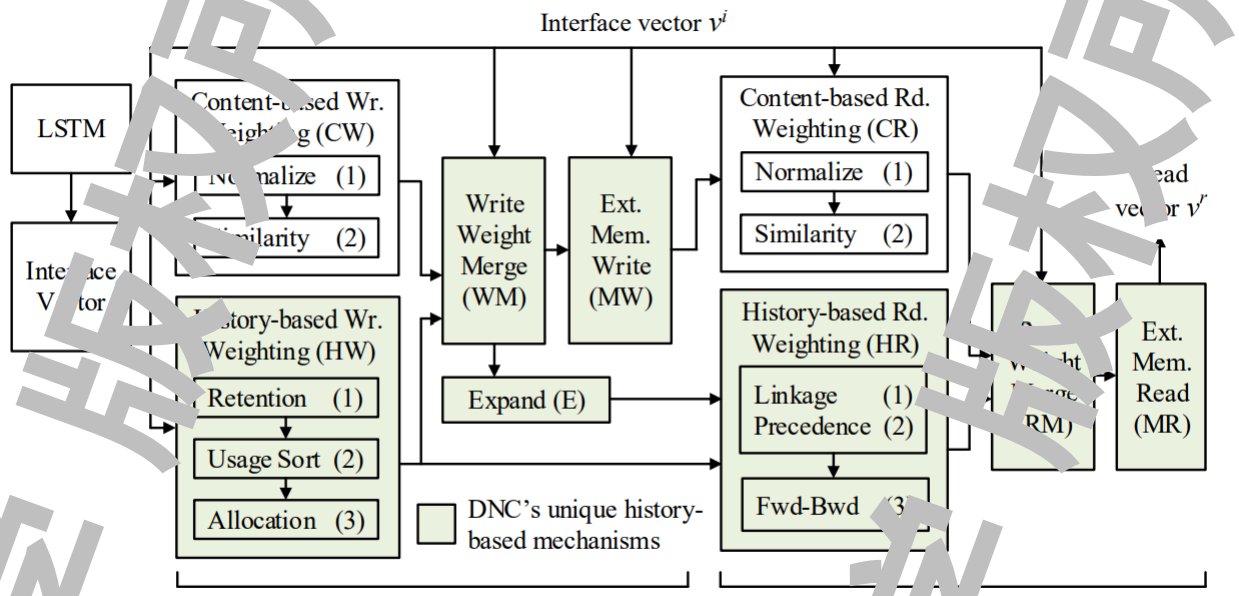


按内容寻址：通常利用注意力机制来完成。

典型神经网络模型：记忆增强神经网络NTM、DNC

- 神经图灵机、微分神经计算机等





DNC Soft Write

CW.(1): $||\mathrm{M}[i,\cdot]||\leftarrow\mathrm{M}[i,\cdot]\ \ ||k^w||\leftarrow k^w$

CW.(2): $w^u\leftarrow\mathrm{softmax}(s^w||\mathrm{M}[i,\cdot]||\cdot||k^w||)$

HW.(1): $\psi[i]=\prod_{r=1}^{R}(1-\mathrm{g}^f[r]w^r[i,r])$

HW.(2): $\mathrm{I_s}\leftarrow\mathrm{sort}\,(u+w^w\text{-}u\circ w^w)\circ\psi$

HW.(3): $w^a[\mathrm{I_s^i}]=(1\text{-}u[\mathrm{I_s^i}])\prod u[\mathrm{I_s^{1\to i}}]$

WM: $w^w=\mathrm{g}^w(\mathrm{g}^a w^a+(1\text{-}\mathrm{g}^a w^u))$

E: $w[\cdot,i]=w^w$

MW: $\mathrm{M}\leftarrow\mathrm{M}\circ(E\text{-}w^w(v^e)^{\mathrm{T}})+w^w(v^w)^{\mathrm{T}}$

DNC Soft Read

CR.(1): $||\mathrm{M}[i,\cdot]||\leftarrow\mathrm{M}[i,\cdot]||\cdot||k^r||\leftarrow k^r$

CR.(2): $r^u\leftarrow\mathrm{softmax}(s^r||\mathrm{M}[i,\cdot]||\cdot||k^r||)$

HR.(1): $\mathrm{L}=\{(\quad w\text{-}w^{\mathrm{T}})\circ(\quad)w^w p^{\mathrm{T}}\}\circ(E-I)$

HR.(2): $=(1-\sum_{i=1}^{N}w^w[i])p+w^w$

HR.(f^r L $b-\mathrm{L}^{\mathrm{T}}w^r$

R $=r_1 b$ $r^u+m_3^r f^r$

MR: $v^r=\mathrm{M}^{\mathrm{T}}$

北京大学
PEKING UNIVERSITY

Attention is All You Need

Acknowledgement to 李萌

语言模型为序列打分

- 语言模型为一个 token 序列赋予一个概率

$$p(x_1, x_2, \ldots, x_t)$$

高概率

"the mouse ate the cheese"

语法正确、语义合理。

低概率

"the cheese ate the mouse" ✗

语法错误或语义不合理。

概率 p 不只是一个数字。要给出有意义的概率，模型必须隐式地学会语法、语义和世界知识。

简史：熵与预测

语言建模源于信息论：好的预测意味着好的压缩。

$$H(p) = \sum_{x} p(x) \log \frac{1}{p(x)}$$

直觉

- 常见事件需要更少的比特。
- 意外事件需要更多的比特。
- 有结构的语言比随机符号更可预测。

简单示例

```
"the mouse ate the cheese"
→ 更短的编码

"mouse the the cheese ate"
→ 更长的编码
```

这把语言建模归结为一个简单的问题：我们能多好地预测下一个符号或单词？

n-gram 语言模型

n-gram 模型仅利用前 n-1 个 token 来预测下一个 token。

$$p(x_i \mid x_{1:i-1}) \approx p(x_i \mid x_{i-n+1:i-1})$$

三元组示例（n=3）

$$p(\text{cheese} \mid \text{the mouse ate the}) \approx p(\text{cheese} \mid \text{ate the})$$

通过计数估计概率

$$p(x_i \mid x_{i-1}) = \frac{\text{count}(x_{i-1}, x_i)}{\text{count}(x_{i-1})}$$

在 token x_{i-1} 出现的所有次数中，统计后跟着 x_i.

n-gram 模型计算代价很低
其概率来自大规模语料中局部模式的计数。

n-gram 模型的局限

示例上下文：Stanford has a new course on large language models. It will be taught by ___

1. 固定的局部上下文

语料可以很大，但每次预测只用到最后 $n-1$ 个 token。

$$p(x_i \mid x_{1:i-1}) \approx p(x_i \mid x_{i-n+1:i-1})$$

当 n 很小时，**Stanford** 落在局部上下文之外，无法影响预测。

2. n 较大时计数稀疏

增大 n 能提供更多上下文，但完全相同的长上下文很少重复出现。

$$\text{count (long exact context)} \approx 0$$

由于可供计数的证据太少，基于计数的估计会变成零或不可靠。

核心权衡：n 较小会丢失长程信息；n 较大会让基于计数的估计在统计上变得稀疏。

神经语言模型

用神经网络来建模 token 序列上的概率。

$p(x_1, x_2, \ldots, x_L)$ where p_θ = neural network with parameters θ

文本序列 $x_1, x_2, \ldots, x_L$ → 输入 → 神经网络 $p_\theta(\cdot)$

输入是啥

- 序列记为 $x_1, x_2, \ldots, x_L$
- 每个 x_i 最初都是一段文本

关键障碍

- **神经网络不能直接以原始单词作为输入。**
- **输入必须是数值张量。**

下一个问题：如何把文本变成神经网络可以处理的数字？

从词到 Token

在进入神经网络之前，文本必须先变成数字。

原始文本 → token → token ID → 向量

1. 文本分词（tokenize）

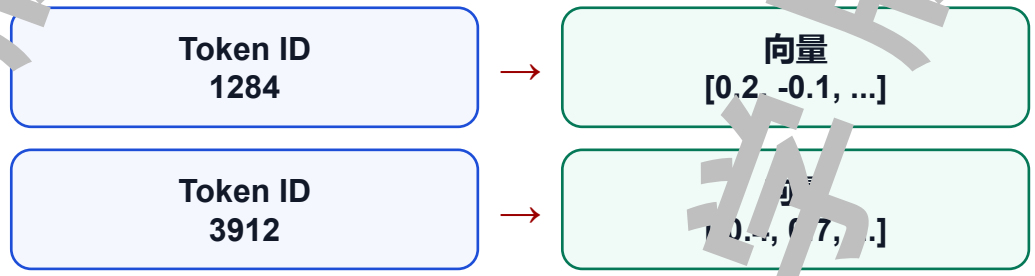
原始文本：
"Large language models are unbelievable."

Token：
["La" "ge" " language", ..., " un", "believ", "able", "."]

Token ID：
[[illegible], 3912, ..., 918, 4410, 982, 271]

token 可以是一个词、词的一部分或标点符号

2. 查询嵌入（embedding）



每个 ID 从嵌入表中选出一行

$$E[\text{token IDs}] \rightarrow X \in \mathbb{R}^{L \times d}$$

现在模型得到一个张量 X：每个 token 位置对应一个向量。

从张量输入到模型架构

现在输入已经是张量，神经网络可以处理它了。

输入张量
$x \in \mathbb{R}^{L \times d}$

→

神经网络
p_θ

→

序列概率
$p_\theta(x_1, \ldots, x_L)$

关键问题：p_θ 应该采用什么结构？

接下来：模型架构。

北京大学
PEKING UNIVERSITY

从语言模型到模型架构

打开模型内部，介绍编码器、解码器以及 decoder-only Transformer。

Acknowledgement to 李萌

为什么需要架构？

我们已有张量和神经网络，缺的是网络结构。

$$X \in \mathbb{R}^{L\times d} \xrightarrow{\text{architecture } F_\theta} H \xrightarrow{\text{probability head}} p_\theta(x_1, \ldots, x_L)$$

F_θ 必须做什么

- 读入一串 token 向量。
- 让每个 token 利用相关的上下文。
- 产生带上下文的表示 H.

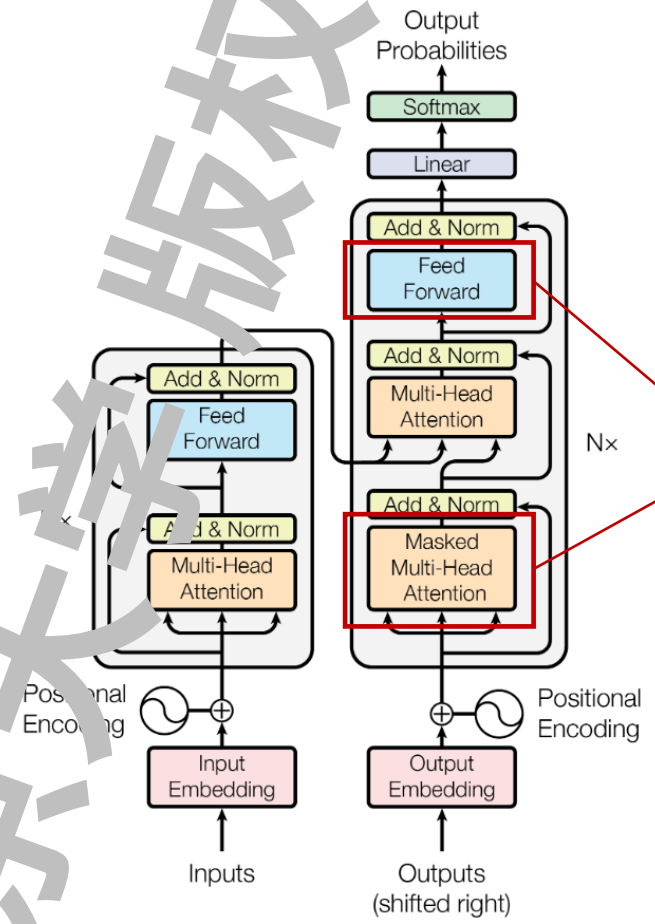
难点在哪里

- **上下文可能很长。**
- **不同 token 依赖于不同的其他 token。**
- **训练必须能扩展到海量数据。**

架构决定了信息如何在 token 之间流动。

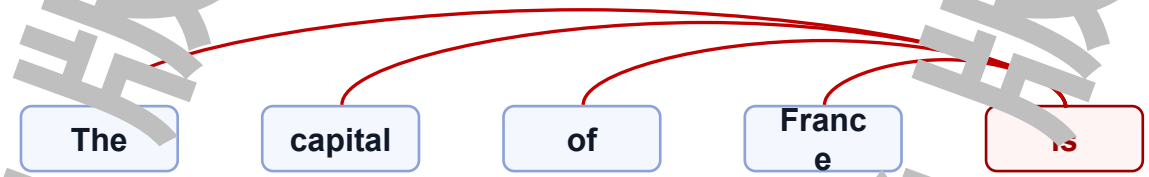
Transformer：Attention Is All You Need

一个 Transformer 模块内部



关键部分：注意力

注意力：建模 token 之间的信息



对于 token "is" 而言，有用的上下文可能包括 "capital" 和 "France"。

FFN：建模通道之间的信息

三类 Transformer 架构

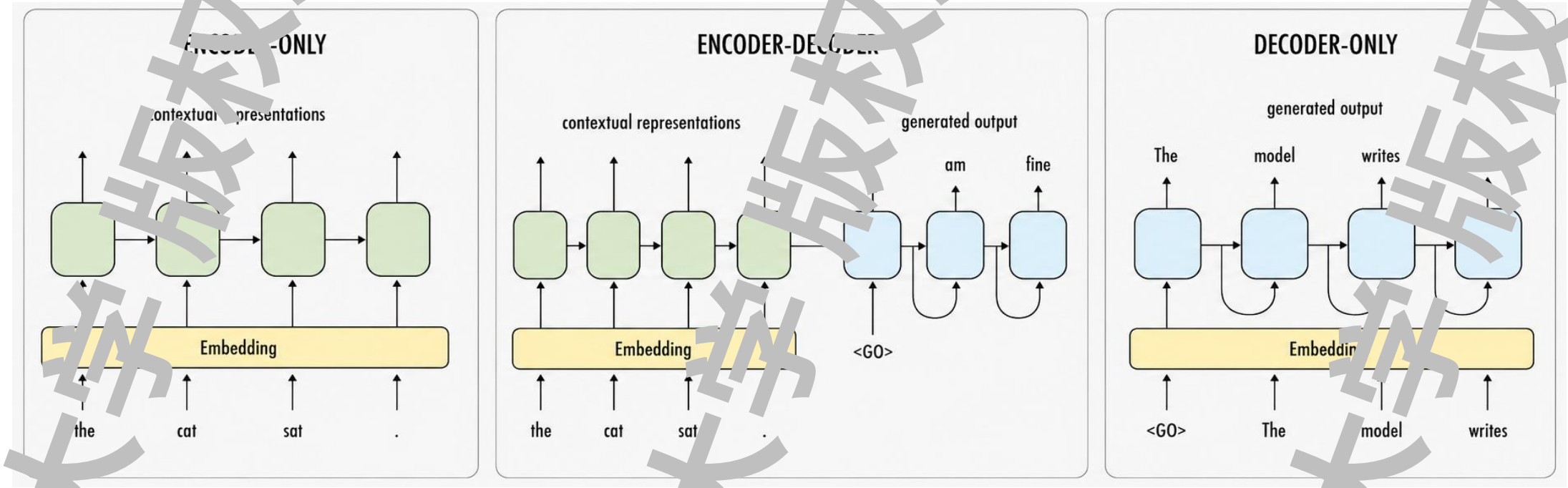
不同类型的 Transformer 适用于不同的语言任务。

类型	典型用途	代表模型
仅编码器（Encoder-only）	理解完整输入：分类、检索、掩码词预测。	BERT, RoBERTa
编码器-解码器（Encoder-Decoder）	将一个序列映射为另一个序列：翻译、摘要、文本到文本任务。	T5, BART
仅解码器（Decoder-only）	续写序列：生成、对话、代码、推理。	GPT, LLaMA, Qwen, DeepSeek

编码器负责读，解码器负责写。

三种 Transformer 架构模式

示意图展示了每种架构天然擅长的任务。



读取完整输入，输出一个标签或表示。

任务：分类 / 填空

读取源序列，然后逐个 token 生成目标序列。

任务：翻译 / 摘要

生成一个 token，追加到序列后，再生成下一个。

任务：续写 / 对话

Decoder-only：自回归生成

decoder-only 语言模型每次生成一个 token，逐步延长文本。

生成循环

- 从提示词 / 前缀开始。
- 预测下一个 token 的分布。
- 选择或采样一个 token。
- 追加到序列后并重复。

```
提示词:
The capital of France is

第 1 步:
The capital of France is → Paris

第 2 步:
The capital of France is Paris → .
```

上下文不断增长

```
当前上下文:
The capital of France is

生成:
Paris

新上下文:
The capital of France is Paris
```

核心操作：预测下一个 token。

自回归语言模型：训练与推理

自回归建模把序列概率转化为反复预测下一个 token。

$$p(x_{1:L}) = p(x_1)p(x_2 \mid x_1)\cdots p(x_L \mid x_{1:L-1}) = \prod_{i=1}^{L} p(x_i|x_{1:i-1})$$

训练

对一条文本序列，在多个位置上计算下一个 token 的损失：

```
the → mouse
the mouse → ate
the mouse ate → the
the mouse ate the → cheese
```

概念上按前缀展示；实际实现是并行的。

推理

用同样的下一个 token 预测来续写提示词。

prompt → completion

提示词：
```
the mouse ate
```

补全：
```
the cheese
```

自回归模型的因果掩码

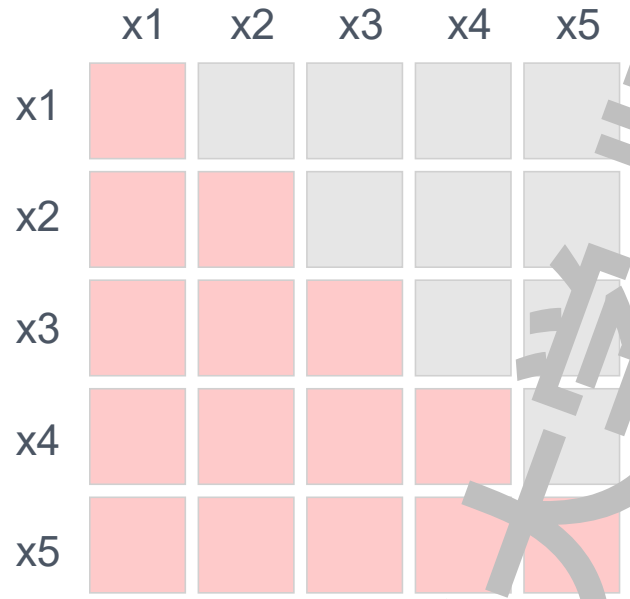
训练时可以并行处理所有 token，但每个位置只能看到过去。

预测规则

```
the → [no st
the mouse  ate
the mous  ate → the
the mouse ate the → cheese
```

遮住未来的 token，让模型无法偷看答案。

因果注意力图



行关注列。红色 = 可见；灰色 = 被遮住的未来。

$$p_\theta(x_{t+1}|x_{\le t})$$

Decoder-only 为何成为主流

ecoder-only 模型把训练、生成和扩展统一到一个简单目标上。

$$\text{large text corpus} \Rightarrow \max_{\theta} \sum_{t} \log p_\theta(x_{t+1}|x_{\le t})$$

数据简单

原始文本本身就提供训练信号：预测每一个下一个 token。无需标注。

生成自然

同一个操作就能把提示词续写成答案、代码或推理过程。

扩展性好

更多的数据、参数和算力一再提升了 decoder-only 语言模型。

因此，现代大模型架构通常就是一叠解码器模块。

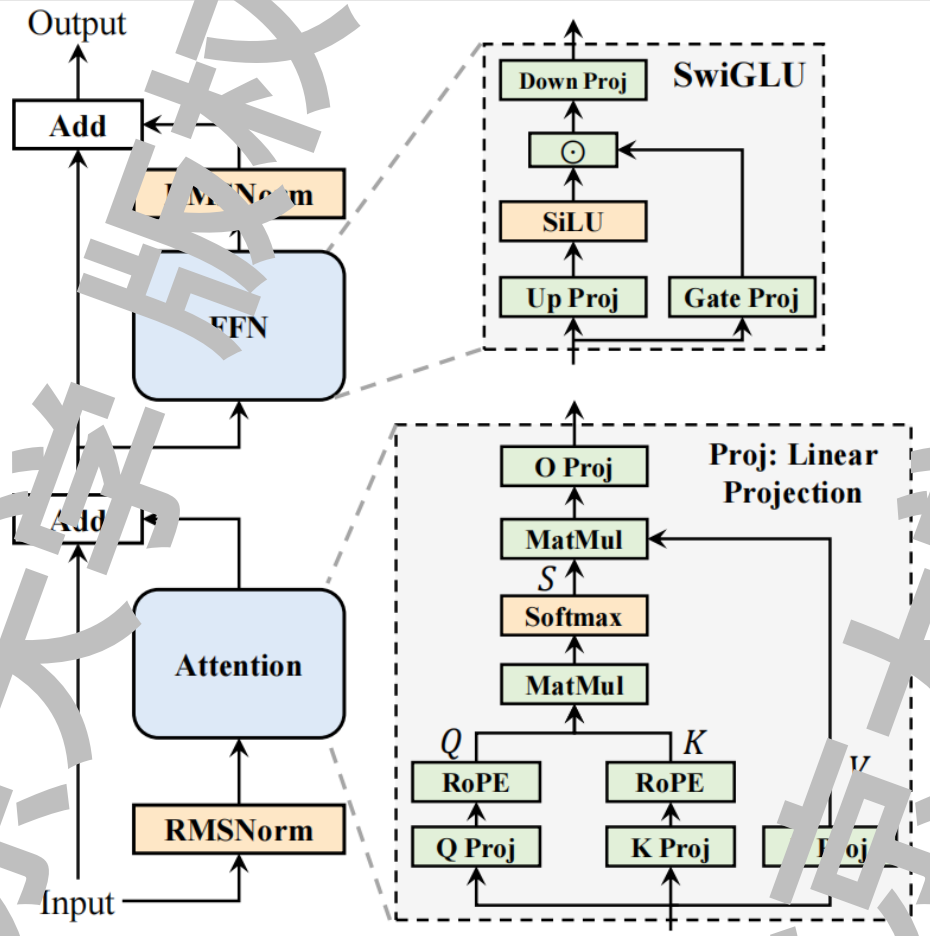
北京大学
PEKING UNIVERSITY

现代大模型架构

一个解码器模块由注意力、MLP、归一化、位置编码组成，有时还包含 MoE。

现代解码器模块

一个解码器模块



各部分的作用

- 注意力：在不同 token 位置之间混合信息。
- FFN / SwiGLU：变换每个 token 的表示。
- RMSNorm：保持激活值在稳定的尺度上。
- 残差相加：保留旧表示的同时加入更新。
- RoPE：向注意力中注入位置信息。

现代大模型主要就是许多这样的解码器模块反复堆叠。

自注意力

注意力让每个 token 自己选择哪些先前的 token 有用。

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

Q：查询（Query）

这个 token 在寻找什么。

K：键（Key）

每个 token 提供什么用于匹配。

V：值（Value）

加权后向前传递的信息。

掩码 M 保证了自回归行为：未来的 token 被屏蔽

FFN 与 SwiGLU

注意力混合 token 之后，FFN 分别变换每个 token。

FFN 做什么

$$\mathrm{FFN}(x) = W_2\sigma(W_1x)$$

- 在每个位置上独立应用。
- 扩展隐藏维度，施加非线性，再投影回来。
- 增加逐 token 的计算能力。

为什么用 SwiGLU?

$$\mathrm{SwiGLU}(x) = W_2(\mathrm{SiLU}(W_gx) \odot W_ux)$$

- 现代大模型常用门控 MLP。
- 门控决定哪些特征可以通过。
- 通常比普通 FFN 效果更好。

归一化与残差连接

很深的解码器堆叠需要稳定的信号流。

RMSNorm

$$\mathrm{RMSNorm}(x) = g \odot \frac{x}{\sqrt{\frac{1}{d}\sum_j x_j^2 + \epsilon}}$$

- 在注意力或 MLP 之前归一化向量尺度。
- 防止激活值过大或过小。

残差相加

$$x_{l+1} = x_l + F_l(\mathrm{Norm}(x_l))$$

- 保留已有信息。
- 让每个模块学习一个修正量。
- 让极深的网络更容易训练。

现代大模型常用 pre-norm：先归一化，再应用注意力或 MLP。

位置编码与 RoPE

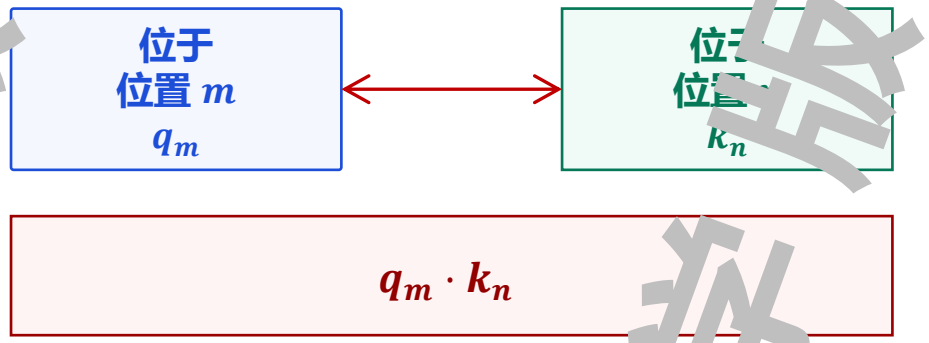
注意力需要位置信息：同样的词以不同顺序排列，含义不同。

位置为何重要

```
dog bites man
man bites dog
```

- token 集合相同。
- 顺序改变了含义。
- 模型必须知道每个 token 出现在哪里。

RoPE 的直觉



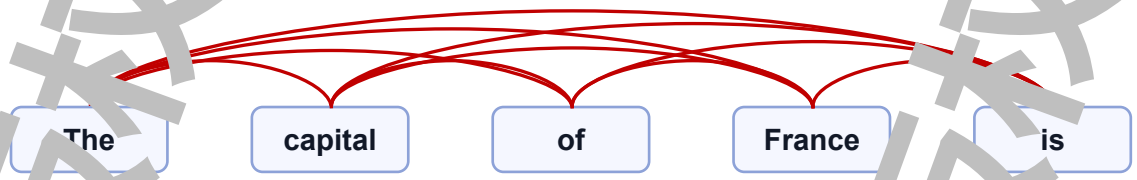
$$q_m \cdot k_n \sim f(\text{content}, m-n)$$

关键性质：Q/K 匹配携带相对位置信息。

长上下文为何昂贵

自注意力对 token 两两比较。

n tokens ⇒ $n \times n$ attention scores ⇒ $O(n^2)$



计算量

上下文越长，token 之间的比较次数越多

上下文长度 ×2
≈ 注意力分数 ×4

显存

长上下文带来巨大的 KV 缓存和沉重的显存带宽压力

上下文长度 ×2
≈ KV 缓存 ×2

这就是长上下文昂贵的原因。

高效注意力：从全注意力到稀疏注意力

长上下文迫使模型避免比较每一对 token。

全注意力

每个 token 都可以关注之前所有的 token。

```
x6 可见：
x1 x2 x3 x4 x5
```

信息充分，但长上下文代价高昂。

滑动窗口

每个 token 只关注一个局部窗口。

```
x6 可见：
x3 x4 x5
```

这是最简单的稀疏注意力模式。

混合注意力

主要使用廉价的注意力，但保留部分全局层。

- **GPT-oss：全注意力层与局部稀疏层交替。**
- **DeepSeek：MLA（低秩潜在注意力）；NSA/DSA 用于稀疏长上下文。**
- **MiniMax：混合 Lightning attention；MSA。**

MLA：先把每个 token 的 K、V 一起压缩成一个维度低得多的潜在向量（low-rank latent）

MLA解决的问题是推理时的 KV 缓存太大

要点：让大部分注意力更便宜，同时保留足够的全局信息流。

稠密模型扩展的瓶颈

当模型变宽时，另一个瓶颈出现了。

稠密层的挑战

larger model → more parameters + more FLOPs

- 更多参数通常意味着更强的能力。
- 90% 的参数来自稠密 FFN 层。
- 参数增长直接增加每个 token 的计算量。

硬件瓶颈

training cost ≈ FLOPs + memory / communication

- 超大规模下计算变得昂贵。
- 搬运激活值和参数同样耗时。
- 我们希望提升能力，但不必让每个 token 使用全部参数。

这催生了稀疏激活：参数很多，但每个 token 只使用其中一小部分。

稠密模型 vs 混合专家（MoE）

MoE 提升模型容量，而不必为每个 token 激活全部参数。

稠密层

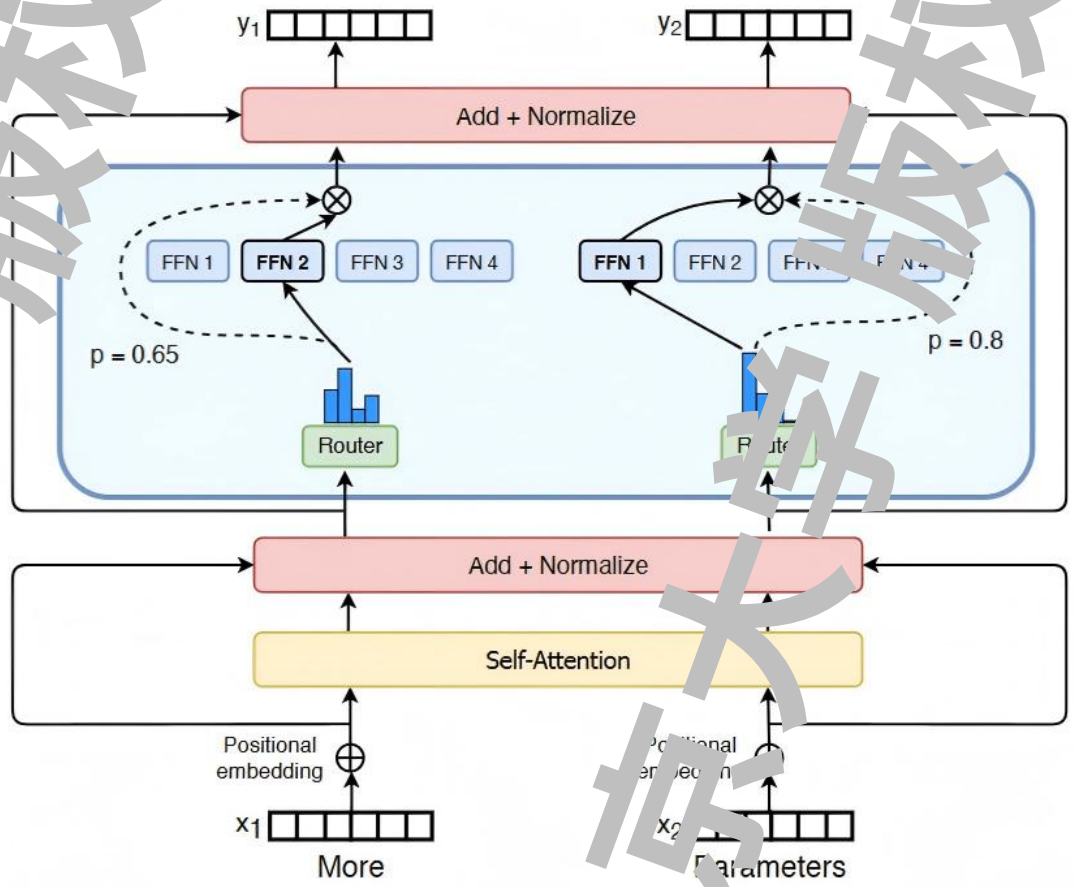
token -> 完整 MLP

MoE 层

token -> 路由器 -> top-k 个专家

稀疏 MoE 示例

DeepSeek V3 671B / 激活 37B	**DeepSeek V4-Pro** 1.6T / 激活 49B
Qwen3-235B-A22B 235B / 激活 22B	**Qwen3.6-35B-A3B** 35B / 激活 3B



架构演进：主要趋势

现代大模型大多是在同一个 decoder-only 骨架上不断打磨。

① Transformer 核心

Transformer

自注意力 | 并行序列建模

② Decoder-only 语言模型

GPT-3

下一个 token 预测 | 规模扩展

③ 稳定的模块

LLaMA

RMSNorm | SwiGLU | RoPE

解码器模块之后的架构趋势

A 高效上下文

GPT-OSS / DeepSeek-NSA

滑动窗口 | 混合注意力 | 线性 / 稀疏

B 稀疏容量

Mixtral / DeepSeek-V3 / Qwen3

路由器 | 专家 | 总参数量巨大

北京大学
PEKING UNIVERSITY

预训练：构建基座模型

先在海量文本上训练 Transformer，再将其适配到有用的任务上。

预训练：核心思路

通过预测下一个 token 来学习语言

从海量文本

```
The capital of France is → Paris
A function takes input and → returns output
In a proof, assume that → ...
To solve the equation, first → ...
```

每个都是从文本中学到的一个下一个 token 关联。

到语言模式

- 语法
- 文本中的事实
- 代码模式
- 推理用语
- ...

预训练是自监督的：文本本身就提供了下一个 token 的标签

预训练损失函数

在每个位置上，正确标签就是文本中的下一个 token。

一条序列产生多个损失

文本：
the mouse ate the cheese

the → mouse
the mouse → ate
the mouse ate → the
the mouse ate the → cheese

训练时这些位置是并行处理的。

交叉熵目标

$$l_t = -log\, p_\theta(x_{t+1} \mid x_{\le t})$$

$$\mathcal{L} = \frac{1}{L-1}\sum_{t=1}^{L-1} l_t$$

- 对真实的下一个 token 给出高概率 → 损失就低。
- 梯度下降更新模型参数 θ

预训练数据

预训练数据的关键是规模、覆盖面和过滤。

常见来源

- 网页
- 书籍、维基百科、新闻
- 代码与论文
- 数学与多语言数据

Token 规模

GPT-3: ~300B

Llama 2: ~2T

Llama 3: 15T+

Qwen2.5: ~18T

T = 万亿 token

过滤很重要

- 去除低质量文本
- 去重
- 平衡领域与语言
- 控制基准测试污染

规模提供覆盖面，过滤决定其中有多少真正有用。

基座模型学到了什么？

基座模型学会的是续写文本，而不是当助手。

有用的模式

- 语法与篇章结构
- 改写与类比
- 来自文本的世界知识
- 代码与数学模式

续写示例

```
he author of "A Room of One's Own" is ___
he square root of 4 is ___
It wasn't just big; it was ___
```

模型根据上下文续写最可能的文本。

预训练数据的局限

模型只能学到数据中存在的模式。

覆盖面的局限

- 冷门领域可能覆盖不足。
- 有些语言的高质量文本少得多。
- 数据截止日期之后的事件缺失。

质量的局限

- 网络文本包含噪声和重复内容。
- 事实可能过时或相互矛盾。
- 基准测试可能意外混入训练数据。

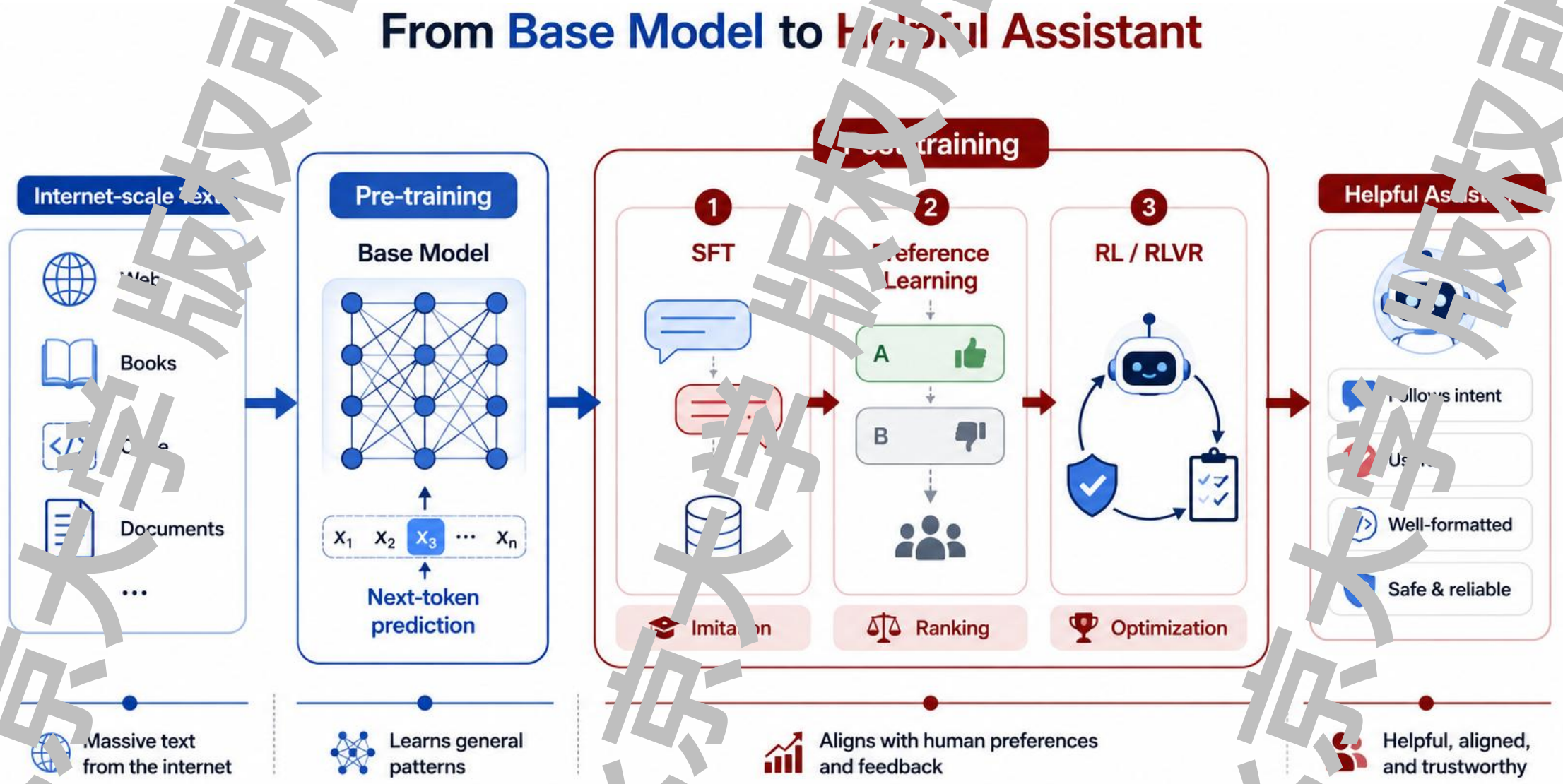
预训练造就强大的基座模型，但仅靠数据并不能定义助手的行为。

北京大学
PEKING UNIVERSITY

后训练：从基座模型到助手

SFT、RLHF、GRPO 与现代强化学习共同塑造有用的助手行为。

为什么需要后训练？


From Base Model to Helpful Assistant
Internet-scale
Books
Documents
Pre-training
Base Model
Next-token prediction
Post-training
1
SFT
2
Reference Learning
A
B
3
RL / RLVR
Imitation
Ranking
Optimization
Helpful Assistant
Follows intent
Well-formatted
Safe & reliable
Massive text from the internet
Learns general patterns
Aligns with human preferences and feedback
Helpful, aligned, and trustworthy

SFT：有监督微调

SFT 教会模型指令遵循的格式。

训练样例

指令：
Explain what a decoder-only Transformer is.

回复：
Sure! A decoder-only Transformer is ...

指令：
What directory are we currently in?

回复：
<tool_call>{"name": "pwd", "arguments": {}}
</tool_call>

SFT 教会了什么

- 回答问题，而不只是续写文本。
- 使用对话和回答的格式。
- 采用乐于助人的风格。
- 工具调用格式。

$$\mathcal{L}_{SFT} = -\sum_{t} \log p_\theta(y_t \mid x, y_{1:t-1})$$

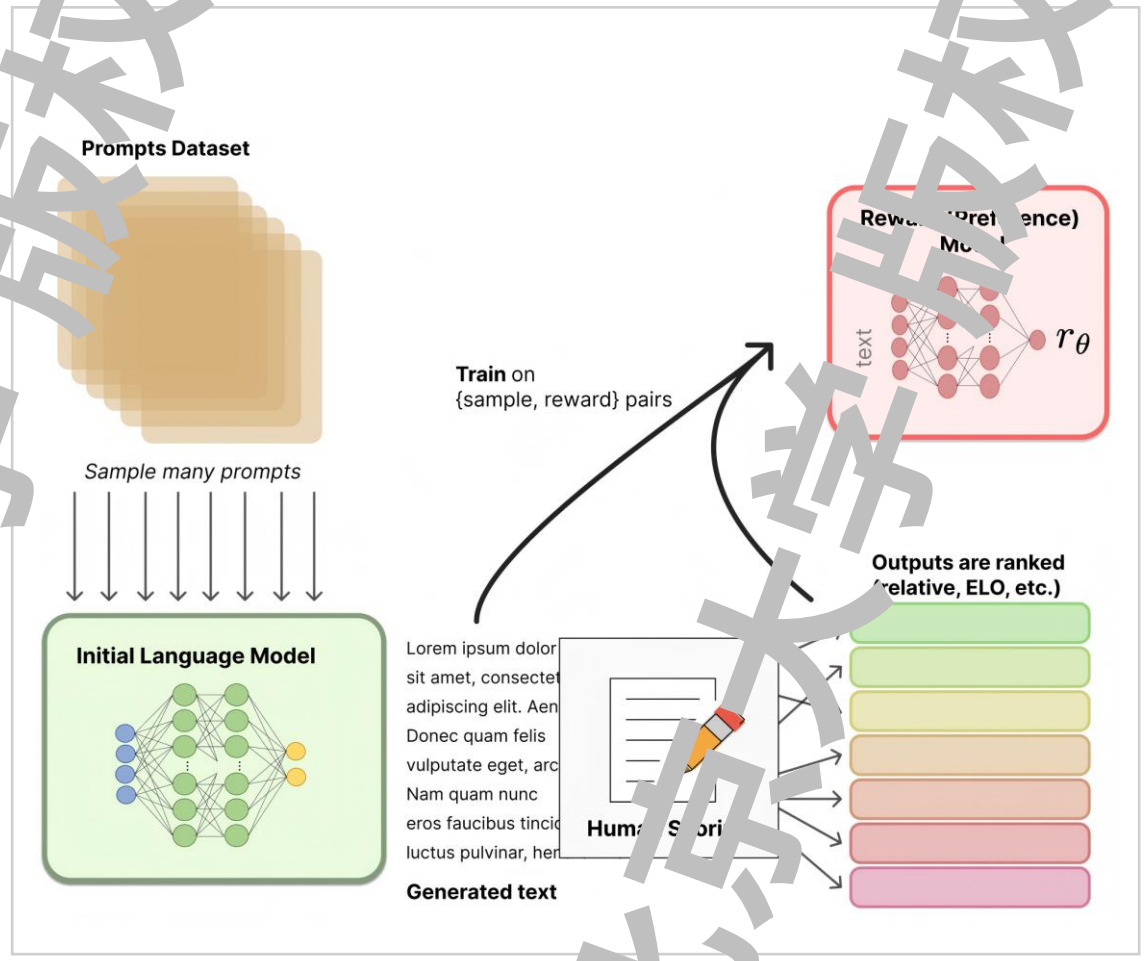
RLHF：学习人类偏好

RLHF 使模型与人类偏好对齐。

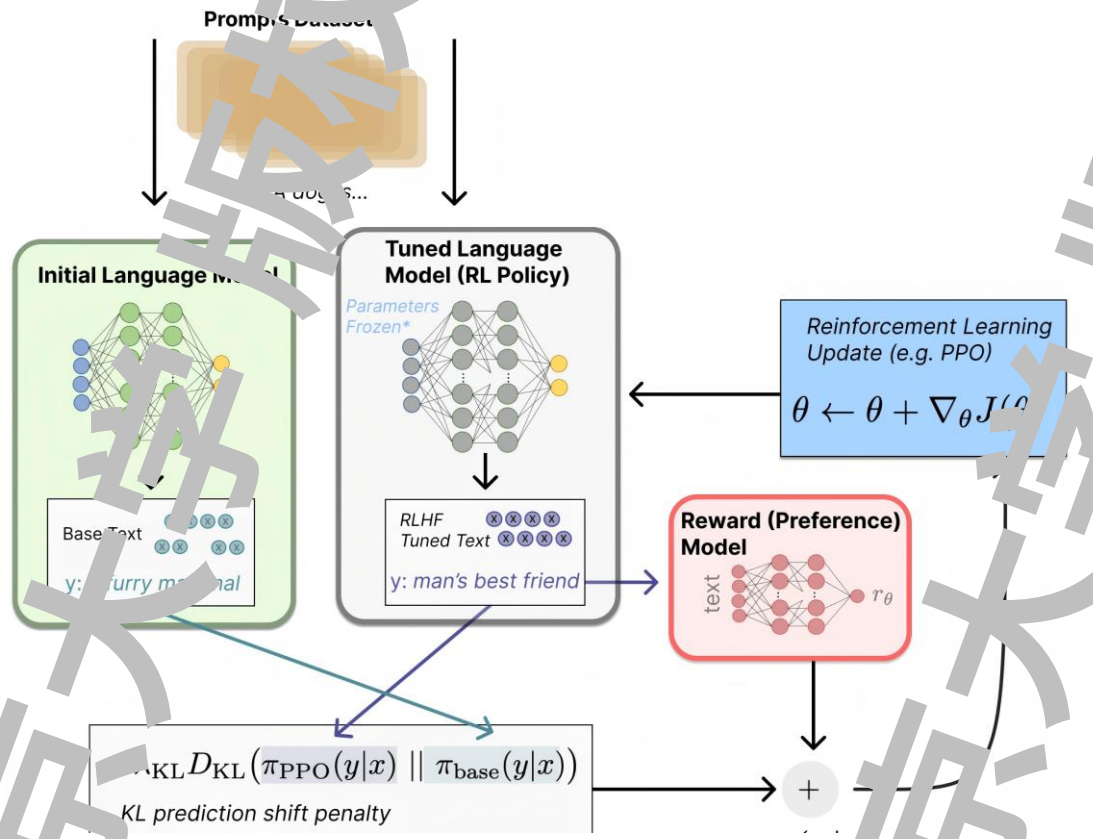
偏好数据

- 对同一提示采样多个回答。
- 由人类选出更好的回答。
- 用比较数据训练奖励模型。

奖励模型近似人类偏好。



RLHF：优化策略

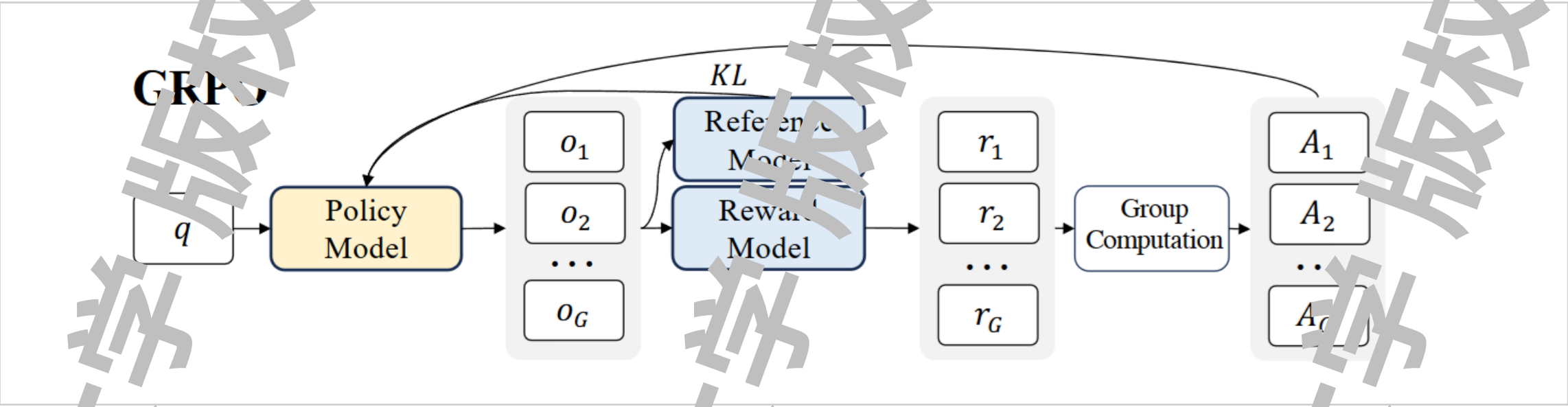


优化思路

- 策略模型生成回答。
- 奖励模型为其打分。
- 强化学习增强高奖励的行为。
- KL 惩罚让模型不偏离参考模型太远。

$$\max_{\theta} \mathbb{E}\left[r_{\phi}(x,y)\right] - \beta D_{KL}(\pi_{\theta} || \pi_{ref})$$

GRPO 通过强化高质量的解答来提升推理能力。



Group Relative Policy Optimization（组相对策略优化）

采样一组

对一个提示词生成多个回复。

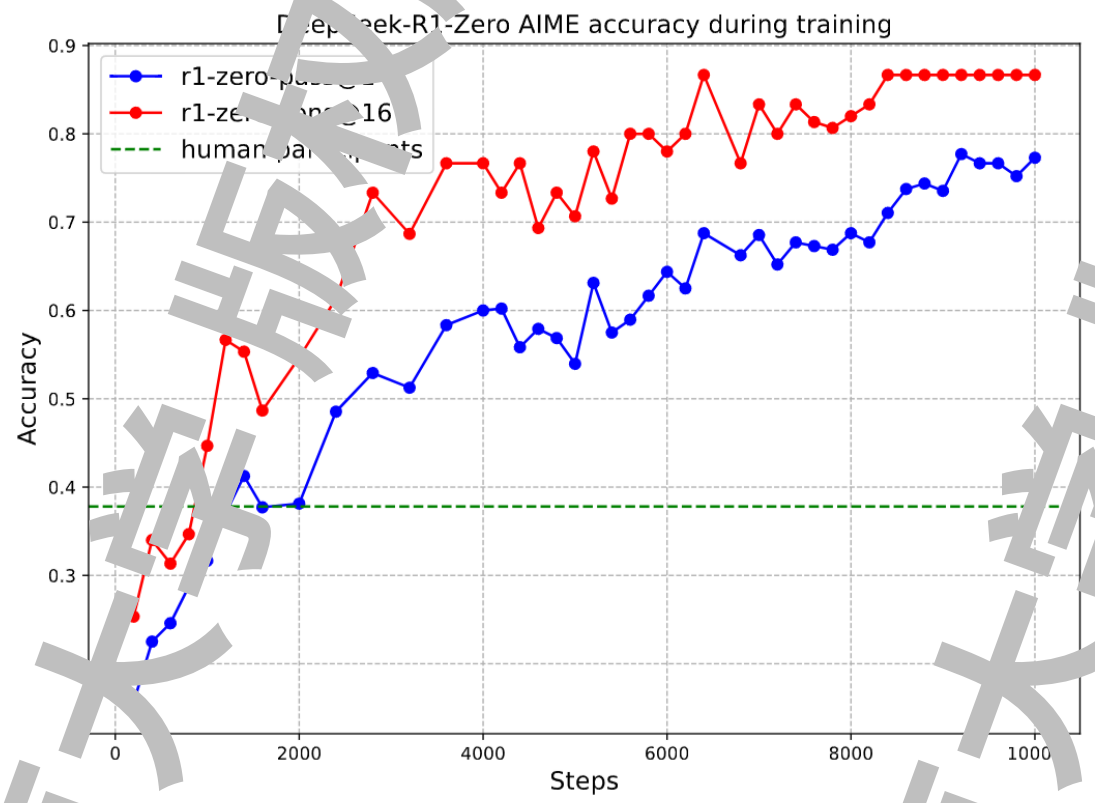
为回复打分

使用奖励模型、验证器或测试用例。

相对更新

提升组内更好回复的概率。

GRPO 训练动态



GRPO 的效果

- 激活基座模型中潜在的推理能力。
- 强化学习训练过程中性能稳步提升。
- 在许多困难基准上，模型超过了人类表现。

核心信息：强化学习可以把潜在能力转化为可见的推理表现。

训练流程总结

各训练阶段逐步把文本预测变成助手行为。

预训练 → SFT → RLH → GRPO / RLVR → 助

预训练

- 语言知识
- 世界规律
- 基础的代码 / 数学先验

SFT

- 指令遵循
- 对话格式
- 有帮助的回复行为

RLHF

- 使模型输出与人类偏好对齐
- 提升清晰度、有用性和回答质量

GRPO / RLVR

- 强化可验证的推理
- 数学 代码
- 长程问题求解

内容总结

从文本预测到现代大模型：完整的图景已经串起来了。

我们讲了什么

- 语言建模：为 token 序列赋予概率。
- 从文本到 token：单词变成 token、ID、嵌入和张量。
- Decoder-only Transformer：注意力混合上下文，MLP 逐 token 计算。
- 架构进步：高效注意力应对长上下文；MoE 提升参数容量。
- 训练流程：预训练构建基座模型；后训练塑造行为与推理。

理解了数据、架构和训练，你就可以开始读懂几乎任何大模型技术报告。

课程总结


后续如何发展?
第三次浪潮
第二次浪潮
(1987)
第一次浪潮
(1956-1974)
第二次低谷
(1987-1993)
诞生
(1956，达特茅斯会议)

北京大学
PEKING UNIVERSITY

从大模型原理到Verilog生成

如何基于大模型的原理去更好的生成verilog代码

任务定义：从规格到 RTL

把 Verilog 生成形式化为条件序列生成问题，并以功能等价为最终判据。

自然语言规格 → 提示 / 上下文 → Transformer 解码 → Verilog 代码 → 仿真与综合验证

形式化描述

给定规格x，模型输出token序列$y = (y_1, y_2, \ldots, y_n)$：

$$p(y \mid x) = \prod_t p(y_t \mid y_{(<t)}, x)$$

训练目标是最大化正确 RTL 的条件似然；但真正关心的是生成代码与规格是否功能等价。

与通用代码生成的差异

- Verilog 描述的是硬件结构，而非顺序执行的语句
- 阻塞/非阻塞赋值、时钟与复位语义容易被误用
- 正确性由仿真波形与形式验证判定，而不是"跑通"
- 还需同时考虑面积、时序、功耗（PPA）

评价指标：语法通过率 → 功能正确率（pass@k）→ 综合后 PPA，三者层层递进。

为什么用 Transformer 建模 Verilog

长程依赖、结构化语法与大规模自监督训练，恰好契合 RTL 代码的特点。

架构层面的契合

- 自注意力机制跨越数百行连接端口声明与使用处
- 因果解码天然匹配"逐 token 续写代码"
- 同一骨架可同时处理注释、规格与代码
- 规模扩展带来的收益在代码任务上尤为明显

需要额外处理的问题

- 公开 Verilog 语料远少于 Python / C++
- 分词器对位宽、下划线命名切分不友好
- 模块层级结构难以用纯线性序列表达
- 语法正确不等于综合可用，更不等于功能正确

长程依赖
端口 ↔ 时序逻辑

结构语法
module / always / assign

可验证
仿真给出明确信号

数据构建与训练流程

先用大规模代码语料获得基础能力，再用高质量样本对齐 RTL 编写范式。

通用预训练
自然语言 + 多语言代码

→

领域继续预训练
Verilog / SyVerilog 语料

→

监督微调
规格–代码配对样本

→

强化学习
以仿真反馈为驱动

数据来源与清洗

- 开源硬件仓库、教学习题、IP 核代码
- 按模块切分，去重并剔除不可综合片段
- 用 EDA 工具过滤无法通过语法检查的样本

监督样本的构造

- 从已有代码反向生成自然语言规格
- **补充 testbench，使样本自带正确性判据**
- **按难度分层：组合逻辑 → 时序 →**

数据质量比数据规模更关键：带 testbench 的样本才能支撑后续的强化学习。

监督微调为何不够

交叉熵只在拟合参考代码，而硬件正确性是离散、可验证、且不唯一的。

监督微调的目标

逐 token 拟合参考代码：

$$L = -\Sigma_t \log p(y_t \mid y_{<t}, x)$$

写法不同但功能等价的实现会被惩罚；而只差一个位宽的错误代码损失却很小。

```
always @(posedge clk)
    q = d;        // 应为 <=，形式相近、后果严重
```

典型失败模式

- 语法完全正确，仿真波形却不匹配
- 把非阻塞赋值写成阻塞赋值，时序错乱
- 状态机漏掉 default 分支，产生锁存器
- 端口位宽与规格不一致，综合报错

需要的是一种能直接对"功能是否正确"打分的训练信号，

而这正是强化学习所擅长的。

可验证奖励的设计

硬件设计流程本身就提供了自动、客观的打分器。

语法与综合性

- 编译器 / lint 检查是否通过
- 是否能被综合工具接受
- 作为门槛项，失败直接给负奖励

功能正确性

- 用 testbench 跑仿真比对波形
- 按通过的测试用例比例给分
- 是主奖励项，权重最大

PPA 质量

- 综合后的面积、时序裕量、功耗
- 在功能正确的前提下进行优化
- 引导模型写出"好"的实现

总奖励可写成分层加权的形式：

$$R = w_1 \cdot R\_语法 + w_2 \cdot R\_功能 + w_3 \cdot R\_PPA，其中\ R\_功能 = 通过测试数 / 测试总数$$

奖励来自工具而非人工标注，因此可以低成本地大规模采样——这是 RTL 任务相对友好的地方。

强化学习训练循环

以 GRPO 为例：对同一[illegible]采样一组候选，用组内相对优势更新策略。

① 采样
对一条[illegible]生成 G 份代码

② 打分
编译 + 仿真 + 综合给出奖励

③ 计算优势
组内归一化，得到相对优势

④ 更新策略
提升高奖励样本的概率

组内相对优势

$$A_i = (R_i - \text{mean}(R)) / \text{std}(R)$$

无需单独训练价值网络，直接用一组样本的奖励分[illegible]做基线。

工程上的关键点

- 仿真耗时是瓶颈，需要并行与结果缓[illegible]
- KL 惩罚防止模型偏离可读的代码风格
- 课程式采样：先易后难，避免奖励[illegible]

奖励稀疏时，可先用语法奖励"热身"，再逐步提高功能奖励的权重。

实验设置与评测基准

以功能正确率为主指标，并观察综合质量与失败类型的变化。

常用基础评测指标

- Verilog Eval：自然语言描述 → 模块实现
- RTL-MT：面向真实设计任务的生成评测
- 主指标 pass@k：k 次采样中至少一次功能正确
- 辅助指标：语法通过率、综合面积与时序

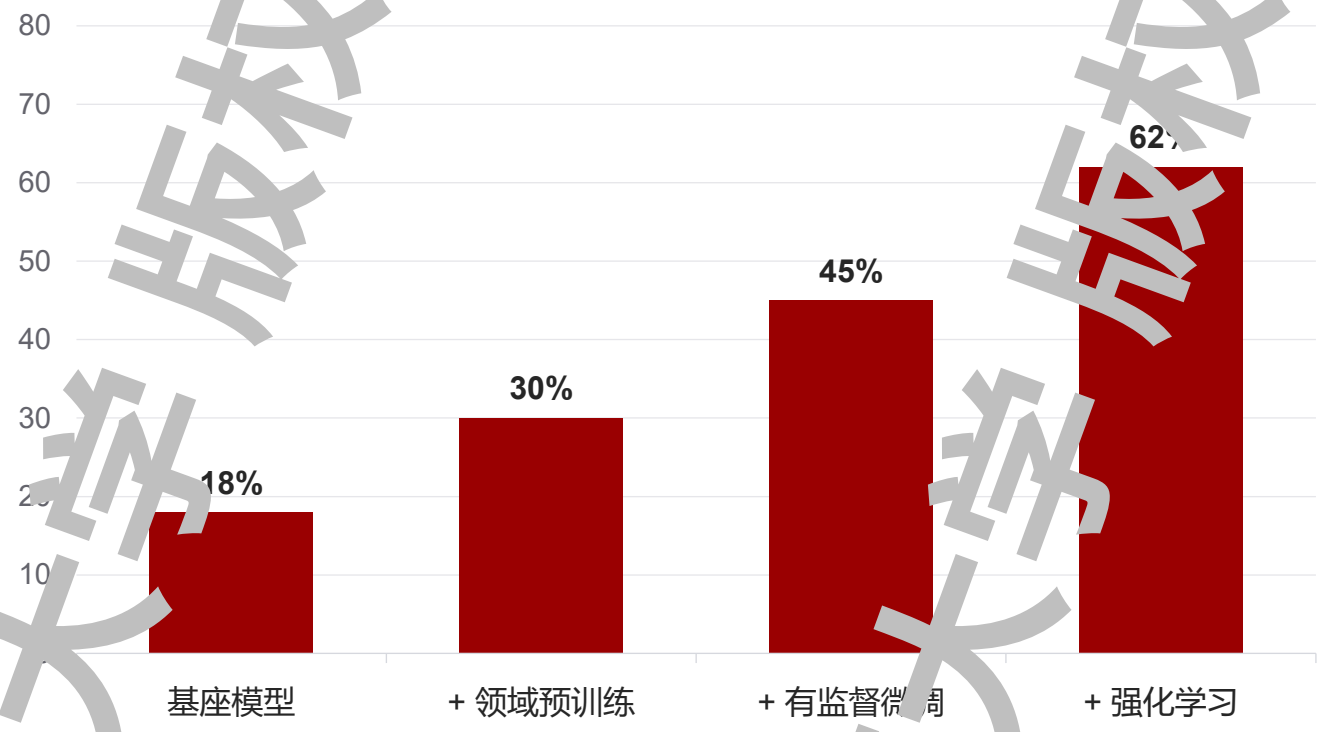
对比设置

- 基座模型：仅通用代码预训练
- + 领域继续预训练：加入 Verilog 语料
- + 有监督微调：规格–代码配对样本
- + 强化学习：加入仿真反馈的可验证奖励

评测需固定采样温度与提示模板，并保证 testbench 与训练数据严格隔离，避免污染。

结果：功能正确率的逐级提升

各训练阶段对功能正确率的相对贡献（示意，用于说明趋势而非实测数值）



可以观察到的现象

- 领域预训练主要修复语法与风格问题
- 有监督微调让模型学会遵循风格格式
- 强化学习带来的增量集中在功能正确性
- 错误类型从"编译不过"转向"边界情况"
- 综合面积在 PPA 奖励下同步改善

趋势结论：可验证奖励作用于监督微调难以覆盖的功能层面，两者互补而非替代。

挑战与展望

从"能写模块"到"能参与设计"，仍有若干关键问题待解决。

当前挑战

- 高质量带 testbench 的数据稀缺
- 仿真与综合开销限制 RL 采样规模
- 大型层级设计远超上下文能力

可能的方向

- 形式验证提供更强的正确性奖励
- 与 EDA 工具链深度闭环迭代
- 分层生成：先架构后模块

落地形态

- 模块级代码助手与补全
- 自动生成 testbench 与断言
- 设计空间探索中的候选生成器

小结：Transformer 提供生成能力，强化学习提供正确性约束，硬件验证流程提供可信修改——三者结合是 Verilog 自动生成的基本框架。

短期看是提效工具，长期看可能重塑 RTL 设计的分工方式。