工程设计课程讲义系列

课程代码 46050107 · 深度学习与图像识别 · 2 学分 / 32 学时

# 人工智能与图像识别

## ——产教融合项目式课程讲义——

第一篇　理论篇：十二讲（第 6–18 周）

第二篇　项目背景篇：三个真实项目到底在做什么

第三篇　任务书篇：15 个小组的任务、实验与验收标准

主讲：张洪进　工程学院

授课班级：2024 机器人 1 班、2024 机器人 2 班

2026–2027 学年第一学期　2026 年 10 月

# 人工智能与图像识别——产教融合项目式课程讲义

主编：张洪进

2026 年 10 月第一版

本讲义配套课程《深度学习与图像识别》（课程代码 46050107，工程学院，专业选修，2 学分 32 学时：讲授 24 学时＋实验 8 学时）。教材：许庆阳《深度学习模型与算法基础》（清华大学出版社，2023 年 12 月第 1 版）；参考书：Ian Goodfellow、Yoshua Bengio、Aaron Courville《深度学习》（人民邮电出版社，2017 年）。本讲义为教学内部资料，其中引用的网络图片与论文插图仅用于课堂教学，版权归属原作者，来源均已逐图标注。

# 前言

## 这本讲义和你以前见过的教材不一样

如果你翻开这本讲义时想的是“又是一本背概念应付考试的教材”，那你很快会发现不对劲——这本讲义里没有“期末划重点”，取而代之的是三份货真价实的工程任务书：今年秋天，你们中的绝大多数人将加入三个真实项目组，和研究生、毕业设计学长一起，把人工智能技术用到海洋石油平台的全景监控、水下锚链检查和海量工程数据治理上。

这就是我们这门课的教学改革方案，我们叫它**“产校验方案”**——真实生产项目直接进入学校课堂，用真实项目检验课堂所学：*12* 讲理论课打底，*4* 次现场实验搭桥，*15* 个小组认领真实课题，第 *19* 周周五（2027 年 1 月 22 日）全部成果提交验收。你在这门课里写下的每一行代码、跑出的每一个数字，都不是“作业题”，而是要被真实系统采用的工程结果。

## 怎么使用这本讲义

全书分三篇，对应你在课程里的三个阶段：

- **第一篇 · 理论篇（第 6–18 周随堂讲授）**：十二讲，从最古老的感知器一路讲到今天的大模型。完全从零基础写起：需要的数学，现场补（蓝色“知识补齐”盒子）；看不懂的地方，给生活化的类比（青色“零基础扫盲”盒子）；每一讲都告诉你它和三个项目的关系（橙色“项目连线”盒子）。每讲最后一道课后编程题，要求你**指挥大模型（AI）替你编程**——这是本课程最核心的技能训练：你不需要从零手写代码，但你必须能看懂、能改动、能讲清楚 AI 给你写了什么。
- **第二篇 · 项目背景篇（第 5–8 周自行阅读）**：三个项目各一章，讲清楚它们“是什么、为什么做、为谁做”。读完这一篇，你拿到任务书时不会迷茫。
- **第三篇 · 任务书篇（贯穿全学期）**：15 个小组每组一份任务书——总体任务、第 19 周周五的提交清单、哪些活 AI 干、哪些活必须你自己干、防糊弄指南、满分验收标准，以及四个实验各自的具体要求。这一篇是你这学期的“作战地图”。

## 关于 AI 的承诺与底线

这门课不禁止 AI，恰恰相反——**这门课要求你用 AI**。四次现场实验的第一课，就是配置你自己的大模型编程助手。但有一条底线贯穿全书：**AI 可以用，结果你自己扛**。实验数据不许直接抄 AI 给的数字；训练精度不许虚构；每一个交付物都要有“命令＋日志＋产物”三件套证据。工程世界里，说谎的代价不是扣分，是事故。

## 致谢

本讲义参考了国内外大量经典教材、论文与开源资料，包括许庆阳《深度学习模型与算法基础》、Goodfellow 等《深度学习》、*Stanford CS231n/CS224n* 课程讲义、以及 *Fast-FoundationStereo*、*Depth*

*Anything V2*、*R-GCN* 等论文与开源项目；三个项目的背景材料来自真实工程资料。所有引用在书中逐处标注，并在附录集中列出。感谢参与三个项目的全体研究生助教与工程师。

张洪进

2026 年 10 月

# 目录

前言 i

第一部分　第一篇　理论篇：十二讲，从感知器到大模型 1

第 0 讲　开课导引：这门课怎么上、怎么学、怎么考 2
0.1 这门课讲什么：从一个神经元到大模型 . . . 2
0.1.1 先排座次：人工智能、机器学习、神经网络与深度学习 . . . 4
0.2 三个真实项目一页速览 . . . 4
0.3 你需要准备什么 . . . 6
0.4 如何用大模型帮你学编程（本书最重要的方法论） . . . 7
0.4.1 AI 编程助手是什么 . . . 7
0.4.2 提示词怎么写：四个要素 . . . 8
0.4.3 agent 循环：AI 是怎么“自己干活”的 . . . 8
0.5 零基础数学包：够用就好的三小块 . . . 9
0.6 零基础 Python 包：从装环境到会读代码 . . . 11
0.7 全书地图：三篇怎么配合用 . . . 12

第 1 讲　感知器、线性单元与学习规则 14
1.1 生物神经元：大脑的原始零件 . . . 14
1.2 感知器：一颗人工神经元 . . . 15
1.3 决策边界：感知器在平面上画一条线 . . . 16
1.4 它怎么学习：感知器学习规则 . . . 18
1.5 从 0/1 到连续：线性单元与最小均方 . . . 20
1.6 天花板与寒冬：一本书按下的暂停键 . . . 21
1.7 从这一讲到 2026 . . . 23

第 2 讲　BP 神经网络、反向传播与梯度检查 25
2.1 多层感知器：自动造特征的流水线 . . . 25
2.2 激活函数：没有它，叠一百层也白叠 . . . 26
2.3 梯度下降：在山地里找最低谷 . . . 28
2.4 反向传播：把责任一层层倒推回去 . . . 29
2.5 梯度检查：给你的反向传播“对账” . . . 30
2.5.1 三种求导范式：从手推公式到 autograd . . . 31
2.6 完整训练一次：把循环跑起来 . . . 32
2.7 从这一讲到 2026 . . . 34

**第 3 讲　超参数、权重初始化、模型训练与评估** **37**
3.1 损失面与梯度下降：先看清脚下是什么地形 . . . 37
3.2 学习率：整张清单上最重要的一个数 . . . 38
3.3 权重初始化：第一步就走对 . . . 40
3.4 batch、epoch、iteration：分清训练的三兄弟 . . . 41
3.5 训练集、验证集、测试集：背题与会做题 . . . 42
3.6 向量化编程：为什么一行 numpy 顶一百行 for . . . 43
3.7 优化器家族一分钟：从 SGD 到 Adam . . . 44
3.8 从这一讲到 2026 . . . 45

**第 4 讲　自编码器** **48**
4.1 无监督学习登场：没有标签的世界 . . . 48
4.2 自编码器结构：自己压缩自己 . . . 49
4.3 隐空间：压缩后的向量是“本质坐标” . . . 50
4.4 去噪自编码器：先弄脏，再学会擦 . . . 51
4.5 变体一分钟与“AE 的当代后代” . . . 52
4.6 从这一讲到 2026 . . . 54

**第 5 讲　卷积神经网络原理与前向计算** **57**
5.1 全连接看图：一场参数灾难 . . . 57
5.2 卷积 = 滑动窗口做点积 . . . 58
5.3 padding 与 stride：滑窗的两个旋钮 . . . 59
5.4 池化：只留最强的响应 . . . 60
5.5 特征图与层级：从边缘到猫脸 . . . 60
5.6 完整前向流水线：从像素到类别得分 . . . 63
5.7 从这一讲到 2026 . . . 65

**第 6 讲　卷积神经网络的训练与计算模式** **67**
6.1 练手房：MNIST 与 CIFAR-10 . . . 67
6.2 LeNet-5 精读：第一栋“经典公寓”的户型图 . . . 68
6.3 数据增强：白捡的“新数据” . . . 70
6.4 训练循环：五个动作的摩天轮 . . . 72
6.5 批归一化 BN：给每一层的数据“扶正” . . . 73
6.6 评估：准确率会骗人 . . . 74
6.7 过拟合对策清单 . . . 75
6.8 从这一讲到 2026 . . . 76

**第 7 讲　卷积神经网络的演化：加宽、加深与残差** **78**
7.1 ImageNet：军备竞赛的“高考考场” . . . 78
7.2 AlexNet：一鸣惊人的三件法宝 . . . 79
7.3 VGG：极简主义——只堆 3×3 . . . 80
7.4 GoogLeNet：多尺度并行与 1×1 省钱术 . . . 82
7.5 深化的障碍：56 层为什么输给 20 层 . . . 83

7.6 残差革命：给梯度修一条高速公路 . . . 83
7.7 设计经验法则：深度、宽度、分辨率的三角 . . . 84
7.8 从这一讲到 2026 . . . 86

**第 8 讲 轻量化网络、全卷积网络与卷积操作模式设计 88**
8.1 模型压缩四板斧 . . . 88
8.2 深度可分离卷积：轻量化的核心积木 . . . 89
8.3 轻量家族一分钟点名 . . . 91
8.4 全卷积网络 FCN：从“是什么”到“在哪儿” . . . 91
8.5 卷积操作模式工具箱 . . . 94
8.6 端侧部署三件套：ONNX、推理引擎与 INT8 . . . 95
8.7 从这一讲到 2026 . . . 97

**第 9 讲 循环神经网络 RNN 99**
9.1 序列数据无处不在：世界是按帧来的 . . . 99
9.2 RNN 的核心思想：一个网络用到底，一张便签传下去 . . . 100
9.3 用 RNN 做语言模型：文字接龙 . . . 102
9.4 BPTT：把展开图当普通网络训练 . . . 104
9.5 梯度消失与爆炸：记忆的大敌 . . . 104
9.6 RNN 的应用与局限，以及 2026 年为什么还学它 . . . 105
9.7 从这一讲到 2026 . . . 107

**第 10 讲 长短时记忆网络 LSTM 与 GRU 109**
10.1 细胞状态：一条直通远方的传送带 . . . 109
10.2 三扇门：该忘的忘，该记的记，该说的说 . . . 111
10.3 LSTM 逐步计算：六行公式，六句人话 . . . 112
10.4 GRU：两扇门的精简版 . . . 113
10.5 实战对比：一张图看懂差距 . . . 115
10.6 2026 年的选型决策：什么时候还用 RNN 家族 . . . 116
10.7 从这一讲到 2026 . . . 117

**第 11 讲 注意力模型 119**
11.1 为什么要注意力：从 Seq2Seq 的“失忆症”说起 . . . 119
11.2 Q/K/V 三兄弟：一次“查字典”式的信息检索 . . . 120
11.3 自注意力：一句话内部的“圆桌会议” . . . 123
11.4 多头注意力与位置编码：一支足球队而非一个球星 . . . 124
11.4.1 多头注意力：八支荧光笔各画各的重点 . . . 124
11.4.2 位置编码：给天生的“脸盲”贴工位号 . . . 124
11.5 Transformer 全家福：用注意力搭一整栋楼 . . . 125
11.6 规模定律：从 GPT-1 到 2026 年的大模型 . . . 126
11.7 Agent：注意力 + 工具 = 会干活的模型 . . . 128
11.8 从这一讲到 2026 . . . 129

**第 12 讲　生成对抗模型** **132**
12.1 生成模型与判别模型：无中生有 vs 看图说话 . . . 132
12.2 GAN：造假者与鉴定师的对决 . . . 133
12.3 训练：一场没有裁判的拉锯战 . . . 134
12.4 GAN 演进长廊：从模糊噪点到以假乱真 . . . 135
12.5 另一条路：变分自编码器（VAE） . . . 136
12.6 后浪：扩散模型 . . . 137
12.7 深度伪造与鉴伪：攻防共生（工程伦理专场） . . . 138
12.8 从这一讲到 2026 . . . 140

**第二部分　第二篇　项目背景篇：三个真实项目到底在做什么** **143**

**第二篇导读：三个项目全景** **144**
0.9 三个项目：先来一张速览表 . . . 144
0.10 共同的课程框架：节点、证据与底线 . . . 145
0.11 15 个小组地图：找到你自己的位置 . . . 145
0.12 怎么读这一篇 . . . 146

**项目一　六目拼接：给六只眼睛一个全景世界** **147**
1.1 需求从哪来：一台相机看不住一座平台 . . . 147
1.2 全景拼接五步曲：从六张歪图到一幅长卷 . . . 151
1.2.1 第一步：畸变校正——先把“哈哈镜”掰直 . . . 151
1.2.2 第二步：特征提取与匹配——在两幅画面里找到“同一个点” . . . 151
1.2.3 第三步：几何对齐——一张 3×3 表格把画面“扶正” . . . 152
1.2.4 第四步：缝合线——让“刀口”藏在最不显眼的地方 . . . 153
1.2.5 第五步：融合——把接缝“羽化”到看不见 . . . 153
1.3 360Stitcher 长什么样：六个模块与一串硬数字 . . . 153
1.4 为什么要“可学习”的参数：九个参数头 . . . 156
1.5 双路线对照：逐个学头，还是整网端到端 . . . 158
1.6 八个组怎么分工：一条流水线上的八个工位 . . . 159
1.7 你将用到第一篇的哪些讲 . . . 160

**项目二　水下双目视觉自动标记：让 AI 看懂海底世界** **163**
2.1 场景与需求：甲方要“AI 看一眼就知道有多远” . . . 165
2.2 水下为什么难：光、匹配、泛化、真值四道坎 . . . 166
2.2.1 第一道坎：光被水“吃掉” . . . 167
2.2.2 第二道坎：立体匹配的前提被破坏 . . . 168
2.2.3 第三道坎：深度网络泛化差 . . . 168
2.2.4 第四道坎：没有深度真值 . . . 168
2.3 双目立体视觉：两只眼睛如何“看出”距离 . . . 169
2.4 没有深度真值怎么办：自动打标签（本章高潮） . . . 171
2.4.1 标签本体：一张逐像素视差图 . . . 171
2.4.2 三步流水线：教师打标、多重过滤、学生迭代 . . . 171

2.4.3 最大的先例：NVIDIA 用 140 万对伪标签蒸出快 10 倍的学生 . . . 172
2.4.4 五条路线与前沿模型 . . . 174
2.5 我们的 500 对数据集：一次真实的翻车与自救 . . . 176
2.5.1 翻车：八个“典型场景”全军覆没 . . . 176
2.5.2 自救：两条硬指标把数据“挖”出来 . . . 176
2.5.3 没有标定板的自标定 . . . 176
2.5.4 一个价值 7 倍的工程决策 . . . 176
2.5.5 验收：数字说话 . . . 177
2.6 四个组怎么分工：四条并行赛道，实验四会师 . . . 179
2.7 回头看：你将用到第一篇的哪些本事 . . . 180

**项目三 数据基座治理：让海量工程数据开口说话 183**
3.1 数据的洪水与资料的迷宫 . . . 184
3.2 让机器“按结构读懂”：数据结构化 . . . 185
3.3 一句话问数：智能体与可追溯查询 . . . 188
3.4 知识门控：AI 抽的知识入库前先过检查 . . . 190
3.5 图谱查关系：知识图谱与图检索 . . . 192
3.6 假内容攻防战（本章高潮） . . . 193
3.7 0.88 的谎言与 0.74 的诚实 . . . 196
3.8 三层身份与验收文化 . . . 198
3.9 回头看：你将用到第一篇的哪些本事 . . . 199

**第三部分 第三篇 任务书篇：15 个小组的任务、实验与验收 202**

**第三篇使用说明：任务书怎么读、怎么交、怎么评 203**
0.10 任务书怎么读：固定七节，先通则后本组 . . . 203
0.11 证据怎么留：三件套、交付契约与求助通道 . . . 203
0.12 AI 怎么用：分工通则与两条红线 . . . 204
0.13 成绩怎么算：建议方案（教师可调） . . . 205
0.14 第 19 周周五交什么：总清单模板 . . . 205
0.15 15 个小组速查表 . . . 205

**第 1 组 六目拼接 · 匹配决策组（九头之一：匹配与几何） 207**
1.1 组档案 . . . 207
1.2 项目背景与你的位置 . . . 207
1.3 核心任务 . . . 208
1.4 第 19 周周五提交清单 . . . 209
1.5 AI 分工与人工必做 . . . 209
1.6 防糊弄指南与满分标准 . . . 209
1.7 四次实验安排 . . . 210
1.7.1 实验一（10 月 12 日）：让大模型替你读代码，你来当裁判 . . . 210
1.7.2 实验二（11 月 2 日）：本机部署，亲眼看见九个头上线 . . . 211
1.7.3 实验三（11 月 23 日）：mc 头真训练、真交付、真对比 . . . 211

1.7.4 实验四（12 月 21 日）：补齐两头，并用新标签复训 mc . . . . . 211

**第 2 组 六目拼接 · 特征质量组（九头之二：特征与时序） 213**

2.1 组档案 . . . . . 213
2.2 项目背景与你的位置 . . . . . 213
2.3 核心任务 . . . . . 214
2.4 第 19 周周五提交清单 . . . . . 215
2.5 AI 分工与人工必做 . . . . . 215
2.6 防糊弄指南与满分标准 . . . . . 216
2.7 四次实验安排 . . . . . 216
2.7.1 实验一（10 月 12 日）：读懂规格书，写出规格书 . . . . . 217
2.7.2 实验二（11 月 2 日）：本机部署，外加一次“占位模型”对照 . . . . . 217
2.7.3 实验三（11 月 23 日）：temporal_state 真训练交付，图像头交设计 . . . . . 217
2.7.4 实验四（12 月 21 日）：真机 patch 联调，图像头迁移转真 . . . . . 218

**第 3 组 六目拼接 · 缝合融合组（九头之三：缝合与融合） 219**

3.1 组档案 . . . . . 219
3.2 项目背景与你的位置 . . . . . 219
3.3 核心任务 . . . . . 220
3.4 第 19 周周五提交清单 . . . . . 221
3.5 AI 分工与人工必做 . . . . . 221
3.6 防糊弄指南与满分标准 . . . . . 221
3.7 四次实验安排 . . . . . 222
3.7.1 实验一（10 月 12 日）：让大模型替你读 C++，你来当裁判 . . . . . 222
3.7.2 实验二（11 月 2 日）：本机部署，顺手记下基线数字 . . . . . 223
3.7.3 实验三（11 月 23 日）：gain_field 真训练交付，seam 头交设计 . . . . . 223
3.7.4 实验四（12 月 21 日）：八头齐，fusion 正式版压轴上场（合并联调） . . . . . 223

**第 4 组 六目拼接 · 数据与标签组 224**

4.1 组档案 . . . . . 224
4.2 项目背景与你的位置 . . . . . 224
4.3 核心任务 . . . . . 225
4.4 第 19 周周五提交清单 . . . . . 226
4.5 AI 分工与人工必做 . . . . . 226
4.6 防糊弄指南与满分标准 . . . . . 226
4.7 四次实验安排 . . . . . 227
4.7.1 实验一（10 月 12 日）：读懂标签生产线，画出数据流图 . . . . . 227
4.7.2 实验二（11 月 2 日）：先在基准图上跑通全流程 . . . . . 228
4.7.3 实验三（11 月 23 日）：真机场地采集，交出第一条验收线 . . . . . 228
4.7.4 实验四（12 月 21 日）：扩量与质检，交出最终数据家底 . . . . . 228

**第 5 组　六目拼接 · 测试集与评测组** **229**
5.1 组档案 . . . 229
5.2 项目背景与你的位置 . . . 229
5.3 核心任务 . . . 230
5.4 第 19 周周五提交清单 . . . 231
5.5 AI 分工与人工必做 . . . 231
5.6 防糊弄指南与满分标准 . . . 232
5.7 四次实验安排 . . . 233
5.7.1 实验一（10 月 12 日）：让 agent 替你读评测工具，你来写说明书 . . . 233
5.7.2 实验二（11 月 2 日）：两条路线都部署，两种探针都点亮 . . . 233
5.7.3 实验三（11 月 23 日）：冻结测试集 v1，基线入账，回归全绿 . . . 234
5.7.4 实验四（12 月 21 日）：台账 v2 与困难样本，裁判的终场哨 . . . 234

**第 6 组　六目拼接 · 端到端数据组** **235**
6.1 组档案 . . . 235
6.2 项目背景与你的位置 . . . 235
6.3 核心任务 . . . 236
6.4 第 19 周周五提交清单 . . . 237
6.5 AI 分工与人工必做 . . . 237
6.6 防糊弄指南与满分标准 . . . 238
6.7 四次实验安排 . . . 239
6.7.1 实验一（10 月 12 日）：读懂端到端的数据餐单，写两路线差异说明 . . . 239
6.7.2 实验二（11 月 2 日）：合成数据全流程，32+8 对先干跑一遍 . . . 240
6.7.3 实验三（11 月 23 日）：首批公开数据集（UDIS-D），50 对起步 . . . 240
6.7.4 实验四（12 月 21 日）：数据扩量与 benchmark 初版，双线收官 . . . 240

**第 7 组　六目拼接 · 端到端训练组** **241**
7.1 组档案 . . . 241
7.2 项目背景与你的位置 . . . 241
7.3 核心任务 . . . 242
7.4 第 19 周周五提交清单 . . . 243
7.5 AI 分工与人工必做 . . . 243
7.6 防糊弄指南与满分标准 . . . 244
7.7 四次实验安排 . . . 244
7.7.1 实验一（10 月 12 日）：读懂你将要训练的东西，画出数据流 . . . 245
7.7.2 实验二（11 月 2 日）：本机 CPU 冒烟，三十分钟走通全链路 . . . 245
7.7.3 实验三（11 月 23 日）：4080 真训练，全书唯一的 200 epoch . . . 245
7.7.4 实验四（12 月 21 日）：全量真实数据微调，v2 收官 . . . 246

**第 8 组　六目拼接 · 端到端优化组** **247**
8.1 组档案 . . . 247
8.2 项目背景与你的位置 . . . 247
8.3 核心任务 . . . 248

8.4 第 19 周周五提交清单 . . . 249
8.5 AI 分工与人工必做 . . . 249
8.6 防糊弄指南与满分标准 . . . 250
8.7 四次实验安排 . . . 250
8.7.1 实验一（10 月 12 日）：拆解一帧的时间都花在哪，写性能预算书 . . . 251
8.7.2 实验二（11 月 2 日）：引擎模式开起来，逐帧计时基线立起来 . . . 251
8.7.3 实验三（11 月 23 日）：复现第 7 组的公开版 v1，指标对得上账 . . . 251
8.7.4 实验四（12 月 21 日）：三档权衡表与 batch 方案，招牌交付收官 . . . 252

**第 9 组 水下双目 · 数据组 253**
9.1 组档案 . . . 253
9.2 项目背景与你的位置 . . . 253
9.3 核心任务 . . . 254
9.4 第 19 周周五提交清单 . . . 256
9.5 AI 分工与人工必做 . . . 256
9.6 防糊弄指南与满分标准 . . . 257
9.7 四次实验安排 . . . 257
9.7.1 实验一（10 月 12 日）：给视频开体检单，你亲自看 30 帧 . . . 258
9.7.2 实验二（11 月 2 日）：抽帧去冗一条龙，切分防住泄漏 . . . 258
9.7.3 实验三（11 月 23 日）：数据组也要懂训练——亲眼看 loss 下降 . . . 258
9.7.4 实验四（12 月 21 日）：换配比、调阈值，用验证 EPE 选出更好的一口粮 . . . 259

**第 10 组 水下双目 · 部署组 260**
10.1 组档案 . . . 260
10.2 项目背景与你的位置 . . . 260
10.3 核心任务 . . . 261
10.4 第 19 周周五提交清单 . . . 262
10.5 AI 分工与人工必做 . . . 263
10.6 防糊弄指南与满分标准 . . . 263
10.7 四次实验安排 . . . 264
10.7.1 实验一（10 月 12 日）：让 agent 开清单，你逐条当审计 . . . 264
10.7.2 实验二（11 月 2 日）：装上 4080，跑通第一对图（本组主力） . . . 265
10.7.3 实验三（11 月 23 日）：微调护航——你的环境要能“载训练” . . . 265
10.7.4 实验四（12 月 21 日）：三组旋钮 + ONNX 初探，给出权衡建议 . . . 265

**第 11 组 水下双目 · 训练组 267**
11.1 组档案 . . . 267
11.2 项目背景与你的位置 . . . 267
11.3 核心任务 . . . 268
11.4 第 19 周周五提交清单 . . . 269
11.5 AI 分工与人工必做 . . . 270
11.6 防糊弄指南与满分标准 . . . 270
11.7 四次实验安排 . . . 271

11.7.1 实验一（10 月 12 日）：把 yaml 读成人话 . . . . . 272
11.7.2 实验二（11 月 2 日）：让一个 batch 走完全程 . . . . . 272
11.7.3 实验三（11 月 23 日）：先立基线，再开微调（本组主力） . . . . . 272
11.7.4 实验四（12 月 21 日）：一次只动一个旋钮 . . . . . 272

**第 12 组 水下双目 · 评测组 274**
12.1 组档案 . . . . . 274
12.2 项目背景与你的位置 . . . . . 274
12.3 核心任务 . . . . . 275
12.4 第 19 周周五提交清单 . . . . . 276
12.5 AI 分工与人工必做 . . . . . 276
12.6 防糊弄指南与满分标准 . . . . . 277
12.7 四次实验安排 . . . . . 278
12.7.1 实验一（10 月 12 日）：先把尺子造准 . . . . . 278
12.7.2 实验二（11 月 2 日）：一键打通“推理 → 视差 → 指标” . . . . . 278
12.7.3 实验三（11 月 23 日）：两版权重同场比武，每模型不少于 50 对 . . . . . 279
12.7.4 实验四（12 月 21 日）：扫阈值、找最差桶、当个好裁判 . . . . . 279

**第 13 组 数据基座 · 数据结构化组 280**
13.1 组档案 . . . . . 280
13.2 项目背景与你的位置 . . . . . 281
13.3 核心任务 . . . . . 281
13.4 第 19 周周五提交清单 . . . . . 282
13.5 AI 分工与人工必做 . . . . . 283
13.6 防糊弄指南与满分标准 . . . . . 283
13.7 四次实验安排 . . . . . 284
13.7.1 实验一（10 月 12 日）：让大模型讲原理，你拿样例当裁判 . . . . . 284
13.7.2 实验二（11 月 2 日）：解析环境装通，10 页流水线跑通 . . . . . 285
13.7.3 实验三（11 月 23 日）：自己造数据，训一个“认版面”的小模型 . . . . . 285
13.7.4 实验四（12 月 21 日）：在线大模型救疑难表格，规则法当对照组 . . . . . 285

**第 14 组 数据基座 · 智能问答与图检索组 287**
14.1 组档案 . . . . . 287
14.2 项目背景与你的位置 . . . . . 288
14.3 核心任务 . . . . . 288
14.4 第 19 周周五提交清单 . . . . . 290
14.5 AI 分工与人工必做 . . . . . 290
14.6 防糊弄指南与满分标准 . . . . . 290
14.7 四次实验安排 . . . . . 291
14.7.1 实验一（10 月 12 日）：当一回 agent，走通“查询-回答-引用” . . . . . 291
14.7.2 实验二（11 月 2 日）：本地小模型上岗，一句话问数跑通 . . . . . 292
14.7.3 实验三（11 月 23 日）：小图上真训练——两层 R-GCN 做链路预测 . . . . . 292
14.7.4 实验四（12 月 21 日）：本地 vs 在线——30 题上一张对比表 . . . . . 292

**第 15 组　数据基座 · 虚假内容识别与对抗训练组** **294**
15.1 组档案 . . . . . 294
15.2 项目背景与你的位置 . . . . . 295
15.3 核心任务 . . . . . 295
15.4 第 19 周周五提交清单 . . . . . 297
15.5 AI 分工与人工必做 . . . . . 297
15.6 防糊弄指南与满分标准 . . . . . 298
15.7 四次实验安排 . . . . . 299
15.7.1 实验一（10 月 12 日）：先当一回受害者——AI 造假你来判 . . . . . 299
15.7.2 实验二（11 月 2 日）：规则当第一道闸，词袋小模型当基线 . . . . . 299
15.7.3 实验三（11 月 23 日）：主力会战——500 条数据集、两个模型、一份自查 . . . 299
15.7.4 实验四（12 月 21 日）：在线判官复核，蒸馏回流再上一分 . . . . . 300

**附录：参考资料与图片来源** **301**

# 第一部分

# 第一篇　理论篇：十二讲，从感知器到大模型

# 第 0 讲　开课导引：这门课怎么上、怎么学、怎么考

欢迎来到《深度学习与图像识别》。在翻开第 1 讲之前，请先把这一章读完——它回答你此刻最关心的四个问题：这门课到底讲什么？我要加入的真实项目是什么？开学前该准备什么？以及，怎么让大模型帮你学编程。最后一个问题是全书最重要的方法论，没有之一：这门课不禁止 AI，恰恰相反，**这门课要求你用 AI**——但用的是门手艺，得先学。

**本章地图**

**讲什么**：课程全貌（12讲主线与 4 次实验）、三个真实项目速览、开学前的准备清单、与大模型协作学编程的方法、零基础数学包与零基础 Python 包、全书地图。**为什么值得读**：这门课一半的分数在项目里，项目的活儿全靠 AI 协作完成——先弄清“怎么用 AI”和“需要什么基础”，后面 12 讲才不会手忙脚乱。**学完你能**：装好环境、跑通第一行 Python、写出让 AI 听懂需求的提示词、拿着全书地图知道自己每一步在往哪儿走。

## 0.1　这门课讲什么：从一个神经元到大模型

如果把整门课压缩成一句话，那就是**从一个神经元，走到大模型**。12讲内容基本按历史顺序展开，恰好重演了人工智能七十多年的主线：1958 年的感知器（第 1 讲）长出了多层结构（第 2 讲），学会了正经的训练流程（第 3 讲）和自我压缩（第 4 讲）；1970 年代末卷积思路登场，网络第一次真正“看懂”图像（第 5–8 讲）；随后网络学会处理序列与语言（第 9–10 讲），最终靠注意力机制（第 11 讲）走到了今天的大模型，还学会了“无中生有”地生成图像（第 12 讲）。每讲都配有课后编程题——题目统一要求你**指挥 AI 实现**本讲核心内容的一个小应用。

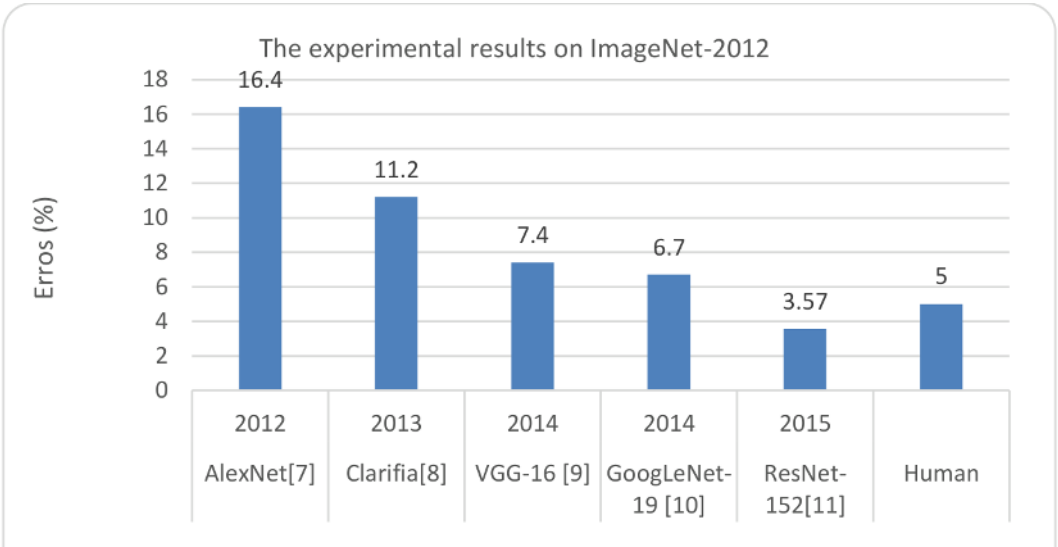



**图 1**　深度学习七十余年发展时间线：从 1943 年的神经元数学模型，到 2006 年“深度学习”命名，再到 2020 年代的大模型

图片来源：维基百科 History of artificial neural networks 词条

下表是 12 讲的路线图，右列是配套教材（许庆阳《深度学习模型与算法基础》，清华大学出版社 2023 年 12 月版）的课后阅读页码——每讲上课前请把对应页码过一遍，带着问题来听。

**表 1**　12 讲路线图：主题、一句话与教材页码

| 讲次 | 主题 | 一句话 | 教材页码 |
|---|---|---|---|
| 1 | 感知器、线性单元与学习规则 | 一个神经元如何从错误中学习 | 11–19 |
| 2 | BP神经网络、反向传播与梯度检查 | 把神经元叠起来，误差“责任倒推” | 20–30 |
| 3 | 超参数、权重初始化、模型训练与评估 | 训练的“火候”与考试的“卷子” | 31–35、48–50 |
| 4 | 自编码器 | 没有标签，网络自己学会压缩 | 35–48 |
| 5 | 卷积神经网络原理与前向计算 | 用滑动窗口看图像 | 50–70 |
| 6 | 卷积神经网络的训练与计算模式 | 数据、增强与训练技巧 | 71–85 |
| 7 | 卷积神经网络的演化 | 从 LeNet 到 ResNet 的三十年 | 86–100 |
| 8 | 轻量化网络、全卷积网络 | 把网络塞进手机，看清每个像素 | 101–115 |
| 9 | 循环神经网络 RNN | 会顺着序列思考的网络 | 167–186 |
| 10 | 长短时记忆网络 LSTM 与 GRU | 给网络装上带闸门的记忆 | 187–203 |
| 11 | 注意力模型 | 大模型时代的地基 | 204–229 |
| 12 | 生成对抗模型 | AI 如何“无中生有”画图 | 230–248 |

再说教学形式。本课程 2 学分、32 学时，拆成两半：**24 学时随堂讲授**（就是第一篇的 12 讲，每次 2 学时讲一讲），以及 **8 学时的 4 次现场实验**。实验不是“照着指导书按按钮”的验证作业：第一次实验你就把大模型编程助手配置进自己的编辑器，此后所有实验和项目作业都默认你在与 AI 协作。四次实验的时间与内容如下。

**表 2**　四次现场实验一览（实验二至四各 10 分共 30 分，实验一不计分；平时作业 20 分、项目成果 50 分，总评满分 100 分）

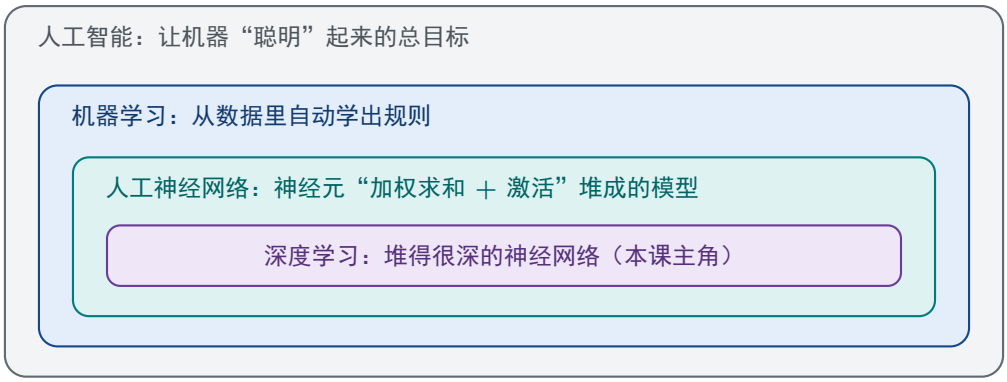
| 序号 | 时间 | 内容 |
|---|---|---|
| 实验一 | 10 月 12 日 | 大模型 agent 模式配置及应用：在 VS Code 中安装 CLINE 编程助手，连接 GLM 或 DeepSeek 大模型 API |
| 实验二 | 11 月 2 日 | 匹配合适程序并执行本地部署：用 Ollama 在自己电脑上运行开源大模型、编译项目程序 |
| 实验三 | 11 月 23 日 | 深度网络的模型训练实践：亲手跑通第 2–3 讲的训练循环 |
| 实验四 | 12 月 21 日 | 模型优化训练实践：把训练跑得又快又稳 |

除了课堂与实验，你的每周预算里还有 **6–8 小时的小组项目时间**：全班按 5 人一组分成 15 个小组，分属三个真实工程项目（下一节就介绍）。第 19 周周五（2027 年 1 月 22 日）全部成果提交验收。注意这个时间结构：**项目不是课程结束后的“大作业”，而是与课程同步进行的“实习”**——开学头几周你就会进入项目组、拿到任务书；此后每次课上学的每一样东西，几周内都会在你的项目里用到。

### 0.1.1　先排座次：人工智能、机器学习、神经网络与深度学习

正文开始之前，先把四个最出镜的词排好座次。它们不是四门并列的学科，而是**一圈套一圈的包含关系**：人工智能 ⊃ 机器学习 ⊃ 人工神经网络 ⊃ 深度学习。各用一句话认个脸：

- **人工智能（AI）**：最大的圈——让机器做出类似人类智能行为的一切尝试的总称，这个名字是 1956 年达特茅斯会议起的；
- **机器学习（ML）**：人工智能里最成功的一个流派——不靠工程师手写规则，让机器从数据里自己把规则学出来；
- **人工神经网络（NN）**：机器学习里的一种模型——用第 1 讲那种“加权求和、过阈值激活”的神经元堆叠连接而成；
- **深度学习（DL）**：把神经网络堆得很深（很多层）——本课程的主角，也是 2026 年所有大模型的底座。




**零基础扫盲：两个教材高频词——“表示学习”与“端到端学习”**

配套教材开篇会反复出现这两个词，这里提前用人话讲一遍，全书统一按这个意思使用。**表示学习（representation learning）**：同样一堆东西，“怎么记”差别巨大。早期的 One-hot 表示像一本超长点名册：一万个类别就造一个一万维向量，只有代表自己的那个位置是 1、其余全是 0——占地方，而且任何两个类别之间“谁跟谁像”的信息量为零。**分布式表示**换成另一条路：用一小段稠密数值（比如 64 个数）拼出万物，每个维度都在说话，含义相近的东西向量也挨得近。把“点名册”学成这种稠密向量的过程，就叫**特征嵌入**（embedding）——大模型里“词向量”三个字的来历。一句话：**好的表示不是人定的，是学出来的**；第 4 讲整讲的训练，练的就是这门功课。**端到端学习（end-to-end learning）**：老办法把一个任务切成好几段，每段手工设计一个算法、各自调参、层层接力；端到端学习把整条流水线换成一个网络——原始数据从这头进，最终答案从那头出，中间全部交给训练自动打磨。一句话：从“层层外包”到“一家全包”。你们六目拼接项目里那条“端到端路线”（第 一 章），名字就是这么来的。

## 0.2　三个真实项目一页速览

这门课最大的不同，是你们要干的活儿来自三个**正在运行的真实工程项目**，而不是教科书后面的习题。下图是整门课 16 周的全景时间轴：绿色是 12 讲理论课，橙色是 4 次实验，蓝色是项目关键节点，红色是第 19 周的提交截止线——四条线并行推进，这张图值得贴在宿舍墙上。




**图 2**　课程 16 周全景时间轴：12 讲理论课（绿）、4 次现场实验（橙）、项目关键节点（蓝）与第 19 周成果提交（红）

三个项目的速览如下表，随后每个项目用一段人话介绍。更完整的背景（甲方是谁、现场长什么样、难点在哪）请阅读第二篇各章，那里每个项目有一整章。

**表 3**　三个真实项目速览（15 个小组，每组 5 人）

| 项目 | 甲方 / 背景 | 做什么 | 组数 | 关键词 |
|---|---|---|---|---|
| 六目拼接 | 360Stitcher 系统总体组 | 六路相机实时 360 度全景拼接：把九类人工调参升级为九个可学习的“参数头”神经网络 | 8 | 实时全景、ONNX、回退保护 |
| 水下双目自动标记 | 中海油 HYSY123 FPSO | 为 ROV 水下检查视频建立双目立体感知：大模型“教师”打伪标签，训练轻量“学生”模型 | 4 | 双目立体、伪标签、锚链 |
| 数据基座治理 | 海洋石油平台 | 治理 126 份 CSV 共 3.1 亿行运行数据与 1001 页工程手册：结构化、一句话问数、虚假内容识别与对抗训练 | 3 | 数据治理、智能体、R-GCN |

**项目一：六目拼接。**想象六只“眼睛”围成一圈盯着海上平台四周，各自拍一路视频，系统要把六路画面实时拼成一幅无缝的 360 度全景。传统拼接算法里有九类参数（畸变校正、特征匹配、对齐、融合……），过去全靠工程师一把一把手工调，换个相机、换个光照就要重调一遍。这个项目做的事，是把这九类参数变成九个**可学习的神经网络“参数头”**：模型自己算参数，系统负责拼接，算完还要转成 ONNX 格式交付，并配一道“回退保护”——模型万一失手，系统自动退回老算法，现场一分一秒都不能停。项目里另有一条端到端路线作对照，用 RTX 4080 训练。详见第 一 章。

**项目二：水下双目视觉自动标记。**HYSY123 是一座漂浮在海上、排水量十几万吨的浮式生产储卸油装置（FPSO），它靠几条巨大的系泊锚链“锚”在油田上方。锚链要靠水下机器人（ROV）定期下水检查，但 ROV 拍回的视频浩如烟海，人工打标签慢、贵、还不稳定。这个项目用**双目相机**的左右两幅画面算“视差”、反推深度，让机器自动识别锚链的每一环；再用 FoundationStereo 这样的大模型当“教师”给画面打伪标签，过滤掉不可靠的，拿来训练轻量的“学生”模型——你们还要自己搭建 500 对锚链立体图对作为家底。详见第 二 章。

**项目三：数据基座治理。**某海洋石油平台的运行数据是 126 份 CSV 文件、共 3.1 亿行，配上 1001 页工程手册——靠人翻，一辈子翻不完。这个项目干三件事：把散乱的数据和手册**结构化**；搭一个**“一句话问数”**的智能体（工程师说人话，Gemma 大模型加 DuckDB 数据库加 R-GCN 图检索负

责听懂并查数）；以及**虚假内容识别与对抗训练**——万一有人往系统里喂假数据怎么办？那就训练一个“监考员”模型来抓，造假方与监考方反复过招、共同变强。详见第 三 章。

**项目连线：理论 12 讲与三个项目的关系**

不要小看速览表里那几行字，这门课的每一讲都有一条暗线通向它们：第 1–3 讲的神经网络基础，是九个参数头、学生模型、监考员模型的共同骨架；第 5–8 讲的卷积网络，负责六目拼接的图像特征与水下画面理解；第 9–11 讲的序列模型与注意力，是“一句话问数”智能体的引擎；第 12 讲的对抗思想，正是数据基座“造假者 vs 监考员”拉锯战的理论原型。**第一篇每讲末尾的“项目连线”盒子会反复提醒你这条暗线**——学理论时知道自己为什么学。

## 0.3　你需要准备什么

工欲善其事，必先利其器。第一次实验（10月 12 日，配置大模型编程助手）不允许空手进场，请在开学两周内对照下面的清单备齐装备。总共四样：**一台能装 Python 的电脑**、**一个大模型 API 账号**、**20 GB 硬盘空间**，以及最重要的——**好奇心**。

**表 4**　电脑最低配置与账号清单

| 项目 | 最低要求 | 说明 |
|---|---|---|
| CPU | 4核 | 无特殊要求 |
| 内存 | 8 GB | 16 GB 更从容 |
| 硬盘可用空间 | 20 GB | 放 Python 环境、模型文件与项目数据 |
| 显卡 | 不强制 | 有 NVIDIA 显卡（8 GB 显存以上）跑本地模型更快；没有也完全能完成本课 |
| 操作系统 | Windows 10/11 或 macOS | 实验机房为 Windows |
| 大模型 API 账号 | GLM 或 DeepSeek 任一 | GLM 开放平台（open.bigmodel.cn）或 DeepSeek 开放平台（platform.deepseek.com），注册即有免费额度，实验一要用 |

这里解释一下“API 账号”是什么。大模型跑在厂商的机房里，你的程序通过 **API**（Application Programming Interface，应用程序接口）把问题发过去、把答案收回来，就像点外卖：厨房在远处，你在窗口下单取餐。注册后你会拿到一串“钥匙”（API Key），实验一里把它填进编程助手，AI 就在你的编辑器里上岗了。除了云端点餐，课程后半段还会学**本地部署**——用 Ollama 把开源模型整个搬到自己电脑上跑（实验二），那时你的笔记本就是厨房。

准备好了硬件，接下来干第一件实事。

**动手时间：第一件事，跑通你的第一行 Python**

**目标：**在一台全新电脑上，看到你自己跑出来的第一行程序输出。全程允许（鼓励）问 AI。

**步骤：**(1)按下一节 0.6 的四步装好 Anaconda 与 VS Code；(2) 打开 VS Code，新建文件 `hello_ai.py`，敲入下面一行；(3) 点右上角“运行”，在下方终端里看到输出。

```
1  print("Hello AI")   # print 是"打印": 把括号里的内容显示到屏幕
```

**如果卡住了，**把这句话发给你的 AI 助手（没装助手就先在网页版大模型里问）：

> “我是完全零基础的学生，电脑是 Windows 11，从没装过 Python。请一步一步教我安装 Anaconda

和 VS Code，并运行我的第一个程序 `print("Hello AI")`。每一步告诉我点什么按钮，出任何报错我会把原文发给你，你要先用一句话告诉我错误原因，再给修改办法。”

**验收**：屏幕上出现 `Hello AI`。就这么简单——但请注意，这句话已经被你“指挥 AI 从零搭环境”跑通了，这正是四次实验和全部课后题的标准工作方式。

## 0.4　如何用大模型帮你学编程（本书最重要的方法论）

先说结论：这门课里，**“会写提示词”比“会背语法”值钱**。你不需要从零手写每一行代码，但你必须能看懂 AI 写了什么、能改动它、能向别人讲清楚它为什么这么写。本节把这套方法拆成四件事：工具、提示词、agent 循环、以及不可逾越的底线。

### 0.4.1　AI 编程助手是什么

**VS Code** 是微软出品的免费代码编辑器，是目前 AI 编程生态的主战场；**Cline** 是运行在 VS Code 里的开源 AI 编程助手插件（官网 cline.bot），背后连接你配置的大模型（GLM 或 DeepSeek）。它在对话框里听你说人话，然后替你新建文件、写代码、保存、运行、看报错、自己修改、再运行——直到任务完成。你在整个过程中的角色，从“打字员”变成了“提需求、做验收的甲方”。

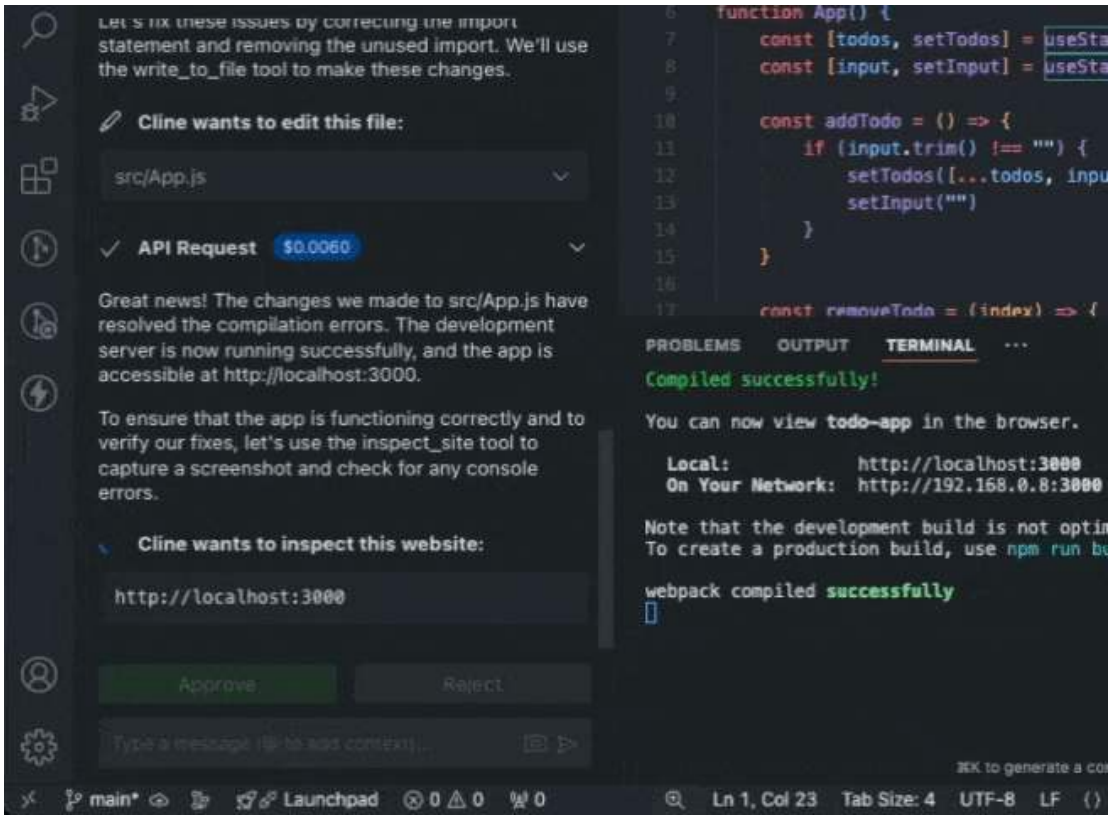



**图 3**　VS Code 与 Cline AI 编程助手：左侧对话框下达任务，助手自动创建并修改右侧的代码文件，还能代为执行命令

图片来源：CLINE 开源项目官方文档 cline.bot

### 0.4.2　提示词怎么写：四个要素

提示词（prompt）就是你发给 AI 的指令。写得含糊，AI 就用最平庸的方式糊弄你；写得具体，它就是顶尖助教。四个要素，配一个“反例 / 正例”对照表：

1. **具体**：说清语言、库、输入输出、参数，别让 AI 猜；
2. **给上下文**：你的系统、已装的环境、学过什么、报错原文；
3. **要解释**：要求逐行中文注释、解释每步在干嘛——这是你“看得懂”的保险；
4. **迭代追问**：第一次不满意就继续调，像带实习生一样带它。

**表 5**　提示词四要素的“反例 / 正例”对照

| 要素 | 反面示例 | 正面示例 |
|---|---|---|
| 具体 | “帮我写个神经网络” | “用 numpy 写一个 2-2-1 全连接网络，tanh 激活，平方损失，学习率 0.1” |
| 给上下文 | （什么都不说） | “我的电脑是 Win11 + Anaconda，只学过 C 语言，numpy 没装过” |
| 要解释 | “直接给我代码” | “每行加中文注释，写完用三句话讲清整体思路” |
| 迭代追问 | 不满意就放弃 | “把学习率改成 0.01 再跑一遍，并解释为什么这次收敛更稳” |

### 0.4.3　agent 循环：AI 是怎么“自己干活”的

你可能会奇怪：AI又看不到你的屏幕，它怎么帮你把程序跑通？答案是它运行在一个**循环**里：**思考**（分析当前局面）→ **行动**（写文件、执行命令）→ **观察**（读取运行结果和报错）→ 再思考……这个“思考–行动–观察”的循环（下图）就是 2026 年所有 AI 编程助手的核心机制，学术源头是 2022 年的 ReAct 论文。Cline 把这个循环用在了你的项目目录上：它改一行代码就编译一次，看见报错自己分析自己改，跑通了才向你汇报。第一次实验（10 月 12 日）你要亲手把这套循环配置起来。

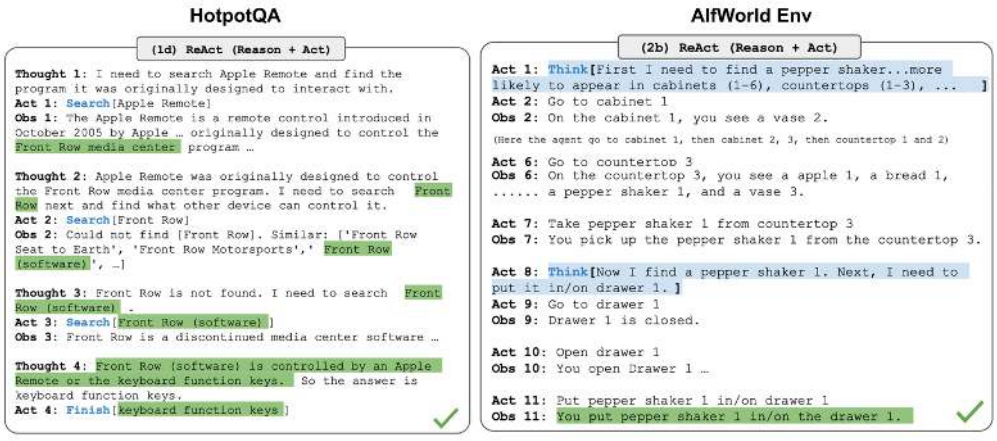



**图 4**　大模型智能体（Agent）的工作循环：思考（Reasoning）→ 行动（Acting）→ 观察（Observing），循环往复直到完成任务

图片来源：ReAct: Synergizing Reasoning and Acting in Language Models, Yao et al., 2022

**前沿速递：从“补全下一行”到“自己干活”（2024–2026）**

如果你在 2023 年用过 AI 写代码，印象大概是“高级自动补全”：打半行，它猜半行。2024 年起，风向变了——编程助手普遍进化为 **agent 模式**：接下完整任务，自己拆解、自己改文件、自己跑测试。2025 年，“氛围编程”（vibe coding）一词被造出来并迅速当选各大词典年度热词：人只描述意图和验收标准，AI 写代码，人负责把关——这正是本课要求你的工作方式。衡量这种能力的权威基准（在真实软件仓库里修复真实 bug 的 SWE-bench 等）成绩在 2024–2026 年间逐年跳涨，头部大模型在标准工程任务上的修复率已从两成上下爬到与熟练工程师相当的水平。你在实验一里配置的 agent，就是这条演化路线的最新形态。顺便说一句：本讲义的部分代码示例与全部插图脚本，也是作者与大模型协作完成的——这本身就是教学内容。

底线必须先讲清楚。

**AI 可以用，结果自己扛**

三条纪律，违反任何一条，视同作业未完成：

1. **不许只抄不看**。每个交付物都可能被随机抽问，讲不清楚就是没做。代码是 AI 写的没关系，**讲不清楚就是没做**。
2. **实验数字不许抄 AI 给的**。准确率、误差、运行时间，必须是你自己的代码在你的数据上跑出来的，交付要有“命令 + 日志 + 产物”三件套证据。
3. **AI 会一本正经地胡说**。这叫幻觉（hallucination）：它可能给你不存在的函数、虚构的结论、“99% 的精度”。它说什么不算数，你跑出来才算数。

工程世界里，说谎的代价不是扣分，是事故。这门课从第一行代码起就按工程标准要求你。

最后，既然整本书都建立在“大模型”上，那大模型本身是什么？这里先用三句话扫个盲，原理在第 11 讲细讲。

**零基础扫盲：大模型到底是个啥？三句话**

**第一句**：它本质是一台**超大规模的文字接龙机器**——你说上半句，它计算“下一个字接什么最合适”，一个字一个字往外蹦。**第二句**：它“读”过互联网上的海量文本，训练过程就是反复调整内部几千亿个**参数**（可以想象成几千亿个“重视程度”旋钮），把“接得像”磨成“接得对”。**第三句**：它靠**注意力机制**在长文本里挑重点（哪句话和当前问题有关就多看哪句），这是它能力的关键，也是第 11 讲的主角。就这些——不神秘，也没有“意识”，但它足够好用，好用到你这门课的成绩里有五十要靠它。

## 0.5　零基础数学包：够用就好的三小块

你们学过高等数学、线性代数、概率论，但多半已经还给老师了。没关系——这本书用到的数学就三小块，而且每一块都只需要“会算、能理解”的程度。以下三个蓝色盒子现场补课，后文用到时会回指。

**知识补齐：向量与点积——“合拍程度”打分器**

**向量（vector）**就是一串有序的数，例如 $\mathbf{a}=(1,2)$，几何上画成一支带方向的箭头。**点积（dot product）**：两个向量对应位置相乘再相加：

$$\mathbf{a}\cdot\mathbf{b} \;=\; a_1b_1+a_2b_2+\cdots+a_nb_n. \tag{1}$$

例如 $\mathbf{a}=(1,2)$、$\mathbf{b}=(2,1)$，点积 $=1\times2+2\times1=4$。**人话翻译：**点积是给两个向量的“合拍程度”打分——方向越一致分越高，互相垂直得 0 分，方向相反得负分。下图里两支箭头的夹角越小，点积越大。第 1 讲的感知器，干的就是这件事：拿“权重向量”给“样本向量”打合拍分。

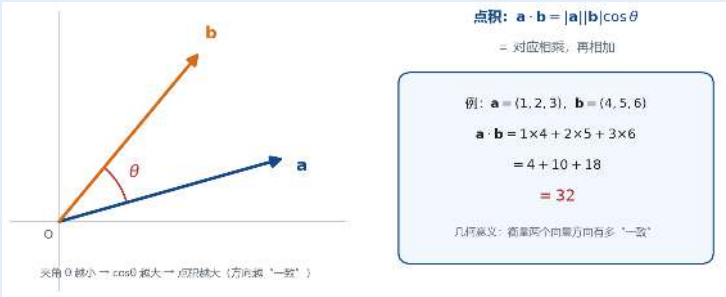



图 5　向量与点积示意：两个向量做“对应相乘再相加”；夹角越小（方向越一致），点积得分越高

**知识补齐：导数与链式法则——“变化率”与“变化率相乘”**

**导数（derivative）**：函数此刻的变化率，几何上是曲线在该点的斜率，记作 $f'(x)$ 或 $\frac{\mathrm{d}y}{\mathrm{d}x}$。导数大，意味着 $x$ 动一点、$y$ 就动很多。例如 $f(x)=x^2$ 的导数是 $f'(x)=2x$：在 $x=3$ 处斜率为 6。**链式法则（chain rule）**：如果 $y$ 随 $u$ 变、$u$ 又随 $x$ 变（复合函数），那么总变化率等于两段变化率相乘：

$$\frac{\mathrm{d}y}{\mathrm{d}x}=\frac{\mathrm{d}y}{\mathrm{d}u}\cdot\frac{\mathrm{d}u}{\mathrm{d}x}. \tag{2}$$

**人话翻译：**像汇率换算——人民币兑美元一段汇率，美元兑日元一段汇率，两段相乘就是人民币兑日元的总汇率。下图里 $x$ 先影响 $u$、$u$ 再影响 $y$，导数就沿着箭头一路乘过去。第 2 讲的反向传播，本质就是链式法则在网络上的规模化使用。

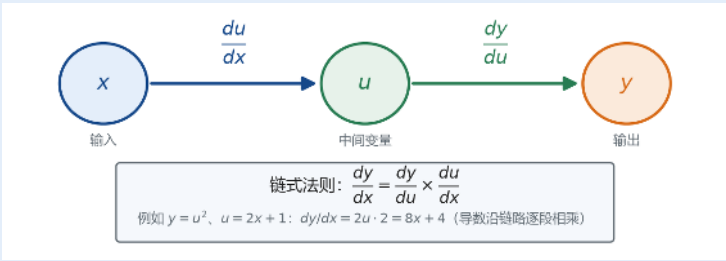



图 6　链式法则示意：$x$ 先影响 $u$、$u$ 再影响 $y$，总变化率 = 两段变化率相乘，导数沿箭头传递

**知识补齐：概率三句话**

**第一句：**概率（probability）是可能性的度量，取值 0 到 1；所有可能情况的概率加起来等于 1。**第二句：**条件概率 $P(A\mid B)$ 读作“在 B 已经发生的前提下，A 发生的概率”。比如“阴天时下雨”的概率就比“平时下雨”大——前提改变了判断。模型说“这张图是猫的概率 0.97”，说的就是一

个条件概率（前提是这张图的像素）。**第三句：**期望（expectation）是加权平均：$E[X]=\sum_i p_i x_i$，把每种结果按发生的可能性加权平均，回答“平均而言会怎样”。第 3 讲评估模型时，“平均误差”就是期望。

## 0.6　零基础 Python 包：从装环境到会读代码

为什么这门课用 Python，而不是你们学过的 C++？一张表说清分工：

表 6　Python、C/C++ 与 AI 工具链在本课中的分工

| 工具 | 它是什么 | 在本课的角色 |
|---|---|---|
| Python | 语法接近自然语言的解释型语言 | **主力**：写模型、调库、画图；也是大模型最擅长写的语言 |
| C / C++ | 编译型语言，你们已学过 | **底层**：PyTorch、ONNX运行时的内核是 C++，被 Python 调用，本课不必直接写 |
| Anaconda | Python 的“全家桶”发行版 + 环境管理器 | **地基**：统一全班环境，避免“我电脑上装不上” |
| VS Code + Cline | 编辑器 + AI 编程助手插件 | **结对程序员**：你说需求，它写代码、跑程序 |
| Ollama | 本地大模型运行器 | **实验二**：把开源模型搬回自己电脑跑 |

环境安装四步走，跟着下图从左到右：(1) 官网下载并安装 Anaconda（一路下一步）；(2) 安装 VS Code，在插件市场装 Python 和 Cline 两个插件，并在 Cline 里填入你的 API Key；(3) 新建一个 `.py` 文件；(4) 点运行按钮。哪一步卡住，就把报错原文发给 AI（见 0.3 节的求助提示词）。




图 7　Python 环境安装四步：安装 Anaconda → 安装 VS Code 与插件 → 新建 `.py` 文件 → 一键运行

环境有了，最后补一条最薄的底：读代码。你不需要会写，但需要看懂。下面 13 行覆盖了本课代码九成以上的语法，每行都值得读三遍：

```
x = 3                       # 变量：直接起名字赋值，不用声明类型
pi = 3.14159                # 小数叫浮点数
name = "robot"              # 引号包起来的是字符串
scores = [90, 85, 77]       # 列表 list：一串值排好队，下标从 0 开始
scores[0]                   # 取第 1 个：90（注意是 0 不是 1！）
info = {"name": "g01", "n": 5}  # 字典 dict：键值对，像查表
info["n"]                   # 按键取值：5
if x > 0:                   # if 判断：注意行末冒号
    print("正数")            # 缩进 4 格表示"属于 if"——缩进就是语法！
for s in scores:            # for 循环：逐个取出列表里的元素
    print(s)                # 会打印 90、85、77 三行
def add(a, b):              # def 定义函数：输入 a、b，返回 a+b
    return a + b
```

```
import math                 # import 引入现成的工具库
math.sqrt(16)               # 调用库函数：4.0
```

三条铁律：**冒号 + 缩进**定归属（缩进错了程序就错）；**下标从 0 开始**；**报错先读最后一行**。除此之外的语法，遇到就问 AI，现学现用完全来得及。

**零基础扫盲：报错了怎么办？**

初学者看见满屏红字的第一反应是“完了”。其实报错是程序在跟你说话，三步应对：**(1) 倒着读：**Python 报错（Traceback）最后一行才是病因——`NameError` 多半是名字拼错，`IndentationError` 是缩进不对，`TypeError` 是拿数字当字符串用了。**(2) 整段复制给 AI：**把报错原文和你的代码一起发过去，问“哪错了、为什么、怎么改”，这是最高效的排错方式。**(3) 实在不行就搜索：**把报错信息原样粘进搜索引擎，九成错误别人都犯过。记住：**报错不是失败，是免费的纠错教练。**

## 0.7　全书地图：三篇怎么配合用

最后交代全书的结构。全书三篇对应课程三个阶段，用法各不相同：**第一篇（理论篇，12 讲）**随堂推进，是每天的主粮——每讲有故事、有公式、有人话翻译、有防糊弄提醒，讲末有一道“用 AI 实现”的课后编程题和一张术语表；**第二篇（项目背景篇）**三个项目各一章，建议开学前两周读完，你拿到任务书时才不会两眼一抹黑；**第三篇（任务书篇）**是 15 个小组的作战地图，含每组任务、提交清单、AI 分工与验收标准，以及四次实验的命令级步骤——这一篇是工具书，不必通读，但每个节点都要对照执行。

正文之前，还要统一三个词的日常用法，全书不再重复解释。**“网络”**（或“神经网络”）：一个由大量简单计算单元组成的函数——吃进数字，吐出数字，中间全是乘法、加法和一点点非线性，它不是生物意义上的网络。**“模型”**：训练好参数的网络，物理上就是一个几 MB 到几百 GB 的文件；平时说“下载一个模型”，就是下载这份装满参数的文件。**“训练”**：用数据自动调整网络参数的过程——第 1–2 讲你会亲手做这件事，做完你就再也不会觉得这个词玄乎了。其余术语在各讲首次出现时都会给“中文（English）”和一句人话，章末术语表可随时回查。

**大模型（Large Language Model, LLM）**
参数规模巨大、在海量文本上训练的语言模型，能接龙式地生成文字，也能被改造去写代码、看图。

**参数（parameter）**
网络内部可调整的数值，多数是“权重”与“偏置”。训练就是调参数；参数量是模型规模的第一指标。

**训练（training）**
用数据自动调整参数、让网络输出越来越接近正确答案的过程。

**推理（inference）**
训练完成后用网络做预测的过程——“跑模型”通常指推理。

**数据集（dataset）**
一批带标注（或不含标注）的数据，训练与评估的原料。

**API**
应用程序接口；这里特指通过网络调用云端大模型服务的方式，需要注册账号并持有密钥。

**提示词（prompt）**

发给大模型的指令文本。提示词质量直接决定产出质量，0.4 节的四要素是本书的通用写法。

**智能体（Agent）**

让大模型进入“思考–行动–观察”循环、能使用工具（写文件、跑命令）自主完成任务的系统，如 Cline。

**幻觉（hallucination）**

大模型一本正经地生成错误内容的现象。对策：验收、复算、要证据。

**本地部署（local deployment）**

把开源模型下载到自己电脑上运行（工具如 Ollama），与调云端 API 相对；实验二的内容。

# 第 1 讲　感知器、线性单元与学习规则

1958年夏天，美国海军在华盛顿召开发布会，介绍一台“会自己学习的机器”。《纽约时报》的报道读起来激动得不像话：海军造出了一种电子计算机的雏形，**预计它有朝一日能走路、说话、看见、写字、自我复制，并意识到自己的存在**。这台机器叫“感知器”（Perceptron），主持建造的是康奈尔航空实验室的心理学家罗森布拉特（Frank Rosenblatt）。它当然没有这么神——事实上它只会做一件小事：盯着一张小图，分辨左右。但把夸张的新闻稿放到一边，这台机器身体里的核心想法——**加权和、阈值、从错误中调整权重**——一路演化了六十多年，变成了你今天可能天天在用的大模型。本讲的任务，就是拆开这台被吹上天的机器：看看它身体里装了什么想法，是什么让它栽了大跟头，又是什么让它笑到了最后。

那台机器后来被称作 Mark I 感知器，是一台你听得见“学习声音”的实体设备：400 个光导管负责“看”像素（这就是它的输入），每个输入连着一个由小马达带动的电位器——一个会自动旋转的旋钮，旋转角度就存着**权重**；训练时马达真的会转动旋钮，满屋子咔哒作响。1962 年罗森布拉特在书里写下的一句话，放在今天依然成立：“感知器的产生，不是为了模仿大脑的每一个细节，而是为了回答一个问题：机器能不能从自己的错误中学到东西？”

**本讲地图**

**讲什么**：生物神经元的工作方式、感知器的数学形式与决策边界、感知器学习规则、从 0/1 输出到连续输出的线性单元与最小均方、以及 1969 年的 XOR 危机与第一次 AI 寒冬。**为什么值得学**：感知器是所有神经网络的最小单元——本课三个项目里的每一个模型，剥到最里层都是它。**学完你能**：徒手算出一个感知器的输出；解释学习规则里每一项的含义；说清平方损失为什么长那样；给同学讲明白“第一次 AI 寒冬”的来龙去脉。

## 1.1　生物神经元：大脑的原始零件

要造人工神经元，先看真品长什么样。人脑约有 860 亿个神经元（neuron），每个神经元平均与几千个同伴相连。它的结构可以粗略想成三段：**树突**（dendrite）像无数根天线，负责从别的神经元接收电化学信号；**胞体**（soma）负责汇总所有收到的信号；当累计信号**超过某个阈值**时，胞体“点火”，沿着**轴突**（axon）——一根细长的输出线——发出一个电脉冲，把信号传给下一批神经元的树突。注意这个机制里最关键的两点：第一，多个输入要**加在一起**跟阈值比；第二，超阈值才放电，不超就沉默——“全或无”。

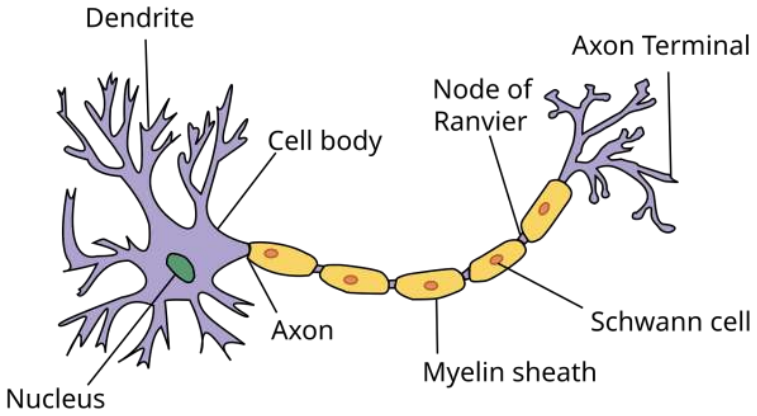



**图 1.1**　显微镜下的生物神经元：从胞体向外伸展的大量细长突起是接收信号的树突，那根最长的输出突起是轴突

图片来源：维基百科 Neuron 词条

那“学习”发生在哪里？生物层面，学习主要是调整**突触**——神经元之间的连接——的**强度**：常用的一条通路越用越通畅，不用的逐渐萎缩。1943年，神经生理学家麦卡洛克（McCulloch）与逻辑学家皮茨（Pitts）把上面这套机制抽象成了一个纯数学对象：**把每个输入乘上一个“强度”，加总，再和一个阈值比**——这就是神经元的第一个数学模型。下图把生物神经元和它的数学替身并排摆在一起，一一对应；本讲的余下部分，就是把右边这个数学替身讲透。

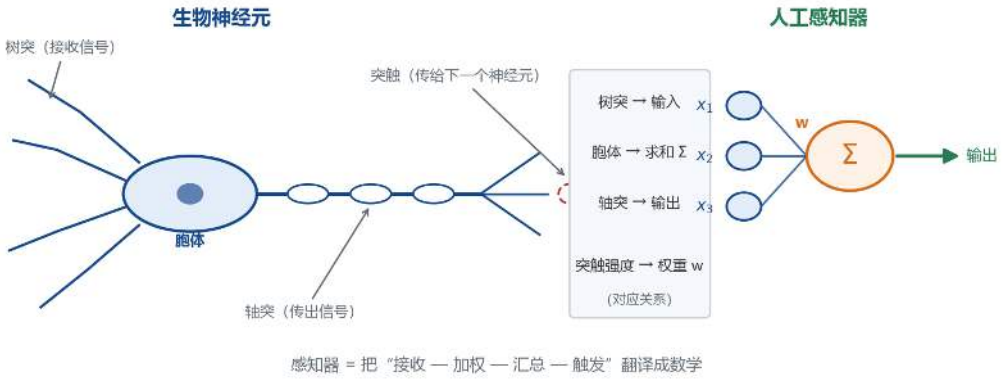



**图 1.2**　生物神经元与感知器对照：树突接收多路信号 ↔ 多路输入；胞体加权汇总 ↔ 求和节点；超过阈值轴突放电 ↔ 阶跃激活输出 0 或 1；突触连接强度可变 ↔ 权重可学习

## 1.2　感知器：一颗人工神经元

罗森布拉特做的，就是把 1943 年的抽象模型变成一台能学习的机器。感知器的全部计算只有一行：

$$y = f\left(\sum_{i=1}^{n} w_i x_i + b\right), \qquad f(z) = \begin{cases} 1, & z \geq 0, \\ 0, & z < 0. \end{cases} \tag{1.1}$$

**人话翻译**：收 $n$ 路输入 $x_1, \ldots, x_n$，每路乘上自己的权重（weight）$w_i$，加总后再加一个偏置（bias）$b$，最后过一道“及格线” $f$——信号总和不低于 0 就输出 1，否则输出 0。函数 $f$ 叫激活函数（activation function），感知器用的是**阶跃函数**（step function），直译就是“过线就响”。$\Sigma$（大写西格玛）是“连加”的数学记号，$\sum_{i=1}^{n} w_i x_i$ 读作“$w_i x_i$ 从 $i=1$ 一直加到 $i=n$”。

权重和偏置各有一个贴身的理解：**权重 = 重视程度**。$w_2 = 0.8$ 表示第二路输入很受重视，$w_3 = -0.2$ 表示第三路是“反向证据”——它越强你越不信。**偏置 = 门槛高低**：$b$ 越大，门槛越低，越容易被触发为 1。权重与偏置合称**参数**（parameter）——感知器的全部参数就这 $n+1$ 个数，所谓**训练**，就是调它们。这个定义先记住，全书还会用几十次。

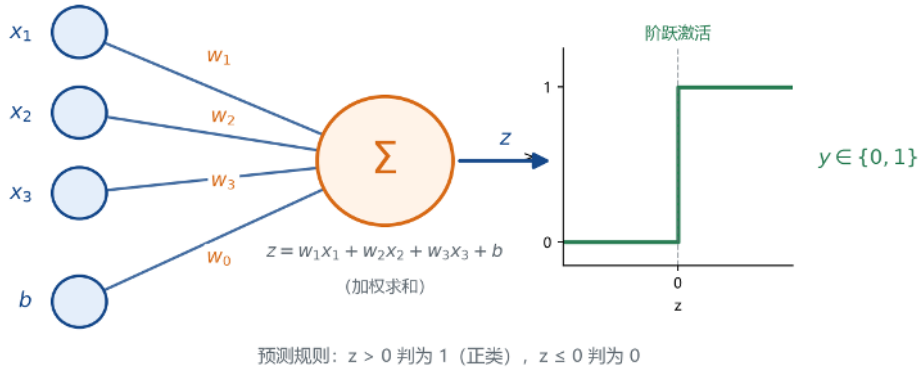



**图 1.3**　感知器结构：三路输入 $x_1, x_2, x_3$ 与偏置 $b$ 各自乘上权重后进入求和节点 $\Sigma$，总和经阶跃激活输出 0 或 1

动手算一遍，比看十遍都管用。

**动手时间：徒手算一个感知器**

设某感知器有三路输入，权重 $w = (0.6, -0.2, 0.8)$，偏置 $b = -1.0$。**第一步（示例）**：输入 $x = (1, 0, 1)$。加权和 $z = 0.6 \times 1 + (-0.2) \times 0 + 0.8 \times 1 + (-1.0) = 0.4 \geq 0$，输出 $y = 1$。注意：权重是负的那一路，乘的还是 0，没帮上忙也没拖后腿。**第二步（换你）**：分别算 $x = (0, 1, 1)$ 和 $x = (0, 0, 1)$ 的输出。（答案：$-0.4 \to 0$；$-0.2 \to 0$。都算对了吗？）**第三步（想一想）**：把 $w_2$ 从 $-0.2$ 改成 0.9，刚才 $x = (0, 1, 1)$ 的输出变吗？（$0.9 + 0.8 - 1.0 = 0.7 \geq 0$，输出变成 1。）——恭喜，你刚才**手工完成了一次“训练”**：为了纠正一个输出，改了一个参数。下一节要讲的，就是让机器自己干这件事。

## 1.3　决策边界：感知器在平面上画一条线

感知器的判断，在几何上有一个漂亮的形象。以两路输入为例，把每个样本看成平面上的一个点 $(x_1, x_2)$，输出由下式决定：

$$w_1 x_1 + w_2 x_2 + b = 0. \tag{1.2}$$

学过直线的你一眼就能认出来：这是**一条直线**。直线一侧的所有点，加权和都大于等于 0，感知器统一输出 1；另一侧统一输出 0。这条线就叫**决策边界**（decision boundary）——感知器并不“理解”你的问题，它只是在平面上找一条线，把两类点分开。下图里，误分类的点会在训练中把直线往正确的方向“推”（怎么推的，1.4 节见分晓）。

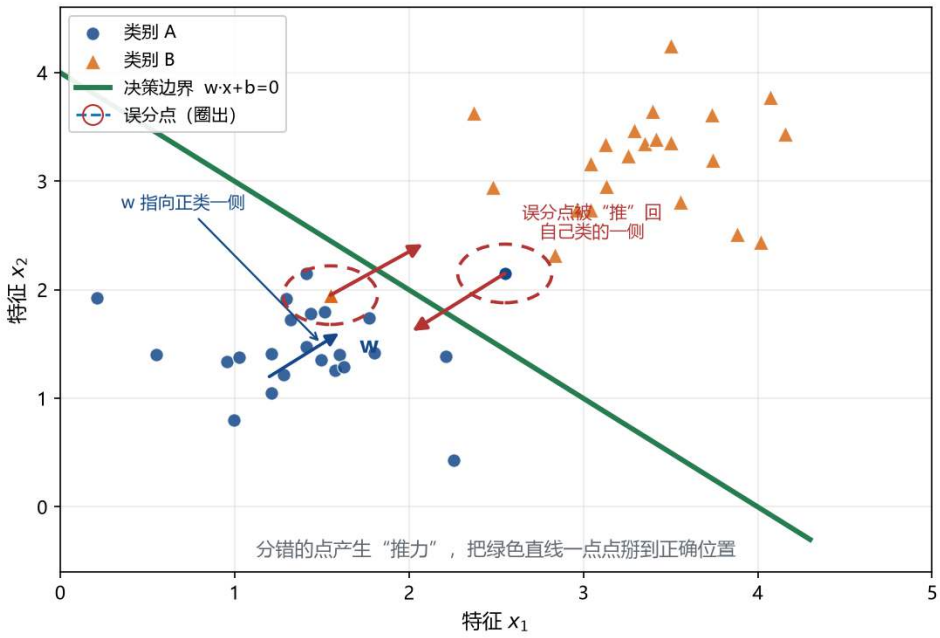



**图 1.4**　二维两类样本与一条决策直线：直线一侧判为正类（圆点）、另一侧判为负类（方块）；箭头示意误分样本会把直线往正确方向推

哪些问题能被一条线分开？最简单的逻辑运算就能说明问题。看下面的真值表：“与”（AND）、“或”（OR）都能一条直线分开（这类问题称为**线性可分**，linearly separable）；而“异或”（XOR，两输入**不同**才输出 1）却怎么摆直线都分不开——正类两点在对角线上，负类两点在另一条对角线上，一条直线永远顾此失彼。记住这个小小的“分不开”，它是 1.6 节一场大雪崩的引信。

**表 1.1**　简单逻辑运算的真值表：与、或线性可分，异或不可分

| 输入 | $(0,0)$ | $(0,1)$ | $(1,0)$ | $(1,1)$ | 一条直线分得开吗 |
|---|---|---|---|---|---|
| 与 AND（同时为 1 才输出 1） | 0 | 0 | 0 | 1 | 能 |
| 或 OR（有 1 就输出 1） | 0 | 1 | 1 | 1 | 能 |
| 异或 XOR（不同才输出 1） | 0 | 1 | 1 | 0 | **不能！** |

“一条直线分开两类”并不只是逻辑游戏，它是大量真实任务的雏形。统计学家费希尔 1936 年引入的鸢尾花数据集是机器学习界最著名的“入门数据”：三种鸢尾花，测量花瓣与花萼的长宽。下图里可以看到，其中两种花（setosa 与 versicolor）只用“花瓣长 × 花瓣宽”两个量画在平面上，几乎一条直线就能分开——本讲课后编程题就是它。至于分不开的场合怎么办，答案要到下一讲：把感知器叠起来。



**图 1.5**　鸢尾花数据集涉及的三种鸢尾花之一：本讲课后题用“刚毛鸢尾（setosa）”与“变色鸢尾（versicolor）”两类，花瓣长宽两个特征在平面上几乎一条直线即可分开

图片来源：维基百科 Iris flower data set 词条

## 1.4　它怎么学习：感知器学习规则

1958年那台满屋咔哒声的机器，是怎么“学习”的？答案朴素得惊人：**错了才改**。对每个带标准答案的样本，感知器先猜一个输出，猜对了就什么都不动；猜错了，就把各路权重朝着“更接近正确”的方向挪一小步。写成公式就是**感知器学习规则**（perceptron learning rule）：

$$w_i \leftarrow w_i + \eta\,(t - y)\,x_i, \qquad b \leftarrow b + \eta\,(t - y). \tag{1.3}$$

**人话翻译**：箭头“$\leftarrow$”读作“更新为”；$t$ 是标准答案（教师信号，teacher），$y$ 是感知器刚才的输出，$\eta$（读“伊塔”，希腊字母 eta）是**学习率**（learning rate）——每次修正的步子迈多大。分三种情形看：**猜对**（$t = y$，修正量为 0）——什么都不动；**把正类错判成 0**（$t - y = 1$）——该样本哪些输入是 1，对应权重就加上一小步，让下次更“重视”它们；**把负类错判成 1**（$t - y = -1$）——反向挪回来。偏置跟着同样修，相当于“顺便调门槛起伏”。下图是“一次错误如何推动直线”的形象版本。

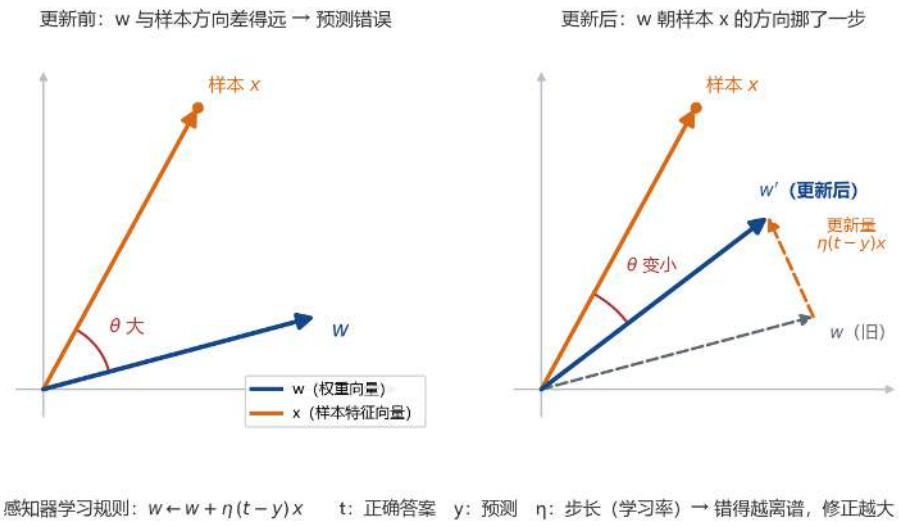



**图 1.6**　感知器学习规则的几何直觉：预测错了才更新——权重向量朝正确答案的方向挪一小步，学习率 $\eta$ 控制步长；直线随之转动与平移

为什么这个“挪一小步”总能把直线挪到正确的位置？这里要用到开课导引 0.5 节数学包里的点积（回看第 0 讲）：感知器的加权和 $z = \mathbf{w}\cdot\mathbf{x} + b$ 本质是给“样本 $\mathbf{x}$ 与权重 $\mathbf{w}$ 有多合拍”打分——夹角越小分越高。错把正类判成负类，说明这支“权重箭头”和样本“夹角太大”，修正量 $+\eta\,(t-y)\,\mathbf{x}$ 恰好把箭头**朝样本方向**扳近一点；反之扳远。一来二去，箭头指向正类那一片，直线就立起来了。

**知识补齐：向量与点积——感知器的打分器（回指第　讲）**

把输入写成向量 $\mathbf{x} = (x_1, \ldots, x_n)$、权重写成 $\mathbf{w} = (w_1, \ldots, w_n)$，加权和就是**点积**：

$$\mathbf{w}\cdot\mathbf{x} = w_1x_1 + w_2x_2 + \cdots + w_nx_n. \tag{1.4}$$

三个要点：**(1)** 点积 = 对应相乘再相加，正是感知器求和那一步；**(2)** 点积是“合拍程度”打分——两个向量方向越一致分越高，垂直为零，相反为负；**(3)** 决策边界 $\mathbf{w}\cdot\mathbf{x} + b = 0$ 就是“合拍分恰好抵消门槛”的等高线：直线始终与 $\mathbf{w}$ 垂直，$b$ 决定它离原点多远。后面各讲凡是出现“打分”“匹配度”“相似度”，底下多半是一次点积。

完整的训练流程是一个循环：拿一批带答案的样本，逐个**预测**；预测错了就按规则**更新**权重；整批过完称为一轮（epoch），再从头来第二轮、第三轮……直到某一整轮一个错都没犯。罗森布拉特在 1960 年证明了一条漂亮的定理：**只要两类确实线性可分，这个循环保证在有限步内停下来**——“能学会”是有数学保证的，不是玄学。至于“分不开”的那种情况，规则会永远改下去、永远学不会——这也正是下一节要谈的天花板。

听完感知器规则的全部故事，往回倒九年，去见一见它的祖师爷。1949 年，心理学家赫布（Donald Hebb）在《行为的组织》一书中提出：学习发生在**连接**上——如果神经元 A 反复参与点亮神经元 B，它们之间的突触连接就该变强。后人把这条假说浓缩成一句英文顺口溜：“Fire together, wire together”——**一起放电的神经元，连在一起**。

**知识补齐：Hebb 学习规则（1949）——学习规则的祖师爷**

赫布讲的是生物脑里的突触，搬到人工神经元上，就成了历史上第一条学习规则，一行写完：

$$w_i \leftarrow w_i + \eta\, x_i\, y. \tag{1.5}$$

人话：输入 $x_i$ 和输出 $y$ 同时活跃，权重就涨，涨多少正比于两者的“共同活跃度”——**被连接的两个神经元同时放电，连接就变强**。全程不需要标准答案：只要两路信号相关就连，因此它又叫**相关学习**。它和本节感知器规则的分工，一张表看清：

| | Hebb 规则（1949） | 感知器规则（1958） |
|---|---|---|
| 要标准答案吗 | 不要——无监督 | 要（教师信号 $t$）——有监督 |
| 拿什么改权重 | 两路信号**同时活跃**（相关性） | 预测与答案的**误差** $(t-y)$ |
| 一句话形象 | “总在一起玩的，加连接” | “错了才改，改在参与出错处” |

请记住这位祖师爷：从本讲的感知器规则、下一节的 LMS，到第 2 讲的反向传播，再到第 4 讲的无监督学习，所有“调权重”的方法血脉里都流着赫布的思想——**活动留下连接，经验塑造网络**。

**零基础扫盲：把“学习率”想成油门**

$\eta$ 这个数为什么叫“学习率”？设想你在驾校练习倒车入库：方向盘打太猛（$\eta$ 太大），车在车位里荡来荡去停不稳；打太轻（$\eta$ 太小），挪半天也进不了库。学习率控制的就是每次修正的幅度：太大，训练在正确答案附近来回震荡；太小，训练慢得让人怀疑人生。选多合适？第 3 讲会用整整一节回答。现在只需记住：**学习率是你拧的第一个“训练旋钮”**。

## 1.5　从 0/1 到连续：线性单元与最小均方

阶跃函数有个先天限制：输出只有 0 和 1，只能回答“是 / 否”。可现实里大量问题要的是**数字**：明天气温多少度？这块板材的厚度是多少微米？而且 0/1 输出下，“差一点”和“错很多”没有区别——猜 0.49 判 0，猜 0.51 判 1，训练信号一样糙。改良方案简单到令人发笑：**把阶跃函数换成一条直线** $f(z)=z$（恒等函数），输出就是加权和本身，一个连续值。这样的神经元叫**线性单元**（linear unit）。1960 年前后，斯坦福的维德罗（Widrow）与霍夫（Hoff）做出了一颗叫 ADALINE 的自适应线性单元，用来自动滤除电话线路里的噪声——这是神经网络第一次在真实通信工程里上岗。

输出连续了，“错多少”也就能算了。定义**平方误差损失**（squared error loss）：

$$E = \frac{1}{2}(t-y)^2. \tag{1.6}$$

**人话翻译**：$E$ 衡量“这次错得多离谱”——预测与真值的差，平方，再乘 $\frac{1}{2}$。三个设计动机：**大错重罚**（差 3 罚 4.5，差 1 只罚 0.5，平方放大大错）；**处处可导、曲线光滑**（下一讲求导要用，阶跃函数除跳点外导数全为零，等于告诉训练“往哪都不坡”，没法用）；**正负误差不抵消**（差 $+2$ 和差 $-2$ 不会互相掩盖）。至于那个 $\frac{1}{2}$：纯粹为了平方求导后冒出来的 2 被约掉，公式干净些。

维德罗与霍夫证明，直接对 $E$ 求导并沿负梯度方向修参数，得到的更新规则与感知器规则**一模一样**（$w_i \leftarrow w_i + \eta\,(t-y)\,x_i$），但含义升级了：每一批样本**每一步都更新**，不再“错才动”。这套方法叫 **LMS 最小均方规则**（Least Mean Square）。请留意这件事的分量：上一节的规则是“错了才动”的经验智慧，这一节的规则是“沿坡往下走”的数学必然——同一个公式，两次发明，而后者脚下的

那条路（沿负梯度更新参数）正是第 2 讲梯度下降与反向传播的地基。顺便，你的词汇表里从此多了一员大将：凡是“衡量输出与正确答案差距的函数”，一律称**损失函数**（loss function），全书反复出现。

## 1.6　天花板与寒冬：一本书按下的暂停键

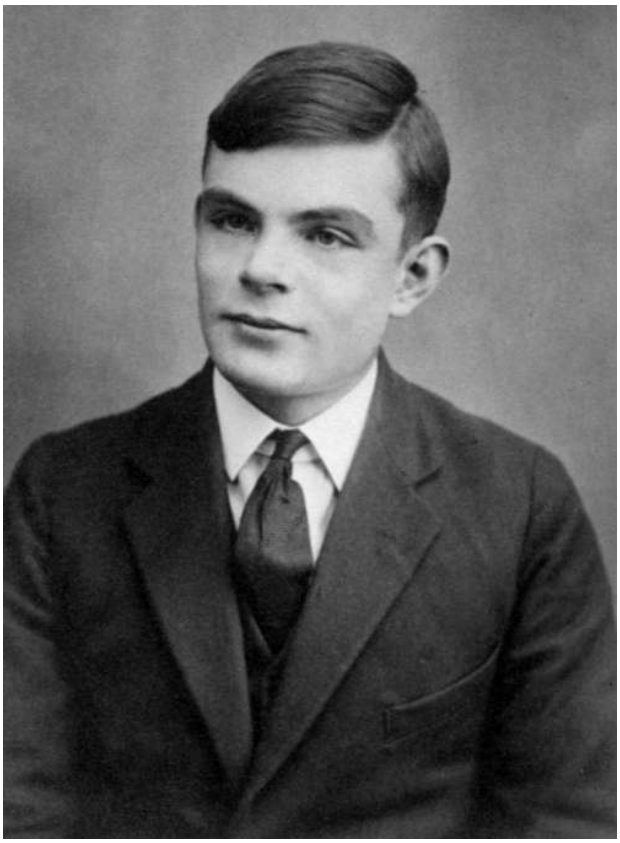


**图 1.7**　艾伦 • 图灵（1912–1954）：1950 年发表《计算机器与智能》，讨论“机器能否思考”并提出模仿游戏（图灵测试）

图片来源：维基百科 Turing 词条

感知器的报道之所以敢写“会自我复制、有自我意识”，是因为当时整个学界对“机器智能”期待爆棚——图灵 1950 年那篇《计算机器与智能》已经把“机器能思考吗”吵成了显学。期望被抬得越高，摔下来就越响。1969 年，MIT 的明斯基（Minsky）与佩珀特（Papert）出版了一本薄薄的书，书名就叫《Perceptrons》。书里没有谩骂，只有数学：他们严格证明，**单层感知器连异或（XOR）这样简单的问题都解决不了**——正是 1.3 节真值表里那个“分不开”。更扎心的还有一系列别的局限。书的结尾其实暗示：多层网络也许能绕过这些限制——但当时没人知道**怎么训练**多层网络：隐藏层接触不到标准答案，“错了才改”的规则对它失灵。

后果来得又快又狠。资助机构觉得这条路被“证伪”了，经费断崖式撤离；研究者的兴趣转向其他路线；神经网络研究进入长达十几年的冰期，史称**第一次 AI 寒冬**。罗森布拉特本人 1971 年在一次划船事故中去世，没能看到自己的感知器“复活”。回头看，这场寒冬令人唏嘘的地方在于：**解药其实早已写出**——1974 年，哈佛博士生韦伯斯（Werbos）在论文里给出了把误差责任从输出层倒推回每一层的方法；1986 年，鲁梅尔哈特（Rumelhart）、辛顿（Hinton）与威廉姆斯（Williams）让它广为人知。那套方法叫**反向传播**，是第 2 讲的主角。感知器死而复生的故事还留了一个教训，也是本

书反复念叨的道理：把牛皮吹破，买单的是整个领域——所以 **AI 可以用，结果自己扛**。

把镜头拉远，七十多年的恩怨情仇可以装进一张表。五个阶段——起步、高举、寒冬、复苏、爆发——注意它和你手里这本讲义的顺序几乎重合：**历史顺序就是学习顺序**，后面 11 讲就是把这张表逐行展开。

**表 1.2**　神经网络编年史：五个阶段一张表（教材第 1 章编年史的人话版）

| 阶段 | 年份 | 事件 | 一句话人话 |
|---|---|---|---|
| 起步 | 1943 | M-P 神经元模型 | 神经元第一次被写成数学：加权求和，过阈值才放电（1.1 节） |
| | 1949 | Hebb 学习规则 | “一起放电，连在一起”：学习规则的祖师爷（1.4 节） |
| | 1957 | 感知器 | 第一台能自己学习的机器，本讲主角 |
| | 1960 | Adaline | 线性单元加最小均方：神经网络第一次工程上岗（1.5 节） |
| 高举 | 1956 | 达特茅斯会议 | “人工智能”正式得名，全行业期望值拉满 |
| 寒冬 | 1969 | 《Perceptrons》 | XOR 按死单层感知器，经费撤离：第一次 AI 寒冬（上文） |
| 复苏 | 1974 / 1986 | 反向传播两次问世 | 韦伯斯先写出解药，鲁梅尔哈特与辛顿让它家喻户晓：多层网络终于能训了（第 2 讲） |
| | 1982 | Hopfield 网络 | 物理学家用“能量地形”讲联想记忆，神经网络回魂 |
| | 1989 | 卷积网络（CNN） | 杨立昆（Yann LeCun）把“看局部”装进网络，机器识别真正开始“看图”（第 5 讲） |
| | 1995 | SVM | 核方法如日中天，神经网络再坐冷板凳 |
| | 1998 | LeNet-5 | 卷积网络读邮编、认支票：在美国银行系统真刀真枪商用 |
| 爆发 | 2006 | Hinton 深度学习 | 逐层预训练破局，“深度学习”正式得名（第 4 讲） |
| | 2012 | AlexNet 横扫 ImageNet | GPU 加深网络，识别错误率一口气砍下约十个百分点，深度学习时代开幕（第 5–7 讲） |
| | 2016 | AlphaGo | 深度网络加强化学习赢下围棋，全民上了一堂 AI 公开课 |
| | 2022 | ChatGPT 与大模型 | 预训练加注意力：从一根人造神经元长成万亿参数（第 11 讲） |

**前沿速递：从 400 个旋钮到万亿参数——“权重即知识”**

把 Mark I 感知器与 2026 年的大模型放上天平：一边是 400 个马达旋钮，另一边是万亿级别的参数（第 11 讲细讲），多了约九个数量级。有趣的是，参数的本质六十八年**没有变过**：每一个依然是“重视程度”旋钮——大模型回答你一句话，就是几千亿个旋钮对文字片段做加权打分、层层接力。所谓“知识存在模型里”，更准确的说法是**知识存在参数的取值组合里**。2024–2026 年大模型的“军备竞赛”——参数从千亿冲向万亿、又反过来研究怎么用更少参数干更多活（第 8 讲的轻量化）——本质仍是感知器时代的三个老问题：**旋钮怎么布置、怎么调、调多少个才够用**。你在这一讲学会的“加权和 + 从错误中调权重”，就是这一切的原点。

**项目连线：第 1 讲 × 三个项目——人人都是感知器的后代**

别以为感知器只是历史课，它此刻就活在三个项目里。**六目拼接**：九个可学习“参数头”网络里有一个 policy（策略）头，它的输出本质是“这一帧要不要用这个特征、这条匹配边要不要信”的 0/1 决策——一颗放大了几十亿倍的感知器。**水下双目**：学生模型里“左右两幅图像块是否匹配”的打分头，干的正是“加权重合拍程度”的活儿，只是把 400 个像素换成了图像块特征（详见第 二 章）。**数据基座**：监考员模型判断“这条内容是真数据还是伪造的”，是教科书式的两类分类——与本讲课后题的鸢尾花分类同一副骨架（详见第 三 章）。它们的差别只在：激活函数换了、层数堆了、参数量爆了。层数怎么堆是第 2 讲的事，怎么调好是第 3 讲的事。

## 1.7　从这一讲到 2026

本讲结束时，你已经拆开了那台 1958 年的机器。把它的核心想法——加权和加阈值、从错误中调权重——放到 2026 年的现场，你会发现它不但没被淘汰，反而站在舞台正中央：2026 年的大模型，剥到最里层仍然是**线性层堆出千亿、万亿个权重，靠梯度下降学习**的感知器放大版。今天这门手艺向两个极端延伸。一端是**稀疏巨模型**：参数大到任何显卡都装不下，于是 MoE 架构（Mixture of Experts，混合专家——把大网络拆成许多“专家”小网络，每个问题只唤醒相关的少数几个）成了主流：总参数上不封顶，单次推理却只动一小部分。另一端是**端侧小模型**：把能看图、会调函数的多模态模型整个塞进手机。夹在中间的，则是一场**开源权重**的公共化运动——权重（回忆本讲：就是那批“重视程度”旋钮的全部取值）被各家以宽松许可公开发布，人人可下载、可研究、可商用，“神经元加权重”从实验室机密变成了全人类的公共资产。

**2026 现场：关键事实**

- DeepSeek-V3（2024 年 12 月发布）：MoE 大模型，总参数 671B，每个词元（token，大模型处理文本的最小单位）仅激活 37B——不足总量的十八分之一；完整训练耗资 278.8 万 H800 GPU 小时，训练过程以“异常稳定”著称（来源：DeepSeek-V3 Technical Report，arXiv 2024，编号 2412.19437）。
- 阿里 Qwen3（2025 年 4 月 29 日开源）：旗舰 MoE 模型总参数 235B、每词元激活 22B，以 Apache 2.0 许可完全开放权重，并提供“深度思考 / 快速回答”双模式（来源：Qwen 官方博客“Qwen3: Think Deeper, Act Faster”，2025）。
- Google Gemma 3n（2025 年 6 月）：Gemma 家族首个多模态端侧小模型，面向手机等资源受限设备，支持文本、图像输入与函数调用——大模型第一次“住进”口袋（来源：Google Developers Blog，2025）。
- 2026 年开源格局：DeepSeek、Qwen、Kimi、GLM 等中国团队与 Meta Llama、Google Gemma、Mistral 共同主导，新世代模型以“原生多模态加多尺寸档位”快速迭代（来源：Featherless.ai 年度开源模型盘点，2026）。

一句话收束：**你在本讲徒手算过的那颗 $\mathbf{w}\cdot\mathbf{x}+b$ 神经元，如今被复制了万亿倍，装进了机房和手机——感知器从未过时，它只是长成了大模型。**

本讲课后题要求你指挥 AI 写一个真感知器，在真实数据集上把它训练出来、画出它学到的那条决策直线。数据就是 1.3 节见过的鸢尾花——五要素如下。

**第 1 讲课后编程题（用 AI 实现）**

**题目**：指挥 AI 辅助实现一个感知器分类器，在 sklearn 自带的鸢尾花数据集上完成两类分类：只用**花瓣长度、花瓣宽度**两个特征，只取 `setosa` 与 `versicolor` 两类；训练后打印准确率，并用 matplotlib 把样本散点和你学到的决策边界画在一张图上；然后把数据换成另外两类（`versicolor` 与 `virginica`）再跑一遍，观察并解释结果为什么变了。

**交付物**：(1) 代码文件 `perceptron_iris.py`；(2) 两次运行的截图（各含准确率数字与决策边界图）；(3) 一段 50 字以上的说明，写清“我改了 AI 代码里的什么、为什么这么改”。

**提示词建议**（可直接抄）：“我是零基础 Python 学生。请用 numpy 手写一个感知器分类器（不要直接调 sklearn 的 Perceptron 类），使用 sklearn 自带的 load_iris 数据集，取花瓣长度、花瓣宽度两个特征，只保留 setosa 和 versicolor 两类。要求：学习率 0.1，训练 10 轮；训练后打印准确率；用 matplotlib 画散点图，并把决策边界直线叠画上去；每一行代码加中文注释；最后告诉我怎么运行这个文件。”

**验收线**：代码在你自己的电脑上能跑通；随机抽问时能不看书讲清楚 $w$、$b$、$\eta$ 各自的作用和“什么时候更新”；至少有一处自己的修改（换特征、换学习率、换类别都算）。**代码是 AI 写的没关系，讲不清楚就是没做。**

**加分项**：`versicolor` 与 `virginica` 在花瓣特征平面上并不线性可分——把阶跃激活换成线性单元加平方损失（1.5 节），预测值以 0.5 为界判类，看准确率会不会更好，并用两三句话解释原因。

**感知器（Perceptron）**

罗森布拉特 1958 年提出的人工神经元：输入加权求和加偏置，过阶跃函数输出 0 或 1。

**权重（weight）**

每路输入前的系数，表示对该输入的“重视程度”，可正可负，是可学习的主要参数。

**偏置（bias）**

加在加权和上的常数，相当于触发门槛的高低调节，也是可学习参数。

**参数（parameter）**

权重与偏置的统称；网络中所有可被训练调整的数。

**激活函数（activation function）**

加在加和结果上的非线性函数；本讲的阶跃是最古老的一种，第 2 讲会换更平滑的几种。

**决策边界（decision boundary）**

网络判类的“分界线”；二维输入时感知器的决策边界是一条直线。

**线性可分（linearly separable）**

两类样本能被一条直线（高维下是一个超平面）完全分开；感知器只对这类问题有理论保证。

**学习率（learning rate）**

每次参数修正的步长 $\eta$；训练中最先要拧的旋钮，第 3 讲专题讨论。

**损失函数（loss function）**

量化“输出与正确答案差多少”的函数；本讲的平方误差 $E = \frac{1}{2}(t-y)^2$ 是最经典的入门款。

**异或问题（XOR problem）**

异或运算四点无法用一条直线分开，单层感知器因此失效；正是它逼出了多层网络与反向传播。

# 第 2 讲　BP 神经网络、反向传播与梯度检查

上一讲结尾，感知器倒在了异或面前：四个点，一条直线怎么也分不开。战胜它的思路朴素得出奇——**一条线不够，就用两条**。把几个感知器“叠”起来：第一层各自画线，第二层对画线的结果再做判断，四点即可分开。感知器时代最大的死结就此解开了一半。为什么只算一半？因为新的麻烦马上到来：第一层神经元藏在网络内部，看不见标准答案，“错了才改”的学习规则对它**失灵**——第一层的权重到底该怎么改？这一讲的主角**反向传播**（Back Propagation，简称 BP），就是那套“把误差责任从输出层一层层倒推回去”的记账方法。它 1986 年成名，直到 2026 年的今天，全世界所有大模型的训练核心仍然是它。

**本讲地图**

**讲什么**：多层感知器（MLP）的结构与“自动造特征”直觉、四种激活函数、梯度下降、反向传播四步公式、以及训练代码的第一道安检——梯度检查。**为什么值得学**：反向传播是三个项目里一切模型训练的引擎；实验三（11月 23 日）你亲手跑的就是本讲的训练循环。**学完你能**：画出 2-2-1 网络并讲清误差怎么倒推；把四步公式翻译成人话；用数值梯度检查抓住手写 BP 的 bug；看懂 PyTorch 的 `loss.backward()` 背后发生了什么。

## 2.1　多层感知器：自动造特征的流水线

把感知器排成队，就得到**多层感知器**（Multi-Layer Perceptron，MLP）。最典型的结构分三种角色：**输入层**（输入层神经元不计算，只负责把数字摆上台面，比如花瓣的长宽）；一个或多个**隐藏层**（hidden layer，真正干活的中间层，外界看不见所以叫“隐藏”）；以及**输出层**（给出最终答案）。相邻两层之间两两全连接（fully connected）——前一层每个输出都连到后一层每个神经元，每个连接上有一个可学习的权重。下图是一个 4-5-2 的三层网络：4 个输入、5 个隐藏神经元、2 个输出，图中每一根箭头都是一个参数。

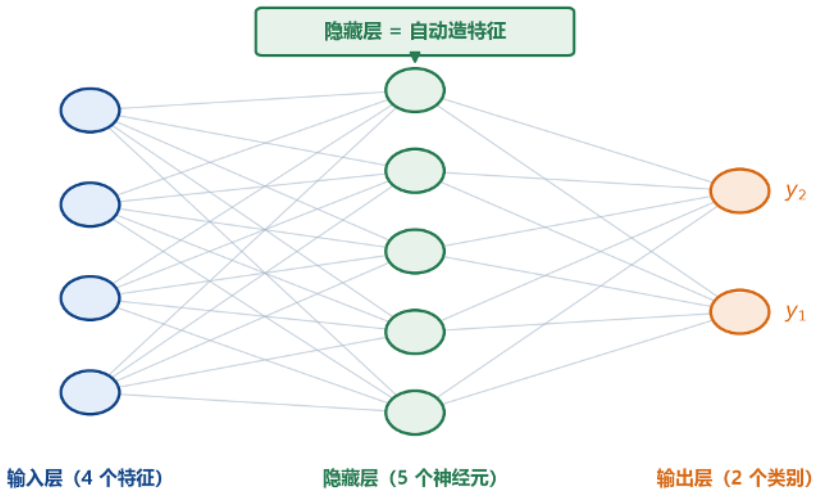



**图 2.1**　一个 4-5-2 的三层全连接网络：输入层只摆数，隐藏层负责“自动造特征”，输出层给出最终判断；每根箭头都是一个可学习的权重

隐藏层到底在干什么？一句人话：**自动造特征的车间**。原始特征（像素值、温度、花瓣长度）进去，加工出来的新特征出来——至于新特征长什么样，没人手工设计，是训练“逼”出来的：哪一种中间表示能让最后那层感知器把直线画开，网络就往哪种表示变形。1989年数学家证明过一条“通用近似定理”：带一个足够宽的隐藏层，MLP 原则上能逼近任意连续函数——“神经网络什么都能学”这句话的理论底气就在这里（当然，“原则上能”和“实际学得动”是两回事，那是第 3 讲的话题）。

现在回到异或。下图左边是那个死局：四个点卡在两条对角线上，一条直线无解。右边是两层网络的解法：第一个隐藏神经元画一条线，学出“至少一边是 1”（或）；第二个隐藏神经元画另一条线，学出“两边同时是 1”（与）；输出神经元对两个隐藏结果做一次“或且非与”——恰好就是异或。两条线分别单独都不行，**合作**起来就分开了。这就是“叠起来”的全部含义：**每一层单独看都是简单的感知器，叠起来才有复杂的本事**。

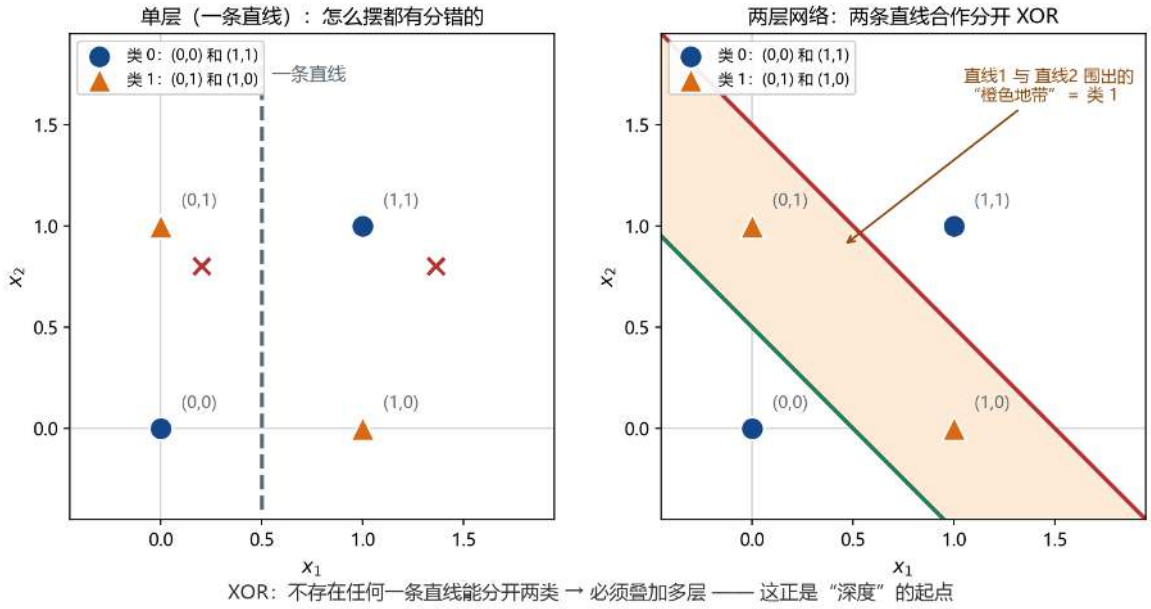



图 2.2　XOR 问题：左——四个点无法用一条直线分开；右——两个隐藏神经元各画一条直线，输出层把两条线的判断组合起来，即可分开四点

## 2.2　激活函数：没有它，叠一百层也白叠

且慢，先堵一个漏洞。如果每个神经元还是上一讲的阶跃函数，或者干脆什么函数都不加（输出等于加权和），会发生什么？代数一算就露馅：两层线性变换叠起来还是线性——

$$\mathbf{W}_2(\mathbf{W}_1\mathbf{x}) = (\mathbf{W}_2\mathbf{W}_1)\,\mathbf{x}, \tag{2.1}$$

两个矩阵乘完是一个矩阵，一百层等价于一层。所以**层与层之间必须夹一道“把直线掰弯”的非线性变换**，网络才有资格学复杂的东西。这道变换就是激活函数，它是每颗神经元的“性格”。五款最常用的如下（下图是其中前四款的曲线）：

- **sigmoid**：$\sigma(z) = 1/(1+\mathrm{e}^{-z})$，把任意数压进 $(0,1)$ 区间，像“概率化的开关”；如今多用于输出层给概率，隐藏层已少见。
- **tanh**：把任意数压进 $(-1,1)$，输出以 0 为中心，梯度表现通常比 sigmoid 顺滑，隐藏层的老将。
- **ReLU**：$\max(0,\, z)$，负数掐灭、正数放行；计算极快、正区不打弯，现代网络的默认首选。

- **LeakyReLU**：ReLU 的改良，负区不留死寂、只留一条斜率 0.01 的小缝。专治 **dying ReLU**（神经元“装死”）：若某个神经元的输入几乎总为负，ReLU 让它输出恒为 0、导数也恒为 0，权重从此收不到任何学习信号；那条小缝保证它总有被唤醒的机会。
- **Maxout**：可学习的激活函数——神经元自带一小片参数，每次取几条候选直线里最高的那个值（分段线性，但“折点”由训练自己学出来）。表达力更强，代价是参数翻倍，工程里不如 ReLU 普及。

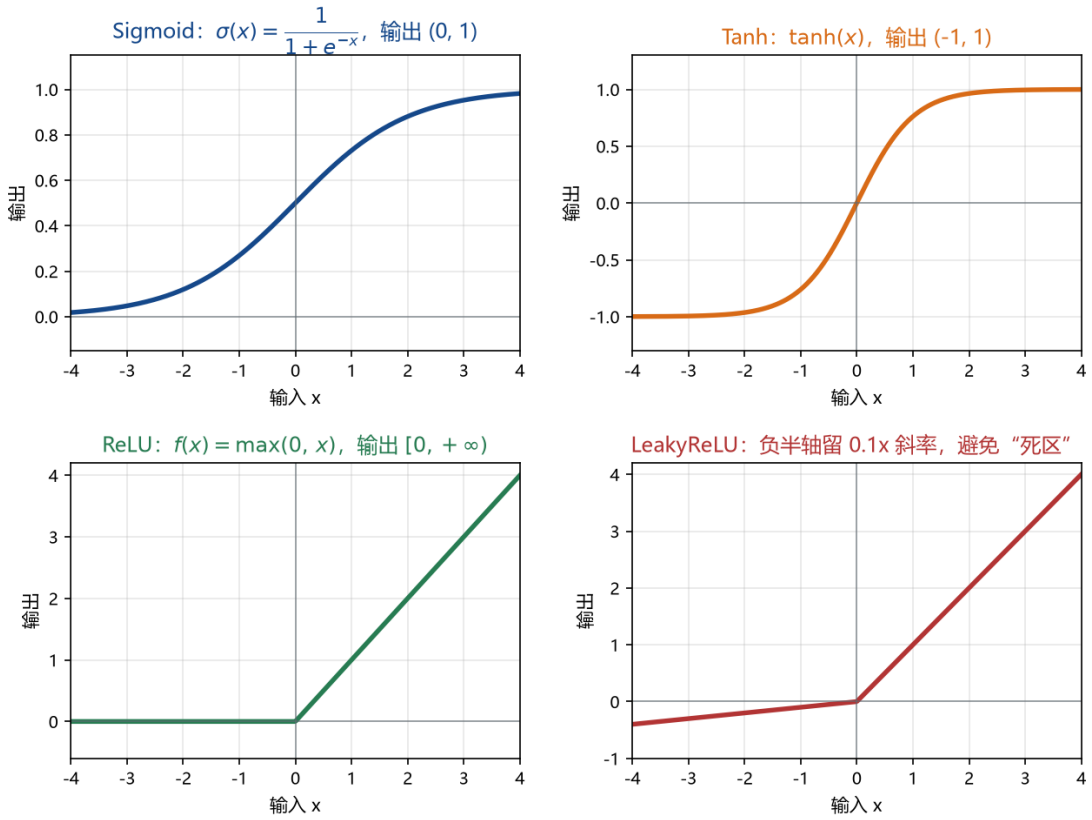



**图 2.3**　四种常用激活函数曲线（2×2 子图）：sigmoid 与 tanh 两端趋于水平（容易饱和），ReLU 与 LeakyReLU 正区斜率恒为 1

**表 2.1**　五款激活函数速查：一句人话与常用位置

| 函数 | 一句人话 | 常用位置 | 点评 |
|---|---|---|---|
| sigmoid | 压到 0–1 的“概率开关” | 输出层（二分类概率） | 两端易饱和，隐藏层已让位 |
| tanh | 压到 −1 到 1，零中心 | 隐藏层（经典网络） | 比 sigmoid 收敛更顺 |
| ReLU | 负数掐灭，正数放行 | 隐藏层（现代默认） | 快、稳，缺点是可能“神经元死亡” |
| LeakyReLU | 负区留 0.01 小缝 | 隐藏层（ReLU 替代） | 专治“死透”的神经元 |
| Maxout | 取几条候选里的最大者 | 隐藏层（不常见） | 可学习的激活函数：折点由训练决定，参数翻倍 |

**零基础扫盲：饱和区——学不动的“平原”**

看上图 sigmoid 和 tanh 的两端：曲线几乎水平。数学上，**越平的地方导数越接近 0**。导数是什么？回忆开课导引数学包（第 0 讲）：导数是“变化率”，在训练里它就是“学习信号的大小”——导数为 0，意味着无论误差多大，这个神经元都“纹丝不动”：**饱和**（saturation）。埋个伏

笔：下一节反向传播会把误差信号乘着导数一层层往回传，路上经过几段饱和区，信号就被乘得所剩无几——这就是让人闻风丧胆的“梯度消失”的根源之一，也是 ReLU 能上位的理由（它的正区斜率恒为 1，信号路过不打折）。第 3 讲训练调参时还会和它打交道。

**Softmax 与交叉熵：分类的黄金搭档**

感知器只回答“是 / 否”，现实里更多问题是**多选一**：这张图是猫、狗还是鸟？两件套专门伺候这类问题。**Softmax** 站在输出层，把一排打分变成一排**概率**：

$$y_i = \frac{\mathrm{e}^{z_i}}{\sum_j \mathrm{e}^{z_j}}, \tag{2.2}$$

人话：谁的分数高谁的概率大，且所有概率加起来恰好等于 *1*——“票数”从此变成“得票率”。**交叉熵损失**（cross-entropy）则站在损失的位置，衡量**预测的概率分布与真值差多远**：

$$L = -\sum_i t_i \log y_i, \tag{2.3}$$

人话：真值 $t$ 是一张只在正确类别处为 *1* 的选票；正确类的得票率越低，罚得越狠——预测满分（$y=1$）罚 *0*，接近零票则罚出天文数字。两件套在现代框架里常被熔成一个损失使用，熔合后的梯度形式简洁得漂亮：

$$\frac{\partial L}{\partial z_i} = y_i - t_i, \tag{2.4}$$

恰好等于“预测概率减真实标签”。推导这里不给（框架也根本不需要你推），但这个结论值得记：误差信号就是“你猜的”减“标准答案”，朴素得像设计好的。第 3 讲的课后题就用这套黄金搭档。

## 2.3　梯度下降：在山地里找最低谷

现在假设网络结构定了、激活函数选了，还剩最后一个问题——参数怎么调？先给“调得好不好”定个尺子：**损失函数**（第 1 讲已登场，比如平方误差）。所有参数的一组取值，对应一个损失值；想象把“每一组参数取值”都摆成平面上的一个点，把损失值当海拔，参数空间就成了一片**山地**：山谷是损失低的好参数，山头是损失高的坏参数。**训练的目标，就是找到最低谷**。

怎么下山？微积分给了一个笨但可靠的答案：站在任何一点，**梯度**（gradient）指向**最陡的上坡**方向——梯度就是“全部偏导数组成的向量”，每个偏导数回答“这一维参数动一丁点，损失变多少”。那么最陡的下山方向就是它的反方向。写成公式：

$$w \;\leftarrow\; w - \eta \frac{\partial L}{\partial w}. \tag{2.5}$$

**人话翻译**：把每个参数 $w$ 朝“损失变小的方向”挪一小步——$\partial L/\partial w$（读“$L$ 对 $w$ 的偏导数”）是这一点上 $w$ 方向的坡度，前面加负号就是下山；$\eta$ 还是老朋友**学习率**，步子迈多大。如此反复，“沿坡下山、走到谷底”，就叫**梯度下降**（gradient descent）。

但请注意这片山地的规模：一个现代模型的参数以百万、亿、万亿计——这是**亿维**的山地，每走一步都要算出**每一个维度的坡度**。靠谁来算？手算一两个参数还行，十亿个不可能。下一节的反向传播，就是那个把这件事变可能的算法。另外先预告一张图：不同学习率下山会走出截然不同的路径（震荡、平稳、龟速），损失曲面的立体长什么样，第 3 讲有专门的等高线图等着你。

## 2.4　反向传播：把责任一层层倒推回去

来看这个场景：网络给出预测后，输出层“知道自己错多少”——预测值和标准答案的差明摆着。可隐藏层呢？它没有标准答案，上一讲“错了才改”的规则无从谈起。**每个权重该背多大的锅？**这就是反向传播回答的问题。

思想可以用一句话讲完：**责任按贡献分配，从右往左传**。输出层的误差是一笔“总账”，往回分账时，谁对这笔误差**影响大**（谁权重大、参与多），谁就多背；每个神经元收到分来的账，再往它上游分……直到每个参数都拿到自己的那份责任，按责任大小修自己。下图里蓝色是数据从左往右的前向流动，红色是责任从右往左的倒推流动——一去一回，正好构成一次完整的训练呼吸。

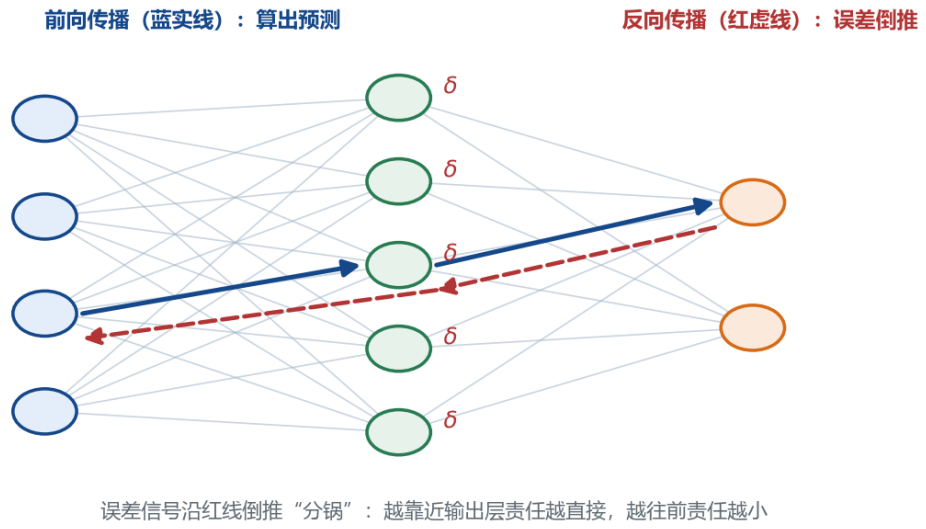



**图 2.4**　反向传播示意：蓝色箭头为前向计算流（输入 → 输出），红色箭头为误差倒推流（输出 → 输入）；每个权重按贡献分得自己的“责任”$\delta$

把“分账”变成数学的钥匙，是开课导引 0.5 节埋好的——链式法则。

**知识补齐：链式法则——反向传播的全部数学（回指第　讲）**

若 $y$ 随 $u$ 变、$u$ 随 $x$ 变，则总变化率相乘：

$$\frac{\mathrm{d}y}{\mathrm{d}x} = \frac{\mathrm{d}y}{\mathrm{d}u} \cdot \frac{\mathrm{d}u}{\mathrm{d}x}. \tag{2.6}$$

一个多层网络，本质是一条很长的复合函数链：输入变成第一层输出，第一层输出变成第二层输入……最后变成损失。要问“某个早层参数动一丁点，损失动多少”，就把沿途每一段的变化率**乘起来**。反向传播没有任何新数学——它只是链式法则的工程化：**沿链条从右往左，把已经算好的导数逐段乘回来**，中间结果全程复用，绝不重复计算。

下面是反向传播的四步公式。设定：共 $L$ 层，第 $l$ 层的加权和记 $\mathbf{z}^{(l)}$，激活后的输出记 $\mathbf{a}^{(l)}$；$\mathbf{W}^{(l)}$、$\mathbf{b}^{(l)}$ 是该层参数；$\boldsymbol{\delta}^{(l)}$（粗体德尔塔 delta）是第 $l$ 层每个神经元的“责任清单”；$\odot$ 表示“对应位置相乘”。每条公式下面都配了人话。

**第一步（前向）**：信号从左到右算一遍，途中存下每层结果：

$$\mathbf{a}^{(l)} = \sigma\big(\mathbf{W}^{(l)}\mathbf{a}^{(l-1)} + \mathbf{b}^{(l)}\big). \tag{2.7}$$

人话：输入逐层加工到输出，顺手把每层的“中间产物”记在账本上——回传时要用。

**第二步（输出层责任）：**

$$\boldsymbol{\delta}^{(L)} = \nabla_{\mathbf{a}^{(L)}} L \;\odot\; \sigma'\big(\mathbf{z}^{(L)}\big). \tag{2.8}$$

人话：答案错多少（损失对输出的导数），乘上激活函数在该点的斜率——斜率是 *0* 的神经元（回忆饱和区）暂时学不动，这是原文照抄的物理规律，不是 *bug*。

**第三步（隐藏层责任）：**

$$\boldsymbol{\delta}^{(l)} = \big(\mathbf{W}^{(l+1)}\big)^{\mathsf{T}} \boldsymbol{\delta}^{(l+1)} \;\odot\; \sigma'\big(\mathbf{z}^{(l)}\big). \tag{2.9}$$

人话：下游的责任，按权重比例“退货”给上游——转置 $\mathbf{W}^{\mathsf{T}}$ 就是“按贡献反着分账”；再乘上本神经元自己的斜率。一层层往左推，直到第一隐层。

**第四步（更新）：**

$$\mathbf{W}^{(l)} \leftarrow \mathbf{W}^{(l)} - \eta\, \boldsymbol{\delta}^{(l)} \big(\mathbf{a}^{(l-1)}\big)^{\mathsf{T}}, \qquad \mathbf{b}^{(l)} \leftarrow \mathbf{b}^{(l)} - \eta\, \boldsymbol{\delta}^{(l)}. \tag{2.10}$$

人话：谁参与犯错谁改——责任大（$\delta$ 大）、参与多（输入 $\mathbf{a}$ 大）的权重改得多；方向与梯度相反（负号），步子由 $\eta$ 定。

至此可以回答本讲开头的问题了：隐藏层看不见标准答案怎么办？**不用看见——它的“答案”就是下游退回来的责任**。再品味一次这件事的效率：一次反向扫描，就把**全部**参数的梯度都算了出来，计算量大约只有两次前向传播。“一次错误，全体对账”，如此便宜，这就是反向传播统治深度学习六十年的原因。

## 2.5　梯度检查：给你的反向传播“对账”

坏消息：上面第三步里那个转置，是手写 BP 代码时最容易写错的地方之一——漏转置、转错轴、导数写错号，程序**不报错**，只是“学得特别慢”或者“根本学不动”。这是深度学习里最难抓的一类 bug：不崩溃，只摆烂。好消息：有一个笨办法能当场抓住它。

办法来自导数的定义本身。想知道 $w$ 处的坡度？把 $w$ 往两边各挪一丁点 $\varepsilon$（读“艾普西龙”，取 $10^{-5}$ 这个量级），量一量海拔变化：

$$\frac{\partial L}{\partial w} \;\approx\; \frac{L(w+\varepsilon) - L(w-\varepsilon)}{2\varepsilon}. \tag{2.11}$$

这叫**数值梯度**（numerical gradient）：下图里，两端点连线的割线斜率，在该点附近就是切线斜率的好替身。它又慢（每查一个参数要跑两遍前向）又准——正好和你手推的**解析梯度**（analytic gradient，即上一节那些公式算出来的）互补：**训练前抽查一批参数，两相对账**，相对误差在 $10^{-6}$ 量级以下即通过。几秒钟的对账，换整个训练的心安——**写训练代码，第一件事永远是梯度检查**，这个习惯从本讲的课后题开始养成，实验三还要用。

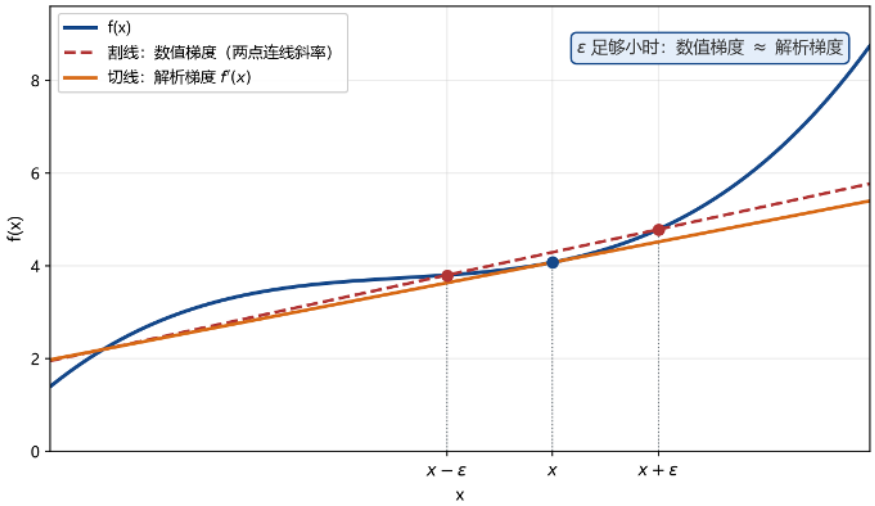



**图 2.5**　梯度检查的几何直觉：函数曲线上两端点连线的割线斜率（数值梯度）≈ 该点切线斜率（解析梯度），$\varepsilon$ 越小越接近

先在纸上找一次感觉，再看代码。

**动手时间：用手机计算器做一次梯度检查**

取函数 $f(x) = x^2$，在 $x = 3$ 处。**解析梯度**（求导）：$f'(x) = 2x$，$f'(3) = 6$。**数值梯度**（定义）：取 $\varepsilon = 0.1$，$f(3.1) = 9.61$，$f(2.9) = 8.41$，于是

$$\frac{9.61 - 8.41}{2 \times 0.1} = 6.0.$$

两相对账，完全一致。再把 $\varepsilon$ 换成 0.01 算一遍，看是不是更贴近 6——**割线在逼近切线**。整个过程只要一部手机计算器，你刚刚手工执行了一次“梯度检查”。

进阶版：把你手推的 BP 公式和数值定义各写一段代码，对每个参数比较（`w0` 取几个代表值）：

```
def num_grad(f, w, eps=1e-5):    # 数值梯度：定义式的笨办法
    return (f(w + eps) - f(w - eps)) / (2 * eps)

def ana_grad(w):                 # 解析梯度：你手推的公式 f'(x) = 2x
    return 2 * w

for w0 in [-1.0, 0.5, 3.0]:      # 抽查三个点
    print(w0, num_grad(lambda w: w**2, w0), ana_grad(w0))
# 输出两列几乎相同的数 => 你的"手推公式"对账通过
```

### 2.5.1　三种求导范式：从手推公式到 autograd

刚才的梯度检查，其实亮出了求导三大流派中的一派。三派同台看一眼，你就明白自己身在何处、框架在替你干什么：

**表 2.2**　三种求导范式：怎么算、好在哪、命运如何

| 范式 | 怎么算 | 优点 | 命运 |
|---|---|---|---|
| 数值微分 | 把参数挪一丁点、量海拔变化：只挪一边叫前向差分，两边都挪取平均叫中心差分——2.5节的梯度检查就是它 | 绝对可信，永远不错 | 太慢：每查一个参数要跑两遍前向，只配当“对账员” |
| 手动微分 | 人推公式，把导数一行行写死进代码 | 快 | 易错：模型改一处、导数全部重推——早期神经网络库与 OpenCV 时代的常态 |
| 自动微分 | 把程序拆成加减乘除等基本算子，每个算子的导数已知，链式法则自动把全程串起来 | 又快又准又通用 | 现代框架标配 |

一句话记住第三行：**PyTorch 的 autograd 就是自动微分——你写前向，它送你反向**。至于这件事后来被工程界推到了什么程度，本讲章末的前沿盒子接着说。

## 2.6　完整训练一次：把循环跑起来

万事俱备。把前向、损失、反向、更新串成一个循环，就是全部深度学习训练的骨架。下面这段 10 行伪代码（numpy 语法，`@` 表示矩阵乘法）在 2-2-1 的 MLP 上拟合异或，正是课后题的骨架：

```
W1, b1, W2, b2 = init()                 # 1 初始化参数（怎么初始化见第3讲）
for step in range(20000):               # 2 训练循环开始
    A1 = tanh(X @ W1 + b1)              # 3 前向：算隐藏层（tanh）
    Yhat = A1 @ W2 + b2                 # 4 前向：算输出（线性）
    L = mean((Yhat - Y)**2)             # 5 损失：这一轮错多少
    # —— 训练前先做梯度检查（2.5 节），通过后再开跑 ——
    D2 = 2 * (Yhat - Y) / len(X)        # 6 反向：输出层责任（四步之第2步）
    D1 = (D2 @ W2.T) * (1 - A1**2)      # 7 反向：隐层责任（转置退货+tanh导数1-A1^2）
    W2 -= lr * A1.T @ D2                # 8 更新：谁参与犯错谁改（四步之第4步）
    W1 -= lr * X.T @ D1                 # 9 同上
    # 每 1000 步打印一次 L，盯着它下降
```

逐段对照：第 3–5 行是四步公式的第一步（前向）加损失；第 6–7 行是第二、三步（算责任，注意第 7 行里 `W2.T` 的转置退货和 `1 -A1**2`——tanh 的导数正好是 $1-\tanh^2$）；第 8–9 行是第四步（更新）。所谓“训练”，**不是人去拧几十亿个旋钮，而是误差沿着网络把每个旋钮该转多少自动算出来**——机器自己执行的、规模化的第 1 讲手工训练。等第 6 讲你见到 PyTorch 版的训练循环，会发现换的只是“中间三行”变成 `loss.backward()`，骨架一毫米没动。

**BP 七步训练流程（教材经典流程）**

把本讲的零件按顺序装起来，就是教材里那条经典的七步流程——它同时也是你今后任何训练代码的目录：

1. **初始化**参数（别用全 0——为什么，第 3 讲揭晓）；
2. **样本输入**：取一批训练样本喂进网络；
3. **前向**逐层计算到输出，对照答案算出误差；

4. **反向**传播：按四步公式逐层算出每个参数的误差信号（责任清单 $\boldsymbol{\delta}$）；
5. **更新权重**：沿负梯度方向挪一小步；
6. **循环**第 2–5 步，直到损失不再下降（收敛）；
7. 拿**测试集**检查精度——成绩合格，才算训成。

对照 2.6 节的十行代码：第 1 行是第 1 步，第 3–5 行是第 2、3 步，第 6–7 行是第 4 步，第 8–9 行是第 5 步，外层 for 循环是第 6 步，课后题最后打印的准确率是第 7 步——七步你其实已经全部见过了。

**动手时间：在浏览器里“玩”一个神经网络**

理解七步流程最好的方式，是亲眼看它转起来。打开 `playground.tensorflow.org`（TensorFlow Playground，谷歌出品的神经网络可视化沙盘，浏览器直接用、零安装）：左边一排是超参数旋钮（学习率、激活函数、层数、每层神经元数），右上角点“运行”，就能**亲眼看着**一个小网络在平面上把两色数据点分开——决策边界像水波一样慢慢变形，那就是梯度下降本尊。建议玩三局：(1) 选螺旋（spiral）数据集、只留 1 个隐藏神经元，看它怎么都分不开（重温 XOR 的死局）；(2) 加到 2 层 8 个神经元再试，看边界扭出花来；(3) 把激活函数从 tanh 换成 ReLU，对比收敛快慢。十分钟的沙盘，胜过十页文字。

**胜者为王（WTA）竞争学习**

前面所有训练都靠标准答案纠错，其实还有一条不用答案的路。**竞争学习**让输出神经元互相竞争：每来一个样本，大家各自用“与它的合拍程度”（回忆第 1 讲的点积打分）投标，**只让得分最高的赢家激活，其余全部沉默**——这就是**胜者为王**（Winner-Take-All，WTA）。赢家独享这次更新：把自己的权重往样本方向挪一点。一来二去，每个赢家的权向量停在数据某一片的“正中央”，**自动收敛为聚类中心**——一个标签不用，网络自己把数据分了堆。这是**自组织学习的入门**，第 4 讲的无监督自编码器，是这条路上的更远一站。

**前沿速递：自动微分——你写前向，反向白送**

本讲花最大力气讲的反向传播，在 2026 年的日常研发里几乎没人手写——不是因为它不重要，恰恰因为它太重要，被“焊”进了框架。PyTorch、JAX、MindSpore 等主流框架都内置**自动微分**（automatic differentiation，俗称 autograd）：你只写前向计算，框架沿计算图自动、精确地求出每个参数的导数——本质就是把你的 $\delta$ 倒推过程自动化到任意复杂的网络上，连梯度检查都替你留好了接口。2016 年前后它让深度学习研究从“推导一个网络三个月”变成“搭一个网络一下午”；2024–2026 年，大模型动辄千亿参数，手写反向早已物理上不可能，自动微分加分布式训练就是大模型时代的“纸和笔”。但框架会算不等于你不用懂：本讲的梯度检查思想至今仍是排查训练问题的第一件兵器——第 3 讲你会亲手用它抓住“学习率调错”与“初始化不当”两个现行犯，实验四（12 月 21 日）还有它的实战任务。

**项目连线：反向传播——三个项目的共同引擎**

本讲的训练循环，就是三个项目里所有“可学习”二字的底气。**六目拼接**：九个参数头的每一次微调，都是把拼接误差（错位的像素）经 BP 倒推到每个卷积核上；端到端路线在 RTX 4080 上烧掉的每一度电，都在算 $\delta$（详见第 一 章）。**水下双目**：教师模型 FoundationStereo 参数全部冻结（不训练），学生模型的上亿参数，全靠反向传播从“与教师伪标签的差距”里一步步学出来——第二篇第 二 章那个“教师–过滤–学生”闭环的下半环，转的就是这个循环。**数据基座**：监考员真假分类器与 R-GCN 图检索模型，训练全是本讲的循环在转（详见第 三 章）。落到日程上：**实验三（11 月 23 日）你将亲手在真实数据上跑通这个循环；实验四（12 月 21 日）学把它跑得又快又稳**——那是第 3 讲的内容。

## 2.7　从这一讲到 2026

本讲的“前向—损失—反向—更新”，2026 年每天在全世界数以万计的显卡上运转，但它早已不是你手里的十行代码，形态发生了几重进化。第一重：**被编译**。PyTorch 2.5 起把整个反向传播捕获成一张计算图、交给编译器统一优化（Compiled Autograd）——反向传播从“运行时逐步递归求导”，变成“编译器手里可供全盘优化的图”。第二重：**被分布式**。模型大到一台机器装不下：中间激活存不下，就上“梯度检查点”（gradient checkpointing——注意别与本讲的梯度检查混淆：那是验算导数对不对，这是先扔掉中间结果、反向时再重算，拿时间换显存）；多卡之间梯度还要互相同步，DeepSeek-V3 干脆用 FP8（用更短的数字格式存数与运算，省显存省带宽）加双向流水线 DualPipe（前向与反向像双向车道交错通行），把反向传播的通信开销整个藏进计算里。第三重是悬念：**BP 会被取代吗**？辛顿本人提出的 Forward-Forward 方案（两次前向、不用反向梯度）2025 年已能训练卷积网络，是最有生物学味道的挑战者——但在大模型规模上仍全面落后。**反向传播活得很好，只是你越来越看不见它。**

**2026 现场：关键事实**

- PyTorch 2.5（2024 年 10 月）引入原型特性 Compiled Autograd：“捕获整个反向传播过程”为单一计算图交给 torch.compile 编译，且不受前向图断裂影响——反向传播正被纳入编译器流水线（来源：PyTorch v2.5.0 Release Notes，GitHub，2024）。
- 万亿参数训练的显存瓶颈靠“重算换显存”：梯度检查点 / 激活重算是标配；2025 年 8 月提出的 Adacc 框架把激活压缩与重计算统一调度，吞吐量较现有框架提升 1.01–1.37 倍（来源：Adacc，arXiv 2025，编号 2508.00806）。
- DeepSeek-V3 训练 671B MoE 模型时，核心矩阵运算用 FP8 精度，并以 DualPipe 双向流水线使反向传播中的 all-to-all 通信与另一路计算完全重叠，全部训练仅 278.8 万 H800 GPU 小时（来源：DeepSeek-V3 Technical Report，arXiv 2024，编号 2412.19437）。
- Forward-Forward（两次前向、无反向梯度）2025 年已扩展到卷积网络并正式刊出，被视为有生物学依据的 BP 替代路线；学界共识是其规模与性能尚不能取代 BP（来源：Scodellaro et al.，Training convolutional neural networks with the Forward-Forward algorithm，Scientific Reports，2025）。

一句话收束：**反向传播四十年没被推翻，但它正在被“编译”和“分布式化”——未来稀缺的不是会不会推链式法则，而是能不能在万卡集群上让梯度又快又省地流回去。**

本讲课后题不用任何框架，用 numpy 把上面的循环亲手跑通——这是全书唯一一次要求你手写反向传播，目的正是让你以后调用 `loss.backward()` 时心里有底。五要素如下。

**第 2 讲课后编程题（用 AI 实现）**

**题目**：指挥 AI 辅助，用 numpy 手写一个 2-2-1 的 MLP（2 输入、2 个隐藏神经元、tanh 激活、1 个线性输出）拟合 XOR 四个样本：$(0,0)\to 0$、$(0,1)\to 1$、$(1,0)\to 1$、$(1,1)\to 0$。要求顺序：先实现数值梯度检查并打印相对误差，通过后（$<10^{-5}$）才允许开训；用平方损失加全批量梯度下降，学习率 0.1，最多 2 万轮，训练到均方误差低于 0.01；用 matplotlib 画出“训练轮数 vs 损失”曲线并保存为 png；最后打印四个输入的预测值。

**交付物**：(1) 代码文件 `xor_mlp.py`；(2) 三张截图：梯度检查输出、损失下降曲线、四个输入的预测值；(3) 一段 50 字以上的说明，写清“我改了 AI 代码里的什么、为什么这么改”。

**提示词建议**（可直接抄）：“我是零基础学生，正在学反向传播。请用 numpy 实现 2-2-1 全连接网络：2 输入、2 个隐藏神经元（tanh 激活）、1 个线性输出，拟合 XOR 四个样本。要求：(1) 先实现数值梯度检查，打印每个参数解析梯度与数值梯度的相对误差；(2) 用均方误差损失和全批量梯度下降，学习率 0.1，最多 20000 轮，损失低于 0.01 就停；(3) 训练完打印四个输入的预测值；(4) 用 matplotlib 画损失随轮数下降的曲线并保存为 png；(5) 每行加中文注释，并注明哪几行对应‘前向、输出层 $\delta$、隐层 $\delta$、更新’四步。”

**验收线**：梯度检查通过且截图在；损失曲线单调下降并收敛到 0.01 以下；四个预测值与真值之差全部小于 0.1；抽问时能讲清 $\delta$ 如何从输出层倒推回隐藏层、$\mathbf{W}^{\mathsf{T}}$ 为什么会出现在隐层责任里；至少有一处自己的修改（换激活函数、换学习率、加一个隐藏神经元都算）。**代码是 AI 写的没关系，讲不清楚就是没做。**

**加分项**（任选其一）：(1) 把输出改成 sigmoid 激活加交叉熵损失（先问 AI“交叉熵是什么、为什么配 sigmoid”），对比收敛速度；(2) 把两个隐藏神经元学到的两条“直线”画在 XOR 四点散点图上，看看网络自己选了哪两条线；(3) 换三种参数初始化（全 0 / 太大 / Xavier）各训一次并对比曲线——第 3 讲的预告片。

**多层感知器（MLP）**

感知器排层叠成的全连接网络：输入层、一个或多个隐藏层、输出层。

**隐藏层（hidden layer）**

夹在输入与输出之间的计算层，负责“自动造特征”；其参数没有直接的标准答案，靠反向传播获得学习信号。

**全连接（fully connected）**

相邻两层神经元两两相连，每个连接带一个可学习权重。

**激活函数（activation function）**

层间必夹的非线性变换，如 sigmoid、tanh、ReLU；没有它，多层网络塌缩成一层。

**饱和（saturation）**

激活函数进入两端近水平区，导数接近 0，训练信号被“掐灭”的现象。

**梯度（gradient）**

损失对全部参数的偏导数组成的向量，指向最陡上坡方向；它的反方向就是下山方向。

**梯度下降（gradient descent）**

沿负梯度方向反复更新参数、走向损失最低点的迭代方法。

**链式法则（chain rule）**

复合函数的导数等于各段导数相乘；反向传播的全部数学。

**反向传播（Back Propagation, BP）**

利用链式法则把误差责任从输出层倒推回各层、一次算出全部参数梯度的算法，1986 年成名，至今仍是一切模型训练的引擎。

**梯度检查（gradient check）**

用数值定义（两点割线斜率）与手推解析梯度对账，抓手写 BP 的 bug；写训练代码的第一道安检。

# 第 3 讲　超参数、权重初始化、模型训练与评估

上一讲结尾，十行伪代码把训练循环跑了起来：前向、损失、反向、更新，一圈一圈地转。但只要你真去跑它，很快会撞见一件怪事：循环一个字没改，只把开头的几行“设置”换掉——学习率从 0.1 改成 10、权重全部从 0 起步、每批样本数从 100 改成 1——结果可能从“漂亮收敛”直接变成“数字爆炸成 nan”或“损失纹丝不动”。这些必须在开跑前拧好的旋钮，统称**超参数**（hyperparameter）——“参数之上的参数”：普通参数（权重、偏置）由反向传播自动学，超参数由你拧。调它们是一门手艺，而且像学做菜：**学习率是火候**，大火烧糊、小火夹生；**权重初始化是下料**，第一步就歪，后面步步歪；**验证集是尝味道**，菜好不好要边做边尝，不能等端上桌才发现咸了。本讲把这套“火候学”一次讲透：损失面长什么样、学习率怎么选、权重从哪开始、batch 三兄弟怎么分、数据怎么切、优化器家族谁是谁。第 2 讲埋的伏笔——用梯度检查抓住“学习率调错”与“初始化不当”两个现行犯、那张损失面等高线图——也都在这里兑现。

这一讲还是实验三（11 月 23 日，深度网络的模型训练实践）的直接理论装备：那天你敲下的每一条训练命令、盯着的每一条 loss 曲线，用的全是本讲的词汇。带着这个目的读，每一节都不抽象。

**本讲地图**

**讲什么**：损失面与梯度下降的“地形学”、学习率与衰减策略、权重初始化（Xavier/He）、batch/epoch/iteration三兄弟、训练—验证—测试三分与过拟合诊断、向量化编程、优化器家族（SGD、Momentum、RMSProp、Adam）。**为什么值得学**：超参数是三个项目里一切训练动作的把手；实验三（11 月 23 日）是本讲的实弹演习，实验四（12 月 21 日）要学“跑得又快又稳”，靠的也是这些旋钮。**学完你能**：读 loss 曲线诊断训练病症；给网络选对初始化；说清 batch size 的权衡；用 Adam 开训、用早停收工；看懂 `--epochs 200 --batch 8` 这类命令里每个词的含义。

## 3.1　损失面与梯度下降：先看清脚下是什么地形

第 2 讲说过，训练就是在参数空间这片“山地”里找最低谷。当时只给了一句话，现在把地图摊开。把一个只有两个参数的小模型的损失值画成海拔，就得到下面的等高线地形图：同一圈上海拔（损失）相同，中心是谷底。三条彩色路径是三次训练——起点相同、山是同一座，唯一区别是每一步迈多大。

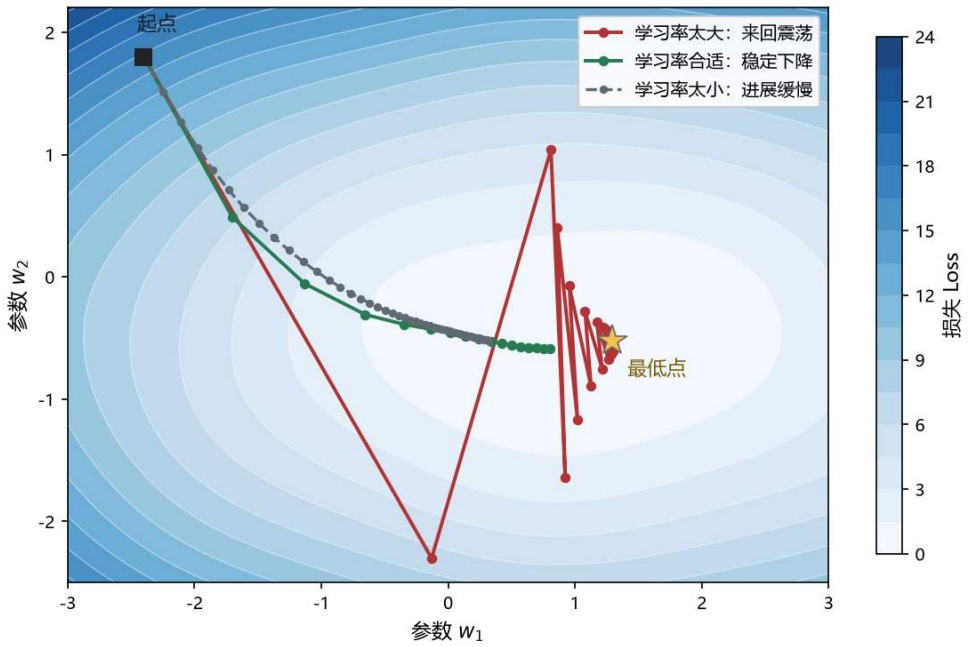



**图 3.1**　损失面等高线与三条下降路径：步长合适（稳步滑向谷底）、步长太大（在两壁间来回反弹甚至越弹越高）、步长太小（走到半路还没到）

读图要点：一条路径平滑转向、直奔谷底——步长合适；一条在谷底两侧来回横跳，甚至越跳越远离谷底——步长太大，每一步都跨过了头；还有一条方向对、步子碎，走出很远还在山腰——步长太小。三条路径的命运从第一步起就注定了，而决定步长的那个数，正是下一节的主角。

这片山地还有一个绕不开的地形问题：谷底不止一个。损失面上有**局部极小**（local minimum，四周都比它高的小坑），也有**鞍点**（saddle point，这个方向看是谷、换个方向看是山脊的“马鞍口”）。20 世纪 90 年代大家最怕局部极小——怕小球掉进半山腰的小坑再也出不来。但 2026 年的共识恰恰相反：现代模型的参数动辄百万上亿，在如此高维的山地里，**某一点要在所有方向上都恰好是坑底，概率极低；真正拦路的，是大量的鞍点和平坦高原**。这带来一个实用的副产物：训练“卡住不动”时，往往不是掉进坑里，而是正走在平地上——多等几步、换一批数据（加大批量内的噪声）或微调学习率，小球常常自己继续往前滚。下次你的曲线停住，先别绝望，也别乱调，等一等再看。

## 3.2　学习率：整张清单上最重要的一个数

学习率（learning rate）$\eta$ 已经出场过两次：第 1 讲感知器挪线的步长、第 2 讲梯度下降的步伐。现在把它讲透。它是所有超参数里**影响力最大的一个**：同一份代码，只改它，能从“三分钟收敛”变成“永远训不动”。判断它合不合适，最直接的仪表是 loss 曲线。下图是同一个网络用三个不同学习率训练的曲线，三种病症一目了然。

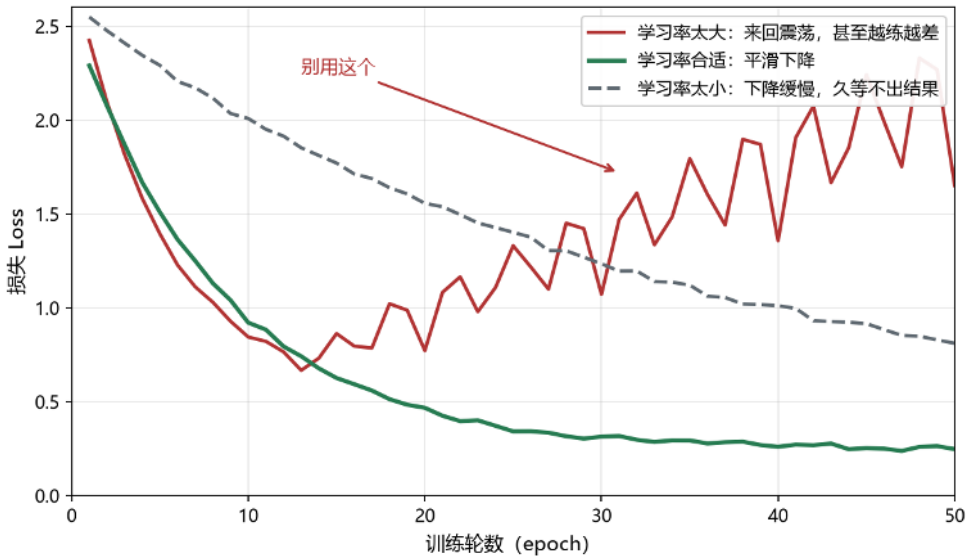



**图 3.2**　同一网络三个学习率的 loss 曲线：太大（剧烈震荡甚至上行）、合适（快速下降后趋平）、太小（缓慢下降，远未收敛）

逐条读图。**太大**：每步都跨过谷底，小球在两壁间弹来弹去，loss曲线剧烈震荡；再大一点，一步弹出场外，数值溢出，loss 变成 nan（not a number，非数）——上一讲说的现行犯一号在此归案：**训练打印出 nan，第一嫌疑人是学习率过大**。**太小**：曲线缓慢下降，看着有救，半天还在半山腰——GPU 烧着、曲线在动，你不敢停、停了又亏。**合适**：前段快速下降、后段趋于平坦，像一条滑梯，一路滑到谷底。

更进一步：好的训练往往不是“一个学习率用到底”，而是**先大后小**——前期大步赶路，后期小步精修。让学习率按计划变化的做法统称**学习率衰减**（learning rate schedule），三款常见配方混个脸熟即可：**阶梯衰减**（每过若干轮减半）、**余弦衰减**（沿余弦曲线从初值一路滑到近 0，2026 年视觉训练的常见默认）、**warmup**（开头几百步从极小值慢慢升到目标值——刚开训时参数还是“乱码”，太早迈大步容易当场翻车，大模型训练标配）。名词现在记住，实验四你会亲手拧。

先在纸上把“火候”的感觉找出来。

**动手时间：用纸笔把三个学习率各“训”五步**

取一维损失函数 $f(w) = (w-3)^2$（谷底在 $w=3$），它的梯度是 $2(w-3)$。用更新式 $w \leftarrow w - \eta \cdot 2(w-3)$，从 $w=0$ 出发，把三个学习率各手算三步：

| | $\eta = 0.05$（小火） | $\eta = 0.9$（中火） | $\eta = 1.1$（大火） |
|---|---|---|---|
| 第 1 步后 | 0.30 | 5.40 | 6.60 |
| 第 2 步后 | 0.57 | 1.08 | −1.32 |
| 第 3 步后 | 0.81 | 4.54 | 8.18 |

对答案：大火每步跨过谷底越跳越远（发散，再算几步就“nan”）；中火在谷底两侧来回弹（震荡）；小火稳稳向 3 爬但一步只挪零点几。请你继续把小火算到第 10 步——离 3 还有约一步之遥，这就是“龟速”的体感。整个过程只要一张草稿纸，你刚刚手工执行了三种“火候”下的梯度下降。课后编程题会让你把这三条曲线用代码真画出来。

## 3.3　权重初始化：第一步就走对

训练开跑前，权重总得有个初值。最省事的念头：全设 0。这恰恰是灾难，而且灾得很有趣。想想第 2 讲的反向传播：同一隐层的所有神经元，输入相同、权重相同，前向算出的输出一模一样；反向分账时它们收到的责任也一模一样，更新之后依旧相同——**一百个神经元永远步调一致，等价于一个神经元**，网络白白胖胖，能力却是“独苗”。这叫**对称性没有被打破**（symmetry breaking）。好比一个班发一模一样的课本、做一模一样的作业，还要求答案分毫不差——永远长不出各有专长的专家。所以初值必须随机，且各个神经元各不相同。

随机，但随成什么样也大有讲究。方差太大：信号一层层放大，传到深处激活值爆表，sigmoid/tanh 全员进入饱和区（回忆第 2 讲那片“导数近乎为零的平原”），梯度消失，loss 当场 nan——现行犯二号归案。**方差太小**：信号层层萎缩，到深处激活值全趴在 0 附近，网络整体“昏迷”。于是“刚刚好”的标准非常清晰：**让信号穿过每一层后，方差不爆不缩**——文献里叫方差匹配（variance matching）。

按这个原则，有两个记名字就能用的经典配方：**Xavier 初始化**（Glorot 等，2010，适合 sigmoid/tanh）：权重方差取 $2/(n_{\text{in}}+n_{\text{out}})$；**He 初始化**（何恺明等，2015，适合 ReLU）：取 $2/n_{\text{in}}$。人话翻译：$n_{\text{in}}$、$n_{\text{out}}$ 是该层进、出的连线数——**连线越多，每个权重就该越“谦虚”（越小）**，免得把一层的信号加爆。现代框架里通常一行代码自动完成，但出了问题你得知道它在防什么病。

**Xavier、He 与 Glorot 条件——一句话版**

为什么 Xavier 的方差是 $2/(n_{\text{in}}+n_{\text{out}})$，He 的却是 $2/n_{\text{in}}$？理想标准（文献称 **Glorot 条件**）其实有两条：信号**前向**穿过每层后激活方差不变，误差**反向**穿过每层后梯度方差不变。麻烦在于**两条不可兼得**——前向想把方差除以 $n_{\text{in}}$，反向想除以 $n_{\text{out}}$，于是各家做了不同取舍：

| 初始化 | 适用激活 | 方差条件（一句版） |
|---|---|---|
| Xavier | tanh / sigmoid | 两条各让一步的折中：$2/(n_{\text{in}}+n_{\text{out}})$ |
| He | ReLU | ReLU 掐掉一半信号，改为全力保前向：$2/n_{\text{in}}$ |
| LeCun | Leaky ReLU / SELU | He 思路的对应保守版：$1/n_{\text{in}}$ |

人话：Xavier 是“和稀泥大师”，两个条件各让一步，适合对称激活；He 发现 ReLU 把负半边全掐了，干脆牺牲反向条件、保住前向激活方差不为零；LeCun 初始化则是同一思路在斜率较小的整流激活（Leaky ReLU 一族）上的对应版。框架里都是拼一个名字的事（`xavier_init`、`kaiming_init`、`lecun_normal`），但报出名字，你得知道它在保什么。

下图就是上一讲加分项里留下的“预告片”正式上映：同一网络、同一数据、同一学习率，只有初始化不同，三条曲线命运迥异。全 0：曲线平得像心电图停机——对称性没破，网络学了个寂寞；太大：前段剧烈震荡甚至起飞；Xavier：平滑下降，一路健康。至此第 2 讲承诺的两个现行犯全部抓捕归案。**训练出问题的排查顺序：先查学习率，再查初始化，最后才怀疑更深的病。**

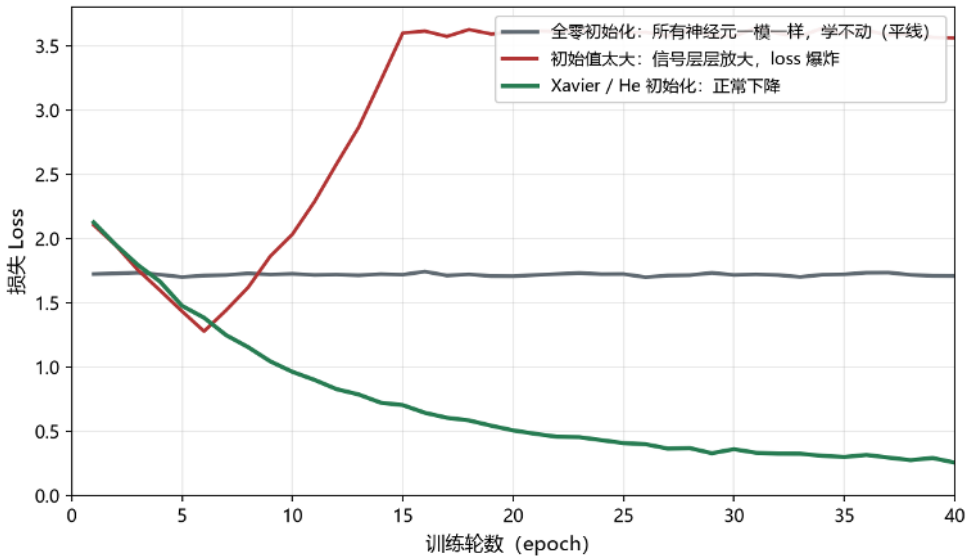



图 3.3　三种初始化的训练曲线对比：全 0（纹丝不动）、方差太大（震荡甚至爆炸）、Xavier（平滑收敛）

## 3.4　batch、epoch、iteration：分清训练的三兄弟

实操里还有三个容易搅浑的词，用吃饭比喻一次分清。训练数据是**全菜单**；**batch**（批）是一口吃的样本数，这一口有多大叫 batch size；**iteration**（迭代）是“咬一口、消化一次”——前向、算损失、反向、更新，完整一圈；**epoch**（轮）是**把整份菜单从头吃完一遍**。算笔账：1000个样本、batch size 取 100，一个 epoch 就是 10 次 iteration；训练 50 个 epoch，参数总共被更新了 500 次。

那“一口吃几个”怎么定？这是一笔典型的权衡账：

表 3.1　batch size 的权衡：大口稳但贵，小口省但吵

| batch size | 优点 | 代价 |
|---|---|---|
| 大（如 256、1024） | 梯度估计稳、路线直；GPU 并行吃得满，单位时间效率高 | 显存开销大；口太大“路太顺”，泛化有时反而变差 |
| 小（如 8、32） | 显存友好；噪声大，反而帮小球跳出鞍点，泛化常常更好 | 曲线抖、得配更小的学习率；单位样本效率低 |

两个极端各有名字：batch 取 1 叫随机梯度下降的“在线”模式，batch 取全量叫（全）批量梯度下降；夹在中间的叫 **mini-batch**（小批量）——现代训练的默认形态，以至于 2026 年大家说“SGD”时，指的几乎都是 mini-batch 版。它的更新式是本讲唯一的公式：

**知识补齐：mini-batch SGD 的更新式——现代训练的底座**

每一步，从训练集随机抽一小批 $\mathcal{B}$，用它估出的梯度下山：

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \eta \nabla_{\boldsymbol{\theta}} L(\mathcal{B}; \boldsymbol{\theta}_t), \tag{3.1}$$

人话：$\boldsymbol{\theta}$（粗体西塔）把全部权重和偏置打包装成一袋；不用全部数据算坡度（太贵），而是每步随机抽一小口当“抽样调查”，用抽样坡度代替真实坡度下山。噪声是特性不是缺陷：它让小球在鞍点附近也晃得动（呼应 3.1 节）。往第 2 讲的十行代码里加两行，就是标准的 mini-batch 训练：

```
for epoch in range(E):              # 外圈：全菜单吃 E 遍
    np.random.shuffle(idx)          # 每轮先洗牌，吃的顺序随机
    for B in batches(idx, bs):      # 内圈：切成一口一口
        theta -= lr * grad(theta, B)   # 一口一更新
```

再补一条工程常识：同一份代码你跑两次、同学跑一次，曲线并不重合——不是玄学。初始化、洗牌都是随机的，由**随机数种子**（*random seed*）统一控制。做对比实验（比如本章课后题）前先把种子固定，否则你分不清差异来自超参数还是来自运气。

## 3.5　训练集、验证集、测试集：背题与会做题

到目前为止我们默认“loss降 = 学好了”。这里有个陷阱：那个 loss 是在**训练数据**上算的。一个记忆力超群的学生把例题全背下来，做原题可以满分——换一道新题立刻露馅。而训练要的是**会做题，不是背题**。所以必须把手头的数据切成三份，扮演三个角色：**训练集**（training set，课本例题，用来更新参数）、**验证集**（validation set，平时小测，用来调超参数、判断何时收手）、**测试集**（test set，高考，只许在最后动用一次，报告最终成绩）。比例常见 8:1:1 或 7:2:1，数据特别少时还有“交叉验证”这种轮流坐庄的切法（记住名词即可）。

一条铁律先立起来：**所有训练中的决定——换学习率、早停、选模型——只能看验证集**。原因看下图就懂：训练 loss 一路下降，验证 loss 却先降后升；两条线开始“张口”的位置，就是学生从“会做题”滑向“背题”的转折点，图中标注的最佳停止点，就是该收工的地方。这个白捡的超参数叫**早停**（early stopping）：验证 loss 最低的那一轮就停手，别恋战。

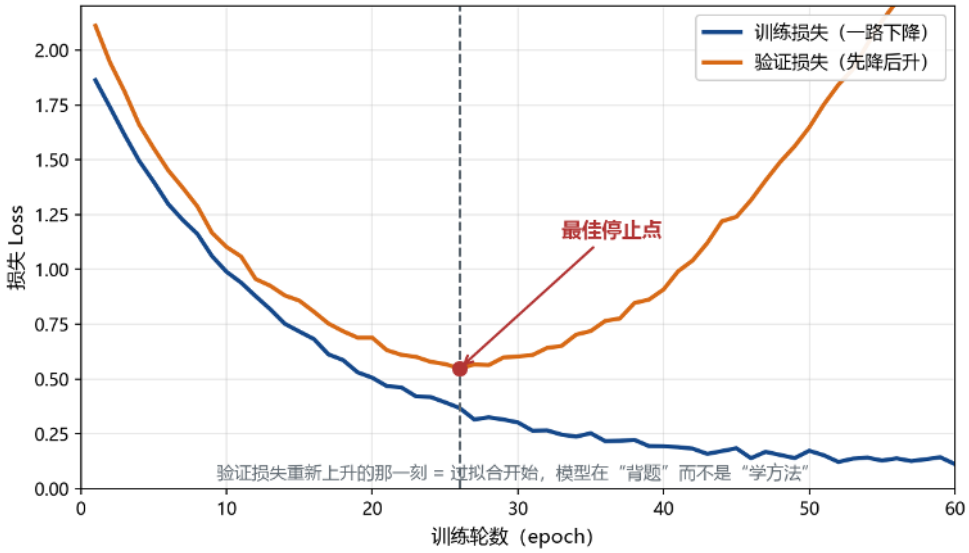



**图 3.4**　训练 loss（一路下降）与验证 loss（先降后升）双曲线：两线张口即过拟合起点，标注处为最佳停止点

两种典型病症总结成一张表，训练时对着自查：

表 3.2　训练病症速查：看两条 loss 曲线下诊断

| 病症 | 训练 loss | 验证 loss | 人话诊断 | 药方 |
|---|---|---|---|---|
| 过拟合 | 一直降 | 先降后升 | 背题了 | 更多数据；正则化；早停；dropout |
| 欠拟合 | 高且平 | 高且平 | 压根没学会 | 加大网络；训练更久；换更好的特征 |
| 刚刚好 | 降后趋平 | 贴着训练线降后趋平 | 健康收工 | 记下超参数，出成绩 |

表里两味新药各一句话入门：**正则化**（regularization，给过大的权重“加税”，逼网络学简单的规律）；**dropout**（训练时每一步随机“掐灭”一部分神经元，逼网络不依赖任何单个“学霸”，测试时全员上岗）。详细的用法第 6 讲见。

**防糊弄：高考卷一张纸都不许提前看**

用测试集调参数，等于把高考真题当练习册刷——成绩必然虚高，而且你永远不知道自己虚高了多少。这不是洁癖，是工程伦理：2024–2026 年大模型评测里，“benchmark 污染”（考题混进了训练数据）是审稿人重点打假的对象，一个刷爆榜的模型若考题不干净，成色立刻存疑。落到你们的项目验收上：六目九头的 A/B 对比里，`metrics.json` 中的每个指标都必须来自验收方指定、你没在上面调过参的数据。切分数据的那一行代码，要在训练开始之前写好、锁死。

## 3.6　向量化编程：为什么一行 numpy 顶一百行 for

实验三你会在真实数据上训练，最先撞上的墙往往不是数学而是速度：同一份作业，同学的代码一分钟出结果，你的跑一晚上还没收敛到一半。九成情况下，差距不在模型，而在**向量化**（vectorization）——把“逐个元素的循环”改写成“对整个数组一次性运算”。

为什么能快这么多？人话版：Python 解释器执行 for 循环时，每一圈都要“重新读一遍代码再执行”，开销巨大；而 numpy 的数组运算在底层是编译好的 C/Fortran 线性代数库（BLAS），一次把整块数据算完，还自动用上 CPU 的向量指令；数据若在 GPU 上，更是几万个小核心同时开工——第 2 讲那行 `X @ W1`，底下是上百万次乘加同时发生。实测感受一下：

```
import numpy as np                  # 导入 numpy，社区习惯缩写为 np
n = 1_000_000
x = np.random.rand(n)               # 一百万个随机数

total = 0.0                         # 写法一：朴素 for 循环
for i in range(n):
    total += x[i] ** 2              # 一次只算一个数

total_v = np.sum(x ** 2)            # 写法二：向量化，一行顶一循环
# 本机实测（量级）：写法一约几百毫秒，写法二约几毫秒
```

下图是这类对比的量级示意：同样算一百万个数的平方和，循环版与向量化版的耗时约 100:1。今后写训练代码，凡是出现“`for i in range(样本数)` 然后逐个处理”的地方，先停下来想一想：能不能整块算？

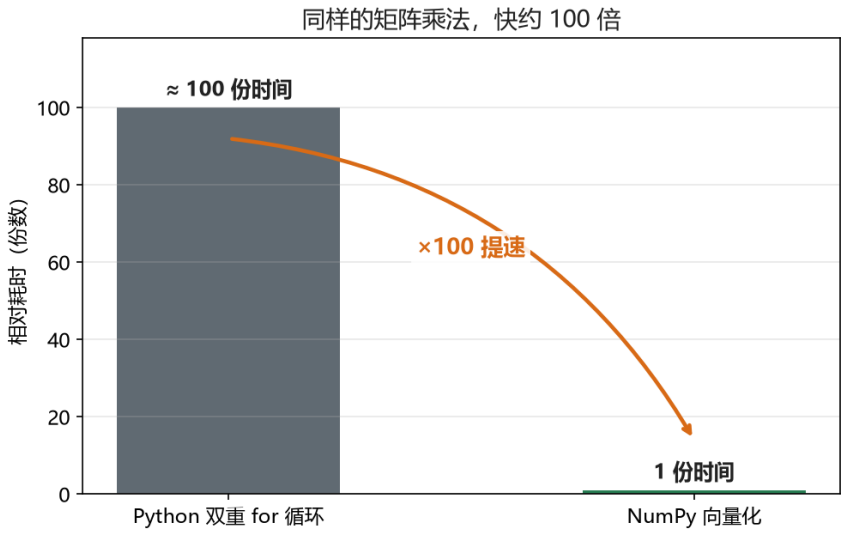



图 3.5　同样的计算，Python 循环与 numpy 向量化的耗时对比（量级示意约 100:1）

**知识补齐：numpy 广播——形状不同的数组也能直接运算**

**广播**（broadcasting）规则一句话：两个数组从最后一维往前对齐，每一维要么相等、要么其中一方是 1（是 1 的一方可自动“复制”扩展到匹配）。两个天天用的例子：

- `X -X.mean(axis=0)`：`X` 形状 $(100, 4)$，均值形状 $(4,)$，广播成 $(100, 4)$——一行完成“每个特征减去自己的均值”，这是数据标准化的核心动作。
- `W * s`：`W` 形状 $(3, 3)$，`s` 形状 $(3, 1)$，每行乘上不同的系数——一行完成“逐行缩放”，反向传播按行算梯度时天天用。

人话：广播就像自动复印粘贴，但并不真的占用那份内存。乘法、减法、比较都支持。

## 3.7　优化器家族一分钟：从 SGD 到 Adam

第 2 讲的更新式 $w \leftarrow w - \eta \partial L/\partial w$ 是最朴素的 **SGD**（stochastic gradient descent，随机梯度下降）：看到坡就迈步，所有参数共用同一个步长。七十年里，一批“改良版下山算法”陆续登场，本讲一分钟讲清最常用的四位——不推导，只记人话：

表 3.3　优化器家族速览：四位选手，各一句人话

| 优化器 | 一句人话 | 出生 | 点评 |
|---|---|---|---|
| SGD | 看坡下山，步长恒定 | 1951 | 朴素；调好了至今仍是 CNN 训练强手 |
| Momentum | 给小球加惯性，冲过小坑小坎 | 1964/1986 | 沟壑地形明显提速 |
| RMSProp | 每个参数自备步长：常走的路小步，少见的方向大步 | 2012 | “自适应步长”代表作 |
| Adam | Momentum + RMSProp：既带惯性又自适配步长 | 2015 | 2026 年的默认首选 |

用做菜的话收束：SGD 是死火候；Momentum 给锅装了个匀速转柄，冲劲带着它碾过小颠簸；RMSProp 给每根管子配了自己的阀门；Adam 说“我全都要”。实践口诀两条：**拿不准就用**

**Adam，学习率 0.001 起步**（Adam 对学习率不敏感，是它称王的主要原因）；卷积网络刷榜与论文复现里 SGD+Momentum 依然常见——它调好后泛化上限常常更高。四位在实验三的代码里都只是 `optimizer=` 后面换一个词的事，但你现在知道换的到底是什么。

**前沿速递：训练一个大模型要烧多少电**

把本讲的旋钮放大十亿倍，就是大模型的账单。GPT-3（2020 年，1750 亿参数）论文自报：一轮训练约 $3.14 \times 10^{23}$ 次浮点运算，用电约 129 万度——够一个三口之家用四百多年。GPT-4 级（2023 年）外界估算再高一到两个数量级：数千万度电，上万张 GPU 连转数月。于是 2024–2026 年上演了一场**效率竞赛**：**混合精度**（用低精度数字做大部分运算，显存与算力约省一半）、**FlashAttention**（2022 年，把注意力计算重新排布，又快又省显存）、**MoE 稀疏专家**（每次只唤醒一小部分参数干活）；2024 年底 DeepSeek-V3 公开报告用约 278.8 万 GPU 小时完成训练，把同级模型的账单砍低一个量级，引发全网对“训练成本”的重估。而“该烧多少”的理论依据是 **scaling law**（规模定律），三个词：**更大模型 + 更多数据 + 更多算力 = 可预测地更强**（2020 年提出；2022 年 Chinchilla 修正：数据要和参数同步加）。反过来看：正因为每次训练都贵得离谱，“用小实验先摸超参数、再放大”的调参手艺在 2026 年依然是稀缺技能——你每避免一次无效训练，省的都是几百万美元。本讲就是这门手艺的第一课。

**项目连线：实验三命令里的每个词，都是本讲概念**

实验三你会敲这样一条命令：`python train.py --epochs 200 --batch 8 --lr 0.001`。拆开看：`--epochs 200`，全菜单吃 200 遍；`--batch 8`，一口吃 8 个样本（显存小的机器跑得起的量级，回忆 batch 权衡表）；`--lr 0.001`，火候拧在 Adam 的常用起点。命令行就是往本讲的超参数字典里填空。**六目拼接**：九个参数头**逐头独立训练**——每个头一份自己的超参数字典（学习率、epoch、batch 各不相同，分类头和回归头脾气不同）；验收标准“逐头指标不劣于规则初值才允许替换”，意味着每个头都要做本讲的训练/验证切分、A/B 对比，并把成绩写进同名 `metrics.json`（第 一 章）。**水下双目**：学生模型从教师伪标签微调，采用“5 个 epoch 一道闸门”制度：每 5 个 epoch 在验证图对上测一次端点误差 EPE（end-point error，预测视差与真值视差的平均距离），不降即停——本讲的早停被写成了制度（第 二 章）。**数据基座**：问数系统里 Gemma 的**温度 0**（temperature 0：永远选最确定的词，不许自由发挥）与上下文长度 8192（一次能读进的 token——词元，大模型处理文本的最小单位——的数量）也是超参数：回答要“每次都一样”，靠的就是把温度拧到 0。它们与 `--lr` 本质同类：模型之外、人来定（第 三 章）。另外预告：实验四（12 月 21 日）的训练诊断环节，还有梯度检查的实战任务。

## 3.8　从这一讲到 2026

本讲的“训练集—验证集—测试集”三段论，正在被 2024–2026 年的实践改写，改写发生在三个方向。第一个方向：**数据有墙**。“更多数据就更强”的缩放定律（回忆章末前沿盒子里的 Chinchilla 修正）撞上了人类文本的储量上限——高质量公开文本被估计将在 2026 年前后“有效耗尽”，业界于是转向**合成数据**（让模型生成数据再来训模型）与多 epoch 重复利用。第二个方向：训练的重心从预训练（第 4 讲的“先学世界观”）移向**后训练**（post-training，基座模型训完之后的第二阶段，教它具体本事）——DeepSeek-R1 证明，不用人工示范答案、直接上大规模强化学习去“激励”，模型自己就能练出逐步推理的本事，这项工作 2025 年登上了《自然》封面。第三个方向最扎心：**评估会漏**。静态基准的考题会混进训练数据——正是本讲防糊弄盒子里那个“benchmark 污染”，它已严重到逼

出换代方案：每月换新题的**活基准**（live benchmark）成了主流，连顶级模型在其上的准确率也不到七成。数据有墙、评估会漏，唯有本讲教的训练方法本身，仍在快速进化。

**2026 现场：关键事实**

- Epoch AI 分析：按 Chinchilla 最优缩放，高质量人类公开文本将在 2026 年前后被有效耗尽，大模型必须转向合成数据、多 epoch 训练与推理时缩放（不加大模型、靠让它多想几步换性能）等新路径（来源：Epoch AI，“Will we run out of data to train LLMs?”，2024/2025）。
- DeepSeek-R1（2025 年 1 月 arXiv 预印本，2025 年 9 月发表于《自然》并登封面）：在大规模纯强化学习下，未经任何监督微调的 R1-Zero 自发出“自我反思、验证”等推理行为（来源：Guo et al.，Nature 645，633–638，2025）。
- LiveBench（ICLR 2025 Spotlight）：每月用最新数学竞赛、arXiv 论文与新闻出题、按客观真值自动评分，把基准污染压到最低；顶级模型在其六大类别上的准确率也低于 70%（来源：White et al.，ICLR 2025；livebench.ai）。
- 活基准已成体系：LiveNewsBench、LiveOIBench、DyCodeEval（ICML 2025）等相继出现，都用持续换题 / 动态生成对抗静态基准的数据泄漏（来源：各论文官方页面，2025）。

一句话收束：**你在这门课里学会的“留出测试集”，在 2026 年成了整个行业最难的事——测试题还没开考就可能已在训练集里，于是评测本身也必须“活”着。**

本讲的课后题，就是把上面所有概念放进同一份代码里做对照实验——这是“调参手艺”的第一次独立操练。五要素如下。

**第 3 讲课后编程题（用 AI 实现）**

**题目**：指挥 AI 辅助，做一个**超参数对比实验**。数据集用 sklearn 的双月牙 `make_moons`（或第 2 讲的 XOR，二选一）；训练同一个两层隐藏层 MLP 三次，**每次只改一个超参数**：三个学习率（如 0.001、0.1、10），或三种初始化（全 0、方差太大如标准差 1、Xavier）。其余设置完全相同（含随机种子）。用 matplotlib 把三条 loss 曲线画进同一张图（带图例），保存为 png；并写一段 50 字以上的结论：哪组最好、其余两组各得了本讲里的哪种“病”（用震荡、龟速、对称性、方差匹配这些词说）。

**交付物**：(1) 代码文件 `hyper_compare.py`；(2) 对比曲线 png 一张；(3) 50 字以上的实验结论；(4) 一段 50 字以上的“我改了 AI 代码里的什么、为什么”。

**提示词建议**（可直接抄）：“我是零基础学生，正在学超参数。请用 numpy（不用 PyTorch）实现：(1) 用 sklearn 的 make_moons 生成 200 个样本（噪声 0.2），按 8:1:1 切分训练/验证/测试集并标准化；(2) 一个 2-16-16-2 的全连接网络，ReLU 激活，输出层 softmax 配交叉熵损失（请用两句话给我解释 softmax 和交叉熵）；(3) 用普通 SGD（不用 Adam，这样才能看清学习率的威力），batch size 32，训练 500 个 epoch，随机种子固定 42；(4) 命令行参数 –lr，取 0.001、0.1、10 各训一次，其他完全相同；(5) 把三条训练 loss 曲线画进同一张图，带图例与中文标题，保存 png，并打印三组的最终验证准确率；(6) 每行中文注释，标明哪一行是学习率、哪里是初始化、哪里是 mini-batch 切分。”

**验收线**：代码能跑，三条曲线对比清晰、结论与曲线对得上（例如“`lr=10` 震荡甚至 nan，`lr=0.001` 龟速未收敛”）；抽问时能讲清 batch、epoch、iteration 三者的区别与 batch size 的权衡；至少一处自己的修改（换成三种初始化对比、加早停、改 batch size 都算）。**代码是 AI 写的没关**

**系，讲不清楚就是没做。**
**加分项**（任选其一）：(1) 给学习率加“阶梯衰减”（每 200 轮减半）重跑，对比曲线尾部形状；(2) 把数据集噪声从 0.2 加大到 0.4，看三种学习率的命运差距如何拉大；(3) 开训前先做第 2 讲的梯度检查，把“通过后再训练”写成流程。

课后阅读：教材《深度学习模型与算法基础》第 31–35、48–50 页，与本讲内容对应；读的时候带着本讲的“旋钮清单”去对照，看书上哪些词你已经能用人话复述。

**超参数（hyperparameter）**
训练开跑前由人设定的旋钮，如学习率、batch size、轮数；区别于训练中由反向传播自动学习的参数。

**学习率（learning rate）**
每步参数更新的步长，最敏感的超参数：太大震荡甚至爆炸成 nan，太小龟速；常配衰减策略（阶梯、余弦、warmup）。

**权重初始化（weight initialization）**
训练前给权重的随机初值：全 0 破坏不了对称性，太大梯度爆炸，太小梯度消失；经典配方 Xavier 与 He，原则是方差匹配。

**batch / iteration / epoch**
一口吃的样本数 / 咬一口消化一次（更新一次）/ 全菜单吃完一遍；batch size 是“稳与贵、噪与省”的权衡。

**mini-batch SGD**
每步随机抽一小批数据估梯度并更新的算法，现代训练的底座；噪声帮小球跳出鞍点。

**训练集 / 验证集 / 测试集**
课本例题 / 平时小测 / 高考；一切训练中的决定只看验证集，测试集只许最后用一次。

**过拟合 / 欠拟合**
背题了 / 没学会；用训练与验证 loss 的“张口”与“齐平”诊断，药方分别是更多数据与正则、加大网络。

**早停（early stopping）**
在验证 loss 最低处收工，白捡的超参数。

**向量化（vectorization）**
把逐元素循环改写成整块数组运算，numpy 与 GPU 提速百倍的根源。

**Adam**
动量与自适应步长的合体优化器，对学习率不敏感，2026 年的默认首选；拿不准就用它，0.001 起步。

# 第 4 讲　自编码器

把一张 3 MB 的照片发给朋友，你的第一反应是压成 zip——压缩程序的本事，是在数据里找重复规律，抠掉冗余。可如果压缩对象是一本 1001 页的设备手册呢？人类的压缩方式是写一页摘要：摘要没法一键还原出全文，却留住了最有用的信息。**这种压缩需要“理解”**。**自编码器**（autoencoder，常缩写为 AE）就是教神经网络学会“理解式压缩”的第一课：给它看大量数据，让它自己琢磨什么该留、什么该扔。

更妙的是，这一课**不需要任何标签**——数据自己就是自己的标准答案。这为理解 2026 年最重要的一件事埋下伏笔：GPT 这类大模型的看家本领，根基正是这种“无标签、自己出题”的学习方式。本讲从自编码器出发，一路走到“基础模型”时代的表示学习——你会看到，第 3 讲学的全套训练手艺，在一个没有标签的世界里照样通用。

**本讲地图**

**讲什么**：无监督与自监督学习为何重要；自编码器的沙漏结构与重建损失；隐空间与插值；去噪自编码器；稀疏 AE 与 VAE 一分钟；预训练—微调范式。**为什么值得学**：自编码器是“表示学习”的敲门砖；三个项目里 FoundationStereo 的隐空间、拼接头的“指纹”，玩法源头都在本讲。**学完你能**：画出沙漏结构并说清瓶颈的用意；解释隐空间插值为什么是“渐变的人脸”；在 sklearn 手写数字集上训一个自编码器，并画出隐空间的散点图。

## 4.1　无监督学习登场：没有标签的世界

前两讲的训练都长一个样：输入配标准答案（花瓣尺寸配花种、像素配数字），网络对着答案改错。这叫**监督学习**（supervised learning）——“监督”指答案像监考老师一样全程在场。好用，但贵：标签要人一条条标。ImageNet 为了给 1400 多万张图片打标签，动用了数万名众包标注员干了几年。而现实世界的处境恰恰相反：**数据泛滥、标签稀缺**——全网有几十亿张照片，没有哪张自带“这是猫”的标签；你们水下项目的 500 对锚链立体图对，在教师模型来打伪标签之前，也没有一个像素被标注过（第 二 章）。

于是换思路。**无监督学习**（unsupervised learning）不给答案，让模型从数据自身找结构：聚成几堆、压成几维、谁和谁像。自编码器就是其中最直观的一员——把输入喂进去，让网络把输入**重建**出来。听起来像左手抄右手？玄机全在中间那道“窄门”，下一节见。而在两者之间还有第三条路——**自监督学习**（self-supervised learning）：不请人来标，但把数据自己加工成“题目 + 答案”：遮住一个词让模型猜词、抠掉一块图让模型补图。题目和答案都从数据里来，一分钱标注费不花。

**表 4.1**　三种学习范式对比：谁出题、谁给答案

| 范式 | 谁出题 | 答案从哪来 | 2026年的代表 |
|---|---|---|---|
| 监督学习 | 人出题（配好标签） | 人给的标签 | 图像分类、目标检测 |
| 无监督学习 | 数据自身 | 无答案，只找结构 | 聚类、自编码器、降维 |
| 自监督学习 | 数据自己加工出题 | 藏在数据里 | GPT 预训练（猜下一词）、BERT（猜被遮的词）、MAE（补图块） |

给 2026 年留个伏笔：大模型的“预训练”阶段吃下几乎整个互联网的文本，用的正是自监督出题；等它学完“世界观”，才轮到少量人工数据做“专业课”。下图是这条主线的历史全景——注意 2017 年之后最粗的那根线，就是自监督预训练。

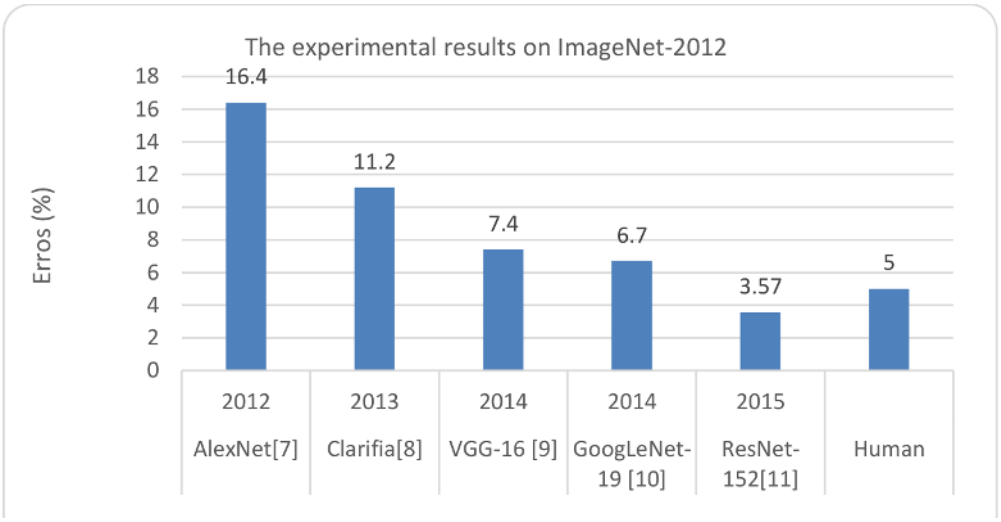



**图 4.1**　深度学习发展的里程碑时间线：从感知器到大模型，无监督与自监督是贯穿后半场的主线

图片来源：维基百科 Deep learning 词条

## 4.2　自编码器结构：自己压缩自己

结构看下图，一个“沙漏”：左边**编码器**（encoder）逐层变窄，负责压缩；正中间最窄处叫**瓶颈**（bottleneck，也叫隐层），信息必须过这道独木桥；右边**解码器**（decoder）逐层变宽，负责还原。训练目标只有一条：**输出尽量等于输入**。度量它的叫**重建损失**（reconstruction loss）：

$$L = \left\|\mathbf{x} - \hat{\mathbf{x}}\right\|^2, \tag{4.1}$$

人话：$\mathbf{x}$ 是输入，$\hat{\mathbf{x}}$（读“$x$ 尖”）是重建输出，把两者逐个维度求差、平方、加总——重建得越像，损失越小。这就是第 *1* 讲的平方误差换了个舞台。

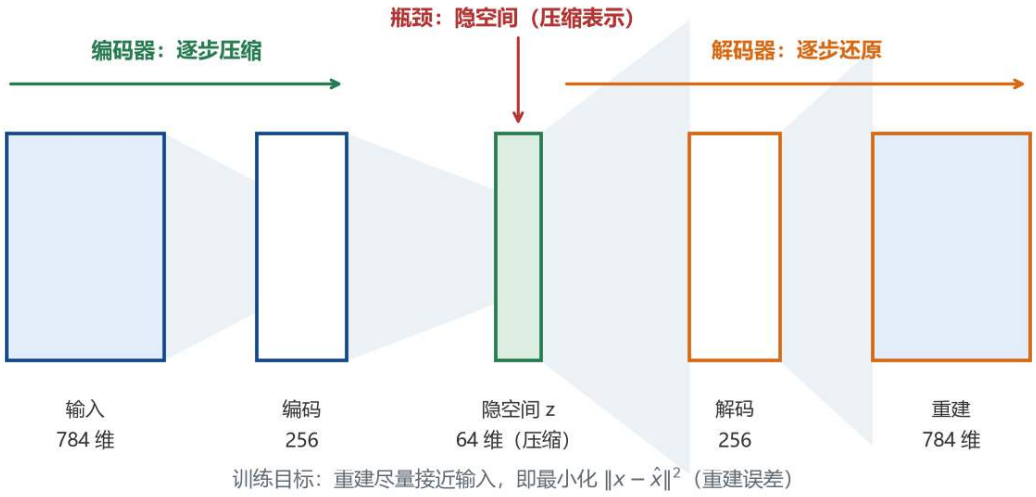



**图 4.2**　自编码器沙漏结构：784 维输入（$28 \times 28$ 手写数字）经编码压到 64 维隐码，再解码重建回 784 维；中间的窄门逼网络学会取舍

瓶颈为什么必须窄？做个思想实验：如果中间层比输入还宽，网络会学会最无耻的解法——恒等映射，把输入原样抄到输出，重建损失为 0，但什么也没学到，等于花一个亿盖了个复印机。**瓶颈是故意收的“信息税”**：64 维装不下 784 维的全部细节，网络被逼到墙角，只能学会分辨“哪些是本质、哪些是边角料”。压缩的意义由此产生。训练流程与第 3 讲毫无二致：前向、重建损失、反向传播、更新——唯一的区别是“标签”就是输入自己。一句话总结：**自编码器就是用反向传播训练出来的、可学习的压缩算法**。

**生活化类比：zip 与自编码器——两种压缩的对话**

zip 是无损压缩：逐字节找重复，能一字不差还原，但它“不懂内容”，对毫无重复规律的照片就无能为力。自编码器是有损压缩：丢掉它认为不重要的细节，还原出来的是“神似”而非像素级相同。这笔亏换来的赚头是：**隐码带着语义**——两张都像“8”的图，隐码也挨得近；zip 的压缩结果没有这种性质。前者像复印机，后者像“看完电影给朋友复述剧情”：说不全，但抓住了主线，而且两部电影若剧情相似，复述稿也相似。

## 4.3　隐空间：压缩后的向量是“本质坐标”

瓶颈处那个 64 维向量，叫**隐码**（latent code）。换个视角看它：每张图被映射成 64 维空间里的一个点，这个空间叫**隐空间**（latent space）。训练好的自编码器有一个漂亮性质：**长得像的输入，在隐空间里挨得近**——两张“8”的隐码距离，远小于“8”和“0”的距离。换句话说，隐空间把“像素的海洋”重新组织成了一张“按本质分类的地图”：下图中同类样本自然聚成簇，不需要任何标签。

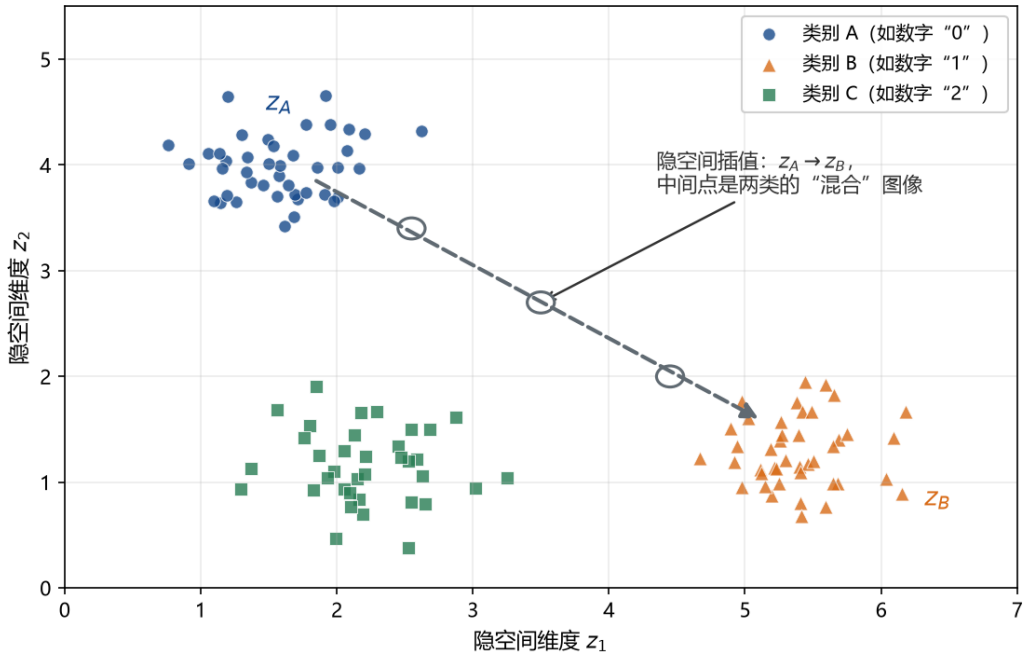



图 4.3　二维隐空间散点（按类别着色）：同类样本聚成簇；两簇之间的插值箭头表示隐码沿连线渐变时，解码出的结果平滑过渡

更神奇的是**插值**（interpolation）。取两张人脸的隐码 $\mathbf{a}$、$\mathbf{b}$，把连线上的点 $\mathbf{z} = (1-t)\,\mathbf{a} + t\,\mathbf{b}$（$t$ 从 0 到 1）逐个送进解码器，得到的是从一张脸平滑渐变到另一张脸的过渡序列：眉毛、鼻子、嘴巴逐个“变形”。注意这不是两张照片的像素叠加——叠加只会得到重影——而是每一点都由解码器重新“画”出一张完整合理的脸。人话翻译：**隐码不是照片的存档，而是脸的参数旋钮；旋钮连续转，脸**

**就连续变**。这说明网络学到的表示，抓住了“生成一张脸”的本质自由度。这个性质立刻能变现：高维数据降到二维看结构（可视化）、离群点检测（隐空间里的孤点）、以及最重要的——**表示**：既然隐码就是“本质特征”，就可以拿它去做分类、检索等其他任务，全程不用标签。这最后一步跨越，是 4.5 节和前沿盒子的主题。

先玩两个纸面游戏，把“瓶颈”和“插值”变成体感。

**动手时间：当一回人肉自编码器，再手算一次插值**

**游戏一（人肉自编码器）**：两人一组。甲看一幅简笔画 10 秒，然后用**不超过 10 个词**描述给乙（这 10 个词就是“瓶颈”——你被迫做取舍）；乙凭描述把画复原。复原结果当然与原图不同，但只要“是一只戴帽子的猫”这类本质还在，压缩就成功；若乙画出了一只没戴帽子的狗，说明瓶颈太窄，本质丢了。试三个瓶颈宽度：10 个词、5 个词、2 个词，看信息在哪一档崩掉。**游戏二（纸笔插值）**：取两个隐码 $\mathbf{a}=(1,0)$、$\mathbf{b}=(3,2)$，在方格纸上标出这两点，用 $\mathbf{z}=(1-t)\mathbf{a}+t\mathbf{b}$ 取 $t=0.25,0.5,0.75$ 算出三个中间点（$(1.5,0.5)$、$(2,1)$、$(2.5,1.5)$），标在连线上——四个点等距排列。想象每个点都由解码器还原成一幅图：这就是“渐变配方”。课后题里你会用真网络把它做出来。

**知识补齐：TSNE——把高维点云拍扁到纸上**

16 维的隐码没法直接画。**TSNE**（t-distributed Stochastic Neighbor Embedding，t 分布随机邻域嵌入）是最常用的“拍扁”工具，一句话原理：**尽量让高维空间里挨得近的点，在二维图上也挨得近，同时把不同的簇彼此拉开**。用法两行：

```
from sklearn.manifold import TSNE
Z2 = TSNE(n_components=2).fit_transform(Z)   # Z 是全部样本的隐码
```

注意：TSNE 只用来**看**，不用来训练；图上簇的形状、大小、簇间距都别过度解读，重点是“同类分不分得开”。

## 4.4　去噪自编码器：先弄脏，再学会擦

普通自编码器有个隐患：如果瓶颈不够窄，网络可能练成“死记硬背型复印机”——重建得很好，但没有学到结构。2008年提出的**去噪自编码器**（denoising autoencoder，DAE）的解法带着一股物理实验味：**先把输入弄脏，再要求还原出干净的**。具体说：给输入 $\mathbf{x}$ 加噪声得到脏版本 $\tilde{\mathbf{x}}$，网络看的是脏图，损失却对着干净原图算：

$$L=\left\|\mathbf{x}-D\big(E(\tilde{\mathbf{x}})\big)\right\|^2, \tag{4.2}$$

人话：$E$ 是编码器、$D$ 是解码器；输入是带噪声的 $\tilde{\mathbf{x}}$（读“$x$ 波浪”），但评分标准是“离干净的原版 $\mathbf{x}$ 有多近”——只抄写不得分，必须把噪声擦掉。

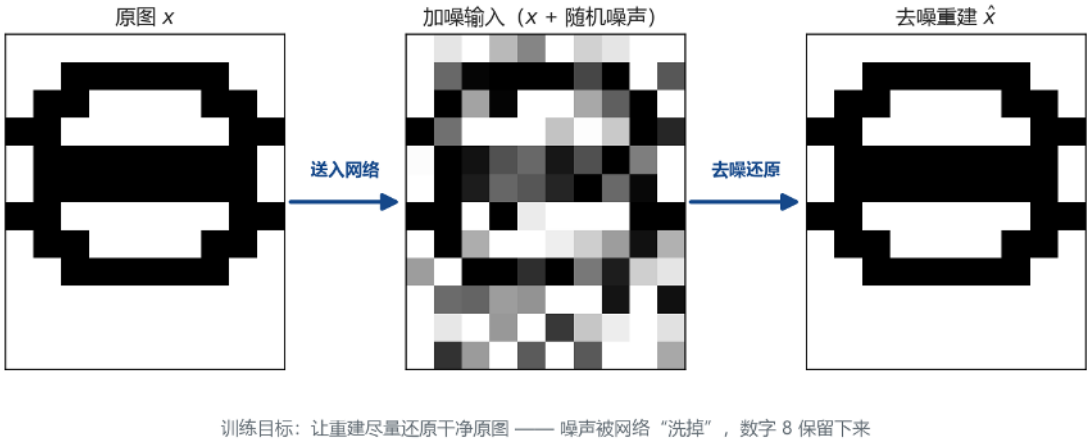



**图 4.4**　去噪自编码器三联图：干净原图（左）加噪变脏（中）送入网络，重建（右）不仅恢复笔画还顺手抹掉了噪声（以手写数字 8 为例）

为什么这样学到的特征更鲁棒？人话：如果输入带噪声而目标干净，网络就不可能靠“原样抄写”得高分——抄捷径的路被堵死了。它必须学到“这一笔大概是 8 的那个圆圈”级别的**结构性知识**，才能把噪声擦掉而不擦伤笔画。**强迫理解，不许复述**。这套“弄脏再还原”的自出题套路，是本讲最重要的思想遗产，2026 年它无处不在：BERT 遮词猜词是文本版去噪，MAE 抠图块补全是图像版去噪，第 12 讲扩散模型的“从纯噪声逐步去噪出图”是它的直系后代——大模型预训练的题目，多数是它的变体。

## 4.5　变体一分钟与“AE 的当代后代”

在上家族合影之前，先把教材里的两位正式成员请上台。**正则自编码器**（regularized autoencoder）：在重建损失之外给权重加 L1 / L2 惩罚（就是第 3 讲那句“给过大的权重加税”的同款手艺，也叫权重衰减）——相当于**给隐层加纪律**：不许死记训练集的细枝末节，想重建得像，就只能靠学到的结构。

**稀疏自编码器：让大多数神经元保持沉默**

稀疏 AE 给隐层加的不是“窄”，而是“闲”。**稀疏性约束**：给每个隐神经元定一条纪律——**平均激活度** $\rho$ 必须很小（比如 0.05：一百次输入里平均只醒五次）；训练时统计每个隐单元的实际平均激活度 $\hat{\rho}$，谁偏离了目标 $\rho$，就用 **KL 散度**（KL divergence，衡量两个概率分布差多远的量，偏差越大罚得越重）写进损失里罚它。罚项把整个隐层压得**大部分时刻沉默**。人话：强迫网络用更少的活跃神经元表达同样的内容，逼出真正有用的特征——像整顿一支人浮于事的团队，每个人必须练出独当一面的本事。教材里“开关式特征、可解释性好”说的就是它：一个只在看到竖线时才亮的神经元，本身就是一条可读的规则。

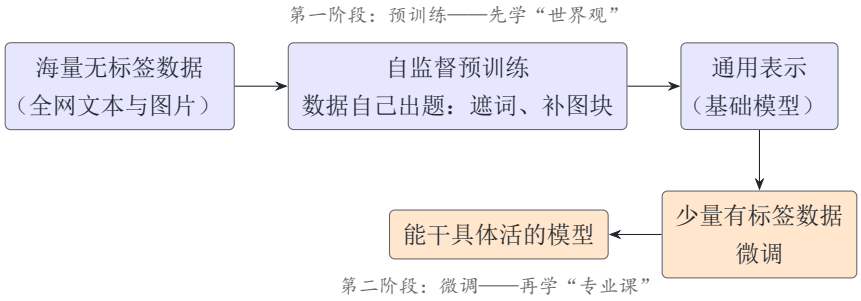
自编码器家族枝繁叶茂，本讲一分钟认全主要成员：

表 4.2　自编码器家族速览：各自加了什么、图什么

| 成员 | 加了什么 | 一句话用途 |
|---|---|---|
| 稀疏自编码器 | 隐层加“稀疏税”：多数隐单元须为 0 | 学出“开关式”特征，可解释性更好 |
| 变分自编码器（VAE） | 隐码改成从一个分布采样，并对分布形状加正则 | 隐空间连续平滑，能采样生成新样本——生成模型的起点（第 12 讲） |
| 卷积自编码器 | 编码、解码器换成卷积层 | 图像任务标配，参数更省、更懂像素的邻居关系 |
| 当代后代（BERT/MAE 等） | “遮住/弄脏，再还原”的自监督出题 | 大模型表示学习的雏形 |

表中再补两句人话。稀疏 AE 的用途刚才盒子里已经讲了，这里只打个比方：让每个隐单元专职检测一种特定模式——像给一百个员工各定一个岗位，不许扎堆。VAE：让隐码不再是一个点，而是一个“云团”（概率分布），并对云团的位置形状提要求，于是隐空间处处连续、随便取一点解码都是合理样本——自编码器由此从“压缩”跨进“生成”，第 12 讲还会遇到它的思想。

对 2026 年最重要的是最后一行。把“自出题、找结构”放大到极限，就是当代 AI 的生产流水线：**预训练**（pre-training）——先在海量无标签数据上做自监督训练（猜被遮的词、补被抠的图块），学到通用表示；**微调**（fine-tuning）——再用少量有标签数据，把这套表示适配到具体任务。两段式如下图：




一句话总结：**先无标签学“世界观”，再少量标签学“专业课”**。2026 年几乎所有语言与视觉模型都按此流水线生产。你们水下项目的教师模型 FoundationStereo 就是立体视觉界的基础模型：先在千万级立体数据上学出通用的立体匹配表示，下游小数据微调即用——你们做的“教师打伪标签、学生学着像教师”，本质上是把它的隐空间表示蒸给学生（第二篇第 二 章）。

**动手时间（纸上）：逐层贪婪预训练——Hinton 2006 的破局操作**

2006 年的死局是：网络一深，反向传播的学习信号到不了底层（第 2 讲饱和区的放大版），深层网络怎么训都输给浅层。辛顿团队的破局操作朴素得像搬砖——**一次只训一层**，学名**逐层贪婪预训练**（greedy layer-wise pre-training）：

1. 第一轮：只把输入接到第一个隐层，把它当**一台单层自编码器**训练到能重建输入，然后**固定**这层权重；
2. 第二轮：把第一层的输出当作新输入，训第二个隐层的自编码器，训完同样固定；
3. 如此逐层向上（堆叠 AE、堆叠稀疏 AE、堆叠去噪 AE，套路全同），最后把整座网络按第 3 讲的全套手艺整体**微调**一遍收工。

拿笔在纸上画一遍 784→196→49 的两层 AE：标出两轮训练里哪些权重在动、哪些被冻结——画完你就懂了“贪婪”二字：每层只顾自己重建得最好，从不保证对整座网络最优，所以叫逐层**贪婪**。人话总结：像盖楼，一层验收封顶再盖下一层，每层的起点都是上一层学到的有用表示而非“乱码”——**这是 2006 年深度学习复兴的钥匙操作**，“深度学习”这个名字也诞生于这几篇论文。学术争议也要如实交代：后来人们发现，有了 ReLU、批归一化和好的初始化（第 3 讲），逐层预训练在多数任务上**不再必要**——但把领域从寒冬里捞出来的历史转折点，非它莫属：没有 2006，就没有 2012。

**前沿速递：表示学习——深度学习的灵魂**

深度学习为什么叫“深”？表面看是层数，实质是**表示学习**（representation learning）：每一层把上一层的表示再抽象一步，边缘拼成纹理、纹理拼成部件、部件拼成物体（下一讲你会看到 CNN 的同款层级）。自编码器是这件事的最小完整闭环。2024–2026 年被称为“基础模型”（foundation model）时代：先在无标签海量数据上学通用表示，再小数据微调甚至零样本直接用——GPT-4、一键分割一切的 SAM、图文对齐的 CLIP 都是；视觉这边，你们项目里的 FoundationStereo（立体匹配）与 Depth Anything（单目深度）也是。两块里程碑各记一句话：simCLR（2020，对比学习——同一张图两次随机增强应“挨着”，不同图应“远离”，无标签学出好表示）；MAE（2021，遮住 75% 的图块让模型补全，把“去噪出题”推到极限）。最后送你一个检验任何模型“有没有表示学习的灵魂”的问题：**把标签全部摘掉，它还能从数据里学到什么？**——学不到东西的，只是台背题机器。

**项目连线：压缩，但不丢关键信息**

**六目拼接**：九个参数头里的 `descriptor_softening` 头，把 $32 \times 32$ 的图像块压成 256 维“指纹”（descriptor，描述子）——标准的自编码器式压缩：上千个像素值被压进 256 维，目的不是省存储，而是**让“像”的图像块在指纹空间里真的挨得近**，两路相机的图像块靠指纹比对完成匹配，比逐像素硬比稳得多（第 一 章）。**水下双目**：浑水弱光下，人工设计的匹配特征频频失效，项目改用深度基础模型的**隐空间表示**替代手工特征——教师打伪标签、学生模仿教师，本质是“把教师的表示蒸给学生”（第 二 章）。**数据基座**：把 1001 页手册压成结构化 JSON，同样是“压缩但不丢关键信息”：摘要式压缩、瓶颈式取舍，而“什么算关键”由问数场景定义——能支撑回答问题的字段一个不能丢，装饰性排版全可扔（第 三 章）。三条线共享同一句话：压缩的境界不在压得多小，而在丢得对不对。

## 4.6　从这一讲到 2026

自编码器“压缩再重建”的思想，在 2026 年无处不在，三条支线都还认得本讲的沙漏。第一条：**视频生成**。阿里 Wan、Sora 这类视频大模型，第一步都是先用 VAE 把整段视频压进低维的**时空潜空间**（时间和空间一起压：一段视频变成一小块三维“隐码”），再在潜空间里做扩散去噪——VAE 成了视频生成的“压缩器”，扩散模型的真身第 12 讲见。第二条：**自监督视觉骨干**。Meta 的 DINOv3 不用一张人工标签，在 17 亿张图片上自监督训练出 70 亿参数的通用视觉骨干（ViT，一种把图像切块当“字”来读的注意力网络，第 11 讲的主角），特征拿来就能分类、分割、检索——这是本讲前沿盒子那句“把标签全部摘掉，它还能学到什么”的满分答卷。第三条走得最远：**世界模型**。JEPA 一系把“在像素里重建”升级为“在嵌入空间里预测”——不逼模型补齐每个像素，只要求它预测“这幅

画面的抽象表示该长什么样”，V-JEPA 2 用超百万小时互联网视频学出对世界运动规律的直觉，只在约 62 小时机器人视频上后训练，就能零样本操控陌生实验室的机械臂。从 AE 到 VAE 到 MAE 再到扩散与 JEPA，主线一以贯之：**把世界压进好的向量**。

**2026 现场：关键事实**

- 阿里 Wan 团队 2025 年 3 月开源整套视频生成大模型（1.3B / 14B 两档，1.3B 版仅需约 8.19 GB 显存），采用扩散 Transformer 范式：自研 VAE 先把视频压进低维时空潜空间再行去噪——本讲的“编码器—潜空间—解码器”就是它的生产形态（来源：Team Wan，arXiv 2025，编号 2503.20314）。
- OpenAI 于 2025 年 9 月 30 日发布 Sora 2（旗舰级视频 + 音频生成模型，支持同步声音与更真实的物理表现）；Sora 类模型同样在 VAE 压缩的潜空间中做扩散生成（来源：OpenAI 官方博客“Sora 2 is here”，2025）。
- Meta 2025 年 8 月发布 DINOv3：纯自监督、零人工标注，在 17 亿张精选图像上训出最高 70 亿参数的 ViT 视觉骨干，并提出 Gram anchoring 技术防止长训练中的特征退化，特征可即插即用于分类、分割、检索（来源：Meta AI Blog 及 arXiv 2508.10104，2025）。
- Meta 的 V-JEPA 2（2025 年 6 月）在超 100 万小时互联网视频上自监督预训练；其动作条件版 V-JEPA 2-AC 仅用约 62 小时无标注机器人视频做后训练，即可在两个陌生实验室的 Franka 机械臂上零样本完成抓取与放置（来源：Assran, Bardes, LeCun et al.，arXiv 2025，编号 2506.09985）。

一句话收束：**从 1991 年的自编码器到 2025 年能想象世界的模型，演化逻辑从未变过：先学会把世界压进潜空间，才能在潜空间里想象世界。**

本讲课后题，是亲手把一台自编码器从压缩训到可视化——顺带把第 3 讲的训练手艺（切分、监控、早停）全用上。五要素如下。

**第 4 讲课后编程题（用 AI 实现）**

**题目**：指挥 AI 辅助，在 sklearn 自带的手写数字集 `load_digits`（1797 张 $8\times8$ 灰度图，共 64 维）上训练一个自编码器：结构 $64 \to 32 \to 16 \to 32 \to 64$（隐码 16 维，约为原维度的四分之一），训练完后交付两样可视化：(1) “原图 vs 重建”对比图——上下两排各 8 个数字，用 matplotlib 画成图像网格；(2) 把全部样本的 16 维隐码用 TSNE 拍扁到二维，画散点图并按 0–9 数字类别着色（十种颜色），看同类是否聚簇。

**交付物**：(1) 代码文件 `digits_ae.py`；(2) 两张 png（重建对比图、TSNE 散点图）；(3) 50 字以上的结论：16 维隐码够不够装下数字的“本质”？哪些数字的簇容易黏在一起、为什么；(4) 一段 50 字以上的“我改了 AI 代码里的什么、为什么”。

**提示词建议**（可直接抄）：“我是零基础学生，正在学自编码器。请用 Python 写一个自编码器：(1) 数据用 sklearn 的 load_digits，按 8:1:1 切分训练/验证/测试集，特征除以 16 归一化到 0–1；(2) 网络结构 64→32→16→32→64，隐码 16 维，ReLU 激活（输出层用 sigmoid），平方误差损失，小批量训练并监控验证损失做早停；(3) 训练完用 matplotlib 画‘原图 vs 重建’上下两排各 8 个数字的对比图，保存 png；(4) 用 sklearn 的 TSNE 把训练集全部隐码降到二维，画散点图，按数字类别 0–9 着色并带图例，保存 png；(5) 每行中文注释，标明哪一行是编码器、哪一行是解码器、哪一行取隐码。”

**验收线**：两张图都生成且能讲清——重建图能指出“丢了什么细节、保住了什么笔画”，TSNE 图上至少能辨认出 2–3 个明显分开的簇，并解释为什么 4 和 9、3 和 8 这类数字的簇容易粘连；能用自己的话说出瓶颈的作用；至少一处自己的修改（隐码改 8 维或 32 维对比、给输入加噪声改成去噪 AE、换激活函数都算）。**代码是 AI 写的没关系，讲不清楚就是没做。**

**加分项**（任选其一）：(1) 把隐码直接压到 2 维重新训练，对比“AE 二维隐码”与“AE 十六维隐码 + TSNE”两张散点图的形状差异，想想为什么不一样；(2) 训练时给输入加高斯噪声做成去噪自编码器，对比重建图的变化；(3) 拿训练好的隐码接一个 k 近邻分类器（digits 自带标签），报告“无监督特征 + 有监督分类”的准确率，与直接用原始 64 维像素对比。

课后阅读：教材《深度学习模型与算法基础》第 35–48 页，与本讲内容对应；重点看书中自编码器结构图与隐层的说法，与本讲沙漏图对照着读。

**无监督学习（unsupervised learning）**

不用人工标签、从数据自身找结构的学习方式；自编码器、聚类、降维都属于这一路。

**自监督学习（self-supervised learning）**

把数据自己加工成“题目 + 答案”（遮词、补图块）再学习；2026 年大模型预训练的标准形态。

**自编码器（autoencoder）**

编码器—瓶颈—解码器的沙漏网络，以“重建输入”为目标训练，学到压缩表示。

**编码器 / 解码器（encoder / decoder）**

沙漏的左半与右半：一个负责压缩、一个负责还原。

**瓶颈（bottleneck）**

沙漏最窄处的隐层；故意收窄的“信息税”，逼网络分辨本质与边角料。

**隐码与隐空间（latent code / latent space）**

瓶颈处的向量，及其所在的高维空间；相似输入在隐空间里彼此靠近，插值即渐变。

**重建损失（reconstruction loss）**

输出与输入的差距，常用平方误差；自编码器的“标准答案”就是输入自己。

**去噪自编码器（denoising autoencoder）**

输入加噪、目标干净的变体；堵死抄捷径的路，逼出鲁棒的结构性知识。

**变分自编码器（VAE）**

隐码改为分布采样并加正则的变体，隐空间连续、能采样生成新样本，是生成模型的起点之一。

**表示学习（representation learning）**

让模型自动学出数据的本质特征；深度学习之“深”的实质，基础模型时代的核心资产。

# 第 5 讲　卷积神经网络原理与前向计算

1968年，哈佛大学的休贝尔（Hubel）与维泽尔（Wiesel）把微电极插进麻醉猫的视皮层，向幕布投出各种线条，记录单个神经细胞的放电。他们发现了一个惊人的分工：**有些细胞只对特定方向的边缘放电**——竖直边缘经过时猛烈放电，线条一转就沉默；这类“边缘侦探”密密麻麻铺满视皮层的第一站。这项工作获得 1981 年诺贝尔生理学或医学奖，更重要的是它给出了一张工程蓝图：**视觉是从“找边缘”开始的，再一层层向上拼装出“认东西”的能力**。

1989 年，贝尔实验室的 LeCun 对着这张蓝图想：生物如此，机器为何不照抄？他把手写数字图片直接喂进一种带“局部小窗口 + 模板全图复用”机制的网络——**卷积神经网络**（Convolutional Neural Network，CNN），第一次让机器可靠地读出成千上万张支票与邮编上的手写数字。生物学出蓝图，反向传播来施工，这一讲讲它的原理与前向计算；怎么训练它，是下一讲（第 6 讲）的事。

**本讲地图**

**讲什么**：全连接看图为何参数爆炸；卷积 = 滑动窗口做点积；三大设计性质（局部连接、权值共享、平移等变）；padding 与 stride 及输出尺寸公式；池化；特征图与层级特征；完整前向流水线与通道。**为什么值得学**：三个项目里的视觉模型全是 CNN 或其直系亲属；实验三（11 月 23 日）训练的就是卷积网络。**学完你能**：手算一个 $3 \times 3$ 卷积；算出任意卷积层的输出尺寸；说清参数到底省在哪；看懂 CNN 结构图与特征图。

## 5.1　全连接看图：一场参数灾难

先弄清为什么不能把图片直接塞进第 2 讲的 MLP。全连接层要求输入是一维向量，所以图片得先“拉直”。一张 $28 \times 28$ 的手写数字拉直成 784 维向量，接 1000 个隐藏神经元——约 78 万个权重，还没开始认字就已经不少。那真实世界的照片呢？算给读者看：

**表 5.1**　全连接看图的参数账：图片越大越离谱

| 输入图 | 拉直后的维度 | 全连接到 1000 个隐单元 | 评论 |
|---|---|---|---|
| 手写数字 $28 \times 28$（灰度） | 784 | 约 78 万 | 勉强能接受 |
| ImageNet 常用 $224 \times 224$（RGB） | 约 15 万 | 约 1.5 亿 | 开始失控 |
| 手机照片 $1000 \times 1000$（RGB） | 300 万 | 30 亿（float32 约 12 GB） | 灾难 |

表格第三行值得掰开现场算一遍。一张 100 万像素的照片（约 $1000 \times 1000$，手机厂商喊了十年的“百万像素”就是这个数），假设只接 1000 个隐藏神经元，光这一层的参数就是 $10^6 \times 10^3 = 10^9$——**十亿个**。什么体感？按 float32 存就是 4 GB，一张照片还没开始认，先吃掉半部手机的运行内存；训练时还得为每个参数另备一份梯度（第 3 讲），内存再翻一倍。而这，只是一张普通手机照、一层的账。

参数多还只是账面问题，更要命的是**拉直把位置信息丢了**。784 个数排成一队之后，“谁挨着谁”就没了：图里的笔画往右挪一个像素，784 维向量几乎每个分量都变了——人眼看来还是同一个数字，网络看来却是一行从没见过的全新输入。更别提第一层每个权重只认识“队伍里的第几个数”，根本不知道第 17 号和第 18 号在图上是左右邻居。空间结构，全连接层一出生就扔了。

于是只剩一条笨路：把一切都背下来。猫挪半步、镜头拉远、头歪 15 度，像素排布立刻全变，逐位对齐的模板马上失效；想靠蛮力兜住，就得为**每一种位置、每一种大小、每一种角度**各备一套模板——组合爆炸，永无出头之日。可视觉识别的基本要求恰恰相反，叫**形变不变性**：怎么挪、怎么缩、怎么转，都得认得是猫。

账算到这儿，病根齐了三条：**参数太多**（每个位置一套班子），**经验不共享**（左上角学出的“猫耳侦探”，到了右上角完全作废，得从头再学一遍——可猫耳不管长在哪个角落，长相是一样的呀），**位置与形变脆弱**（模板焊死在坐标和尺度上，物体一动就失明）。CNN 的全部设计，就是对症下药：**只看小块**（局部连接）、**模板复用**（权值共享），外加一剂**只留最强响应**（池化，5.4 节）兜住小幅位移。

## 5.2　卷积 = 滑动窗口做点积

**卷积核**（kernel，也叫 filter，滤波器）是一块小模板，比如 $3\times3$ 的九个数。用法看下图：把它按在图片左上角，模板盖住的几个像素与模板上对应的几个数**对应相乘再求和**，得到输出图左上角的一个数；然后向右滑一格，重复；一行滑完换下一行，直到扫遍全图。输出图叫**特征图**（feature map）——它标记的是“这块模板在图里最像的地点”。

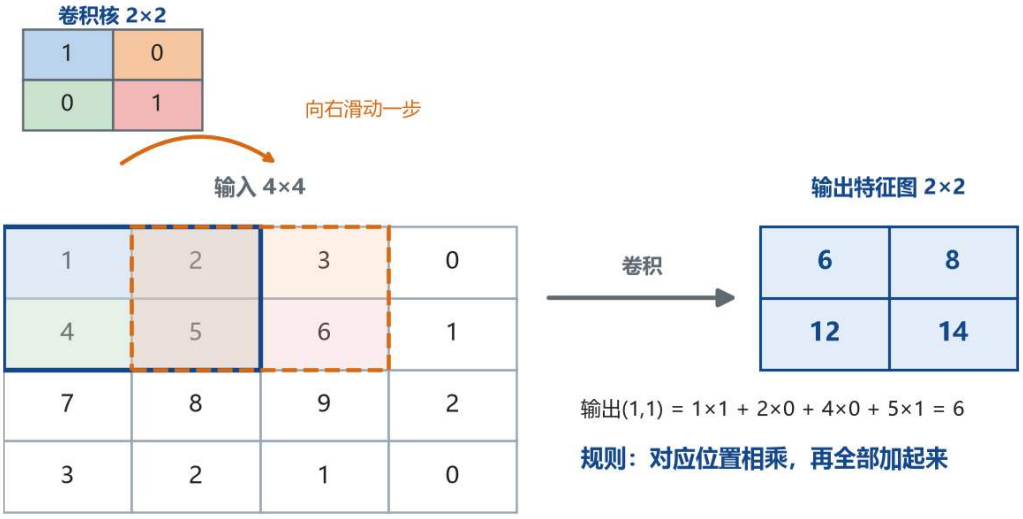



**图 5.1**　卷积的滑动窗口计算：$2\times2$ 的核在 $4\times4$ 输入上每次盖住 4 个像素，对应相乘再求和得到输出一个数；图中演示向右滑动两步

慢着——“对应相乘再求和”？这不就是开课导引里的**点积**（第 0 讲）、感知器里的**加权和**（第 1 讲）吗？完全正确，一分不差。**卷积核就是一个小感知器：只用九个权重，对图片的一小块打“相似度分”**——模板与那块图像越长越像，得分越高。卷积没有任何新数学，它就是点积的滑动应用；当年 LeCun 的突破，与其说是发明了新运算，不如说是发现了“把感知器按在图上滑”这件朴素的事恰好就是视觉需要的。

这个设计自带三条性质，而且条条都能在 1968 年的猫实验里找到原型：

- **局部连接**（local connectivity）：每个输出只依赖输入的一小块，这一小块叫**感受野**（receptive field）——视皮层的“边缘侦探”也只盯一小块视野。
- **权值共享**（weight sharing）：同一块模板扫遍全图，参数量与图像大小无关——猫耳模板在左上角、右上角通用，经验全图共享。

- **平移等变**（translation equivariance）：图里的物体挪个位置，特征图上的响应也挪同样的位置——猫在左在右，猫耳模板都照样点亮，只是亮的格子不同。

**知识补齐：数学课的“卷积”与工程口的“卷积”不是一回事**

严格数学定义的卷积，要把核先旋转 180 度再滑动（“翻面再扫”）；深度学习框架里叫 conv 的函数，实际做的是不翻转的**互相关**（cross-correlation）。为什么敢偷懒？因为核是**学出来的**——网络想要什么样的模板，它自己会学成需要的样子，翻不翻转根本不影响表达能力。名词之争留给数学课，工程口径记住一条即可：本书与主流框架一致，“卷积”都指不翻转的滑窗点积。

手机修图你天天在用卷积，只是不知道它叫这个名字。

**动手时间：手机修图里的卷积核——手算一个角**

打开手机修图 APP 的“锐化”“浮雕”“模糊”，它们就是三块现成的卷积核在图上滑动。锐化核：

$$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix} \qquad \text{模糊核：} \quad \frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix} \qquad \text{浮雕核：} \quad \begin{bmatrix} -2 & -1 & 0 \\ -1 & 1 & 1 \\ 0 & 1 & 2 \end{bmatrix}$$

取一张 4×4 的小图，像素随便填 0–9 的整数；把锐化核对准左上角的 $3\times 3$ 区域，手算输出左上角那个数。自检：若该区域恰好是

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix},$$

则输出 $=0\cdot 1+(-1)\cdot 2+0\cdot 3+(-1)\cdot 4+5\cdot 5+(-1)\cdot 6+0\cdot 7+(-1)\cdot 8+0\cdot 9=5$。想想看：这块核（中心 5、四邻 $-1$）为什么能锐化？区域平坦时输出约等于原值；中心比四邻亮时，输出被进一步拉高——**中心与四邻的差被放大，对比度拉开，就是“锐”**。模糊核则相反，把九个数平均成一团。课后题会让你把这三块核真的跑在一张照片上。

## 5.3　padding 与 stride：滑窗的两个旋钮

滑动窗口上有两个官方旋钮。第一个：**padding**（填充），在图片外圈补一圈 0。动机有二：其一，$3\times 3$ 的核每扫一遍，输出图就比输入缩一圈（$4\times 4$ 进去 $2\times 2$ 出来），网络几十层叠下来，图会缩没了——补 0 圈可以让输出尺寸保持不变；其二，让边角像素也有机会被模板盖到，不至于边缘信息没资格进特征图。第二个：**stride**（步幅），每次滑几格：stride $=1$ 逐格滑；stride $=2$ 跳着滑，输出尺寸约减半，相当于自带一步压缩。

两个旋钮拧完，输出尺寸由一条公式管死（建议背下来，实验三搭网络时天天用）：

$$O=\left\lfloor \frac{N+2p-f}{s} \right\rfloor +1, \tag{5.1}$$

*人话：$N$ 是输入边长，$f$ 是核边长，$p$ 是每侧补 $0$ 的圈数，$s$ 是步幅，$O$ 是输出边长；除不尽的余数直接丢弃（$\lfloor\cdot\rfloor$ 是向下取整）。*

三个算例检验理解：其一，$32\times 32$ 输入、$5\times 5$ 核、$p=0$、$s=1$：$O=(32-5)/1+1=28$——这正是下一讲 LeNet-5 的第一步；其二，同上但 $p=2$：$O=(32+4-5)/1+1=32$——补了两圈 0，尺寸保住了；其三，$224\times 224$ 输入、$3\times 3$ 核、$p=1$、$s=2$：$O=\lfloor(224+2-3)/2\rfloor+1=112$——尺寸减半，是现代网络常用的“一边卷一边缩”配置。下图把前两种情况画在了一起：

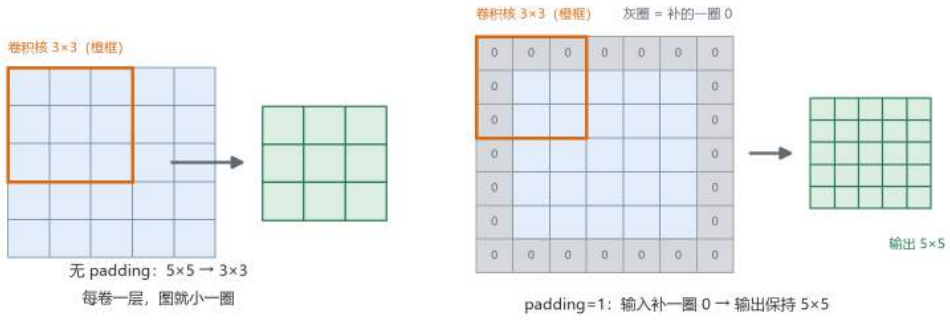



图 5.2　padding 与 stride 的效果：输入外圈补 0 后输出不再缩水；stride = 2 时窗口跳着滑，输出尺寸减半

## 5.4　池化：只留最强的响应

卷积之后常跟一步**池化**（pooling）：把特征图切成不重叠的小块，每块只留一个数。最常用的是 **max pooling**（最大池化）：$2 \times 2$ 区域取最大值，见下图。直觉是：模板在那一小块里“最像的地点”有一个就够了，具体压在左上角还是右下角不重要——**保留最强响应，丢弃精确位置**。顺带的收益：尺寸减半、计算减半，而且对小的位移不敏感（图案挪动一两个像素，最大值还是那一个）。

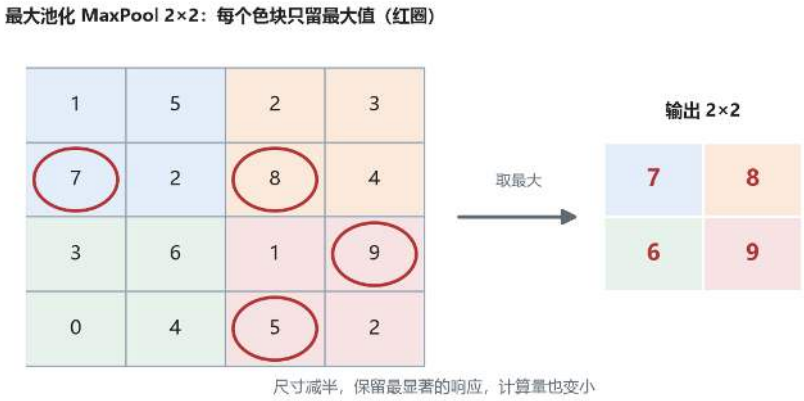



图 5.3　$2 \times 2$ 最大池化：$4 \times 4$ 特征图切成四块，每块取最大值得到 $2 \times 2$ 输出（彩色块标出每块的“冠军”）

另外两位池化成员一句话认识：**平均池化**（average pooling，取平均，手法更温和）；**全局平均池化**（global average pooling，GAP：整张特征图直接平均成一个数，现代分类网络常用它收尾，下一讲见）。注意池化自己**没有任何参数**——它是纯操作，不参与学习；网络要学的仍然只有卷积核和全连接的权重。

## 5.5　特征图与层级：从边缘到猫脸

一块模板扫一张图得一张特征图；多块模板并行扫，得到一叠特征图。层数加深，特征图的内容逐层升级：第一层检出边缘与斑点（正是 Hubel–Wiesel 的“简单细胞”干的事），中层拼出纹理与部件（条纹、转角、眼睛），深层直接响应整个物体（脸、车轮）。先看“一块模板长什么样”：下图里同一条斜边的输入，经过三个不同的核，得到三张“体检报告”——只在边缘经过的位置亮起，三张报告各亮各的。

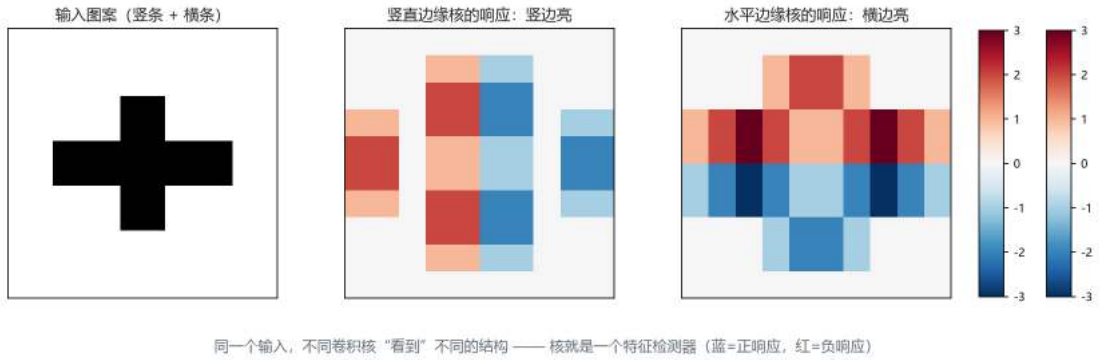



**图 5.4**　一条斜边（左）经过三个不同的卷积核得到三种响应图（中、右）：亮处即模板找到“同伙”的位置

关键问题是：每一层该放什么模板、层级该怎么长？答案：**没人设计，训练自己长出来的**——“认错东西的代价”经反向传播一层层压回去，网络自然把底层核逼成边缘侦探、把深层核逼成物体侦探。2014年 Zeiler 与 Fergus 的可视化工作第一次让人类看清了这套层级（下图，从左到右特征逐层抽象）：

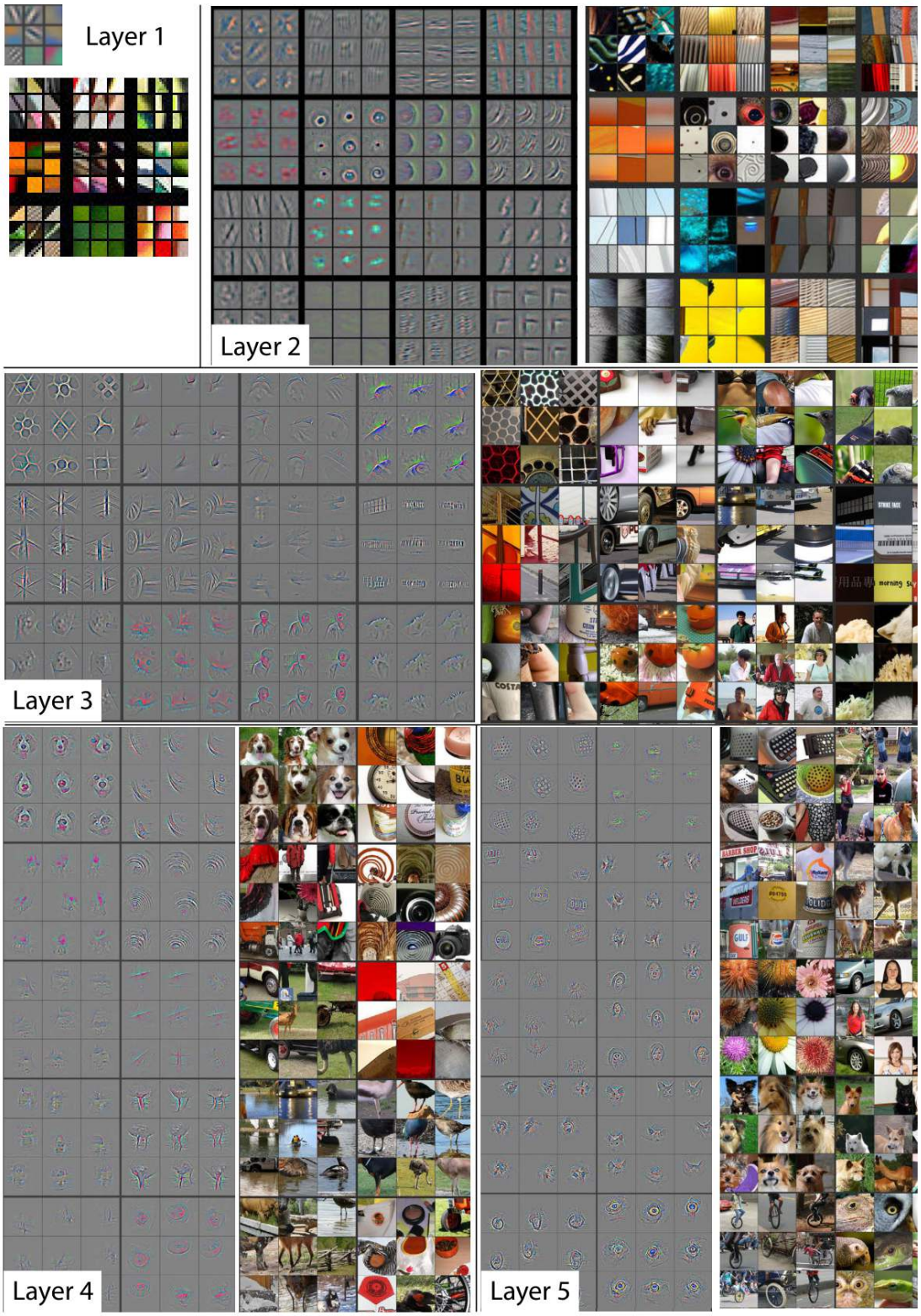



**图 5.5**　CNN 各层学到的特征层级可视化：底层是边缘与颜色块，中层拼出纹理与部件，高层已是物体部件与整体轮廓

图片来源：Zeiler & Fergus, "Visualizing and Understanding Convolutional Networks", 2014

这张图也回答了一个哲学问题：CNN 凭什么“像”视觉系统？因为它把 1968 年猫实验的发现——从边缘侦探到复杂细胞逐级组装——变成了可训练的数学结构。**生物学出蓝图，反向传播来施工，**

本讲开头那句话到这里落地。

## 5.6　完整前向流水线：从像素到类别得分

把本讲积木拼起来。一张图进入 CNN 的完整旅程见下图：图像先进“卷积—池化”单元（卷积提特征、池化缩尺寸），反复若干轮，特征越提越抽象、图越缩越小；然后**拉平**（flatten，把多张小特征图摊成一维向量），送进全连接层汇总判断；最后输出各类别得分。下一讲你会看到：得分经 softmax 变概率、与标签算交叉熵损失、再由反向传播逐层回传，把每个卷积核里的数修一点。

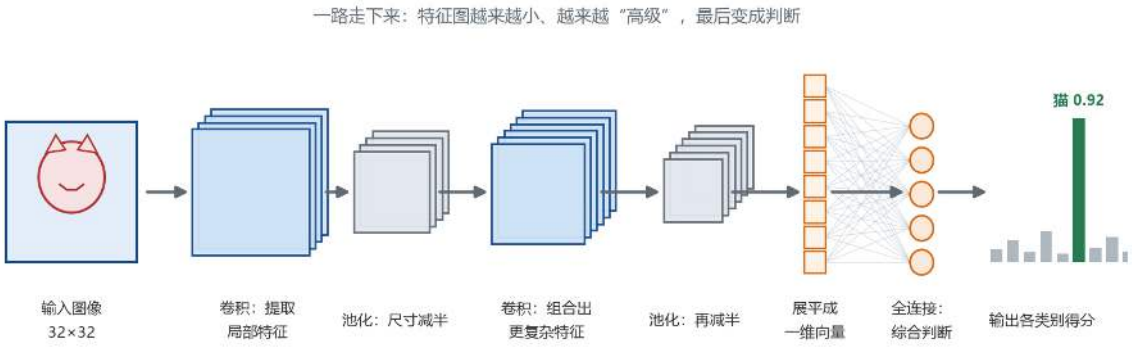



**图 5.6**　CNN 完整前向流水线：图像经“卷积—池化”反复提特征、缩尺寸，最后拉平送入全连接层输出类别得分

还有最后一个必懂名词：**通道**（channel）。其实前面已经埋好了——彩色图本身就是三叠灰度图（红、绿、蓝各一叠），这叫 3 通道输入；而每层卷积输出的一叠特征图（每个核一条），又是下一层的“多通道输入”。于是卷积核其实是个三维小方块：**边长 × 边长 × 通道数**——第一层的核是 $3 \times 3 \times 3$（吃彩色输入），深层的核可能是 $3 \times 3 \times 64$。通道就是“特征的平行宇宙”：每个通道一种模板视角，层层叠加越叠越厚。

**生活化类比：张量——给数字们盖的高楼**

向量是一排数（一维），矩阵是一张数表（二维），**张量**（tensor）是更高维的“数字楼”（三维及以上）：一张彩色图是 $3 \times H \times W$ 的三维张量——三层的楼，每层楼一个颜色通道；一批 32 张图打包就是四维张量 $32 \times 3 \times H \times W$。PyTorch 里一切数据都是张量；回忆第 3 讲的向量化——张量整体运算，正是“一行代码顶一百行循环”的舞台。

最后算总账，看参数到底省了几个数量级。同样处理 $1000 \times 1000$ 的 RGB 照片、提取 100 种模板：

**表 5.2**　参数总账：全连接 vs 卷积（同样 100 种模板的活儿）

| 方案 | 参数量 | 备注 |
|---|---|---|
| 全连接到 1000 个隐单元 | 约 30 亿 | 每个权重只服务一个位置，位置一动就作废 |
| 卷积：100 个 $3 \times 3 \times 3$ 的核 | $100 \times 27 + 100 = 2800$ | 每块模板全图通用，参数与图像大小无关 |

30 亿对 2800——**百万倍的差距**，而且图像越大，省得越多。这就是 CNN 能把整个视觉世界装进一张显卡的记账原理。

**梯度怎么流过卷积与池化？（人话版）**

下一讲的主角反向传播，要从损失出发把“该怪谁”的梯度逐层传回去修参数。它在卷积和池化上怎么走？给四条直觉就够用，完整推导超出本讲义——思路就是链式法则加这四条。① **max 池化：梯度只认最大值**。前向时谁最大，反向时这股梯度就全额判给谁，同块其他位置一律为 0——像个路由器：前向顺手记住“冠军住在几号房”，反向只敲冠军的门，其余位置收到的信号是“与你无关”。②**平均池化：梯度均分**。前向把四个数平均成一个数，反向就把上游来的 1 份梯度平分成 4 份发回去，人人有份，童叟无欺。③**卷积核的梯度：还是点积**。某个核该修多少？把输入特征图与传回来的误差图（上游梯度铺成的图）对应相乘再求和——与前向卷积同款算法，还是点积思想。误差图在哪儿响应大，说明那个位置对这个核的得分影响大，核就朝那儿修。④**“求导时的卷积”不翻转，辨析一句话**。求卷积核自己的梯度（更新参数用的那份）时，做的就是不翻转的滑窗点积——跟前向一模一样；“翻转的核”只出现在把误差继续传给上一层的路上，那是下一讲推导里的细节。给初学者的结论：更新参数的那份梯度，仍是熟悉的点积，不必被“翻转”二字吓住。

**前沿速递：CNN 还是 ViT？2026 年的版图**

2020 年底，Google 的 Vision Transformer（ViT）把第 11 讲的注意力机制搬来看图：把图切成 $16 \times 16$ 的小块当“单词”，用自注意力让每块直接看到全图——不滑窗、不假设“东西长在隔壁”。在海量数据上 ViT 打败了 CNN，一时间“CNN 过时论”甚嚣尘上。但 2024–2026 年的共识冷静得多：小数据、端侧、低算力场景，CNN 仍占优——卷积自带的“局部 + 共享”先验让它在小数据上省样本、在手机上跑得快；超大数据与多模态场景，ViT 更强。两条路线也在合流：ConvNeXt 把 CNN 调校到能与 ViT 掰手腕的精度，各大视觉基础模型则多以 ViT 为骨干。你们项目的 FoundationStereo 就是“ViT 骨干 + 卷积细节修正”的混合体——所以本讲和第 11 讲，一讲都不能翘。

**项目连线：从“人写的模板”到“学出来的模板”**

**六目拼接**：传统方案的 ORB 特征（第 一 章），本质是手工设计的卷积核加点积匹配——FAST 找角点近似一块固定的边缘核，BRIEF 描述子近似一套预先写死的比较规则，全是第 1 讲点积思想的手工版；而九个参数头里的 `feature_uncertainty` 与 descriptor 头，输入正是 $32 \times 32$ 的图像块、网络主体正是几层卷积——把人写的模板升级成学出来的模板，是第 7 讲 CNN 三十年演化的微缩版在你组里上演。**水下双目**：立体匹配的底层逻辑，就是滑动窗口在左右两幅图之间找“长得一样”的地方——逐块打相似度分构成代价体（cost volume），而打分器本身就是卷积网络；FoundationStereo 内部同样在滑窗比对，只是模板全部预训练而来（第 二 章）。**数据基座**：B 类表格恢复里“扫描件图片与字段标签对上号”的环节用的也是视觉模型——小数据表格场景恰恰是本讲说的“CNN 仍占优”的地盘，你选型时的判断力就从这一讲来（第 三 章）。

本讲课后题，是把卷积核从手机修图 APP 里“请”出来，亲手滑一遍——顺便体感一下为什么工程实现绝不许用 Python 双重循环。五要素如下。

**第 5 讲课后编程题（用 AI 实现）**

**题目**：指挥 AI 辅助，写一个“卷积核体验器”：读入任意一张图片（自己手机拍的即可，先转灰度），对它分别应用三块 $3\times3$ 的核——锐化（中心 5、四邻 $-1$）、模糊（九个 1/9）、边缘检测（Laplace 核：中心 $-4$、上下左右为 1）——**滑动窗口必须自己写**：双重 for 循环逐位置“对应相乘再求和”，不许调用 `cv2.filter2D`、`scipy.signal.convolve2d`、`torch.nn.Conv2d` 这类现成卷积 API 来做这三个核。用 matplotlib 把“原图 + 三张结果”画成 $2\times2$ 四宫格，保存为 png。

**交付物**：(1) 代码文件 `conv_toy.py`；(2) 四宫格 png 一张；(3) 50 字以上的说明：三块核各把图片改成了什么样、为什么（用“点积 / 相似度”的语言说）；(4) 一段 50 字以上的“我改了 AI 代码里的什么、为什么”。

**提示词建议**（可直接抄）：“我是零基础学生，正在学卷积。请用 Python + numpy 写一个卷积核体验器：(1) 读入一张本地图片并转灰度（没有现成图就用 skimage 的示例图）；(2) 手写双重循环实现 $3\times3$ 卷积滑动窗口（对应相乘再求和），不许用 cv2.filter2D 等现成卷积函数；(3) 分别应用锐化核 [[0,-1,0],[-1,5,-1],[0,-1,0]]、模糊核（九个 1/9）、边缘检测 Laplace 核 [[0,1,0],[1,-4,1],[0,1,0]]；(4) 结果裁剪到 0–255 后，把原图与三张结果画成 $2\times2$ 四宫格，保存 png；(5) 先在一小块 $4\times4$ 数据上打印中间结果，帮我核对滑动计算对不对；(6) 每行中文注释，并用一句话回答：为什么循环版对大图很慢？工程上该怎么加速？”

**验收线**：四宫格生成且三张结果肉眼可辨（锐化更“脆”、模糊更“糊”、边缘图上轮廓发亮）；能对着图讲清“某个亮点 = 模板与该处图像块的点积得分高”；能当场回答两问——“$3\times3$ 核滑 $1000\times1000$ 的图要点多少次积”（约 100 万个位置、每个位置一次 9 项的点积）与“为什么工程实现不用 Python 循环”（第 3 讲向量化）；至少一处自己的修改（换 Sobel 核、给彩色图逐通道做、补 0 圈保尺寸都算）。**代码是 AI 写的没关系，讲不清楚就是没做。**

**加分项**（任选其一）：(1) 给边缘检测加上一圈 0 的 padding 重跑，对比四角的变化（呼应 5.3 节）；(2) 把 stride 改成 2 重跑，验证输出尺寸公式；(3) 把三块核串成流水线（先模糊再边缘），模拟“上一层的输出当下层的输入”，看看边缘图发生了什么。

课后阅读：教材《深度学习模型与算法基础》第 50–70 页，与本讲内容对应；看书上的卷积图示时，用本讲的“滑动点积”语言把每张图自己讲一遍。

## 5.7　从这一讲到 2026

本讲学的是卷积的“语法”。2026 年的生产现场里，这门语法被写在哪些地方？四个事实，比任何论战都更有说服力。

**事实一：实时与边缘检测的工业默认，仍是卷积。**2026 年 1 月正式发布的 YOLO26 依旧以卷积架构为主干，用“免 NMS 的一对一端到端检测头”把后处理环节直接取消，最小的 YOLO26n 在 CPU 上的 ONNX 推理较上一代最高快 43%，五档模型在 COCO 上做到 40.9–57.5 mAP——车间质检、无人机、车载视觉这些“快字当头”的场景，卷积仍是默认答案。

**事实二：医疗现场的分工表。**2024 年发表于 Journal of Medical Systems 的系统综述（汇总 36 项研究）给出冷静结论：ViT 要靠大规模预训练才见优势；在小数据、低分辨率的场景里，DenseNet-121 这类经典 CNN 依然能打平。“大模型用 ViT、小数据用 CNN”的分工，在医院里同样成立。

**事实三：卷积从主角退成了关键零件。**Stable Diffusion 3（2024 年 3 月）把去噪主力从卷积 U-Net 换成了 Transformer 家族的 MMDiT，但把像素压进潜空间、生成完再解压回来的 VAE 编解码器**仍是卷积网络**——你用文生图得到的每一张图，进出潜空间都得过一遍卷积。这位“压缩器”的真面目，

第 12 讲见。

**事实四：卷积与注意力已经住进同一栋楼。**MobileNetV4（ECCV 2024）把标准倒残差块、ConvNeXt块、前馈层与注意力统一进同一个搜索空间，让算法按目标芯片自动挑组合；它自带的 Mobile MQA 注意力更是专为移动加速器设计——早已不是“卷积还是注意力”的二选一，而是“每一层放哪个”的填空题。

所以 2026 年的正确提问不再是“卷积还是 Transformer”，而是**“卷积应该放在哪一层”**。学完本讲，再修完第 11 讲的注意力，你就同时握有回答这个问题的两支笔。

**卷积 / 卷积核（convolution / kernel）**

小模板在图上滑动、逐块做点积的运算；卷积核是要学习的参数。工程实现实为互相关，不翻转。

**特征图（feature map）**

一块核扫完整张图得到的“相似度地图”，亮处即模板命中之处。

**局部连接与感受野（local connectivity / receptive field）**

每个输出只依赖输入的一小块；那一小块叫感受野。

**权值共享（weight sharing）**

同一块模板扫遍全图，参数量与图像大小无关，经验全图复用。

**平移等变（translation equivariance）**

输入平移，特征图上的响应同步平移；物体在哪，响应就挪到哪。

**padding（填充）**

输入外圈补 0：既保住输出尺寸，又让边缘像素有机会被模板盖到。

**stride（步幅）**

滑窗每次跳几格；stride $= 2$ 输出约减半。输出尺寸公式：$O = \lfloor (N + 2p - f)/s \rfloor + 1$。

**池化（pooling）**

特征图切块留一个数（常用 $2 \times 2$ 取最大），压缩尺寸并换取位移不敏感性；自身没有参数。

**通道（channel）**

特征的平行宇宙：彩色输入 3 通道，每层卷积输出多通道；卷积核因此是三维小方块。

**张量（tensor）**

向量与矩阵的高维推广；一张 RGB 图是三维张量，一批图是四维张量，深度学习框架里的数据通通是张量。

# 第 6 讲　卷积神经网络的训练与计算模式

1998年前后，如果你在美国把一张支票存进银行，很可能它已经被一台机器“读过”了——不是柜员，而是一个卷积神经网络。贝尔实验室的燕乐纯（Yann LeCun）团队在 1989 年就教会了神经网络认手写数字，随后这套代号 LeNet 的系统一路迭代，到 20 世纪 90 年代末，**高峰期每天替全美银行读取 10%–20% 的支票**。这是卷积神经网络第一次大规模商用——比“深度学习”这个词火起来早了十几年。上一讲你已经看懂了卷积和池化怎么算（第 5 讲），也就是知道了这台机器**怎么往前走**；本讲要回答的是后半截问题：它**怎么学会**——训练循环怎么转、数据从哪来、Batch Normalization 为什么几乎无处不在，以及除了准确率，我们还能用什么指标评估它。学完本讲，你就可以亲手把一个 CNN“养大”。

**本讲地图**

**讲什么**：MNIST与 CIFAR-10 两个经典数据集、LeNet-5 逐层精读、数据增强、训练循环的五个动作与 PyTorch 代码、批归一化（BN）、混淆矩阵与精确率/召回率、过拟合对策清单。**为什么值得学**：三个项目的所有训练任务——九个参数头、FoundationStereo 微调、真假样本分类器——转的都是本讲这一个“循环”。**学完你能**：读懂并修改一个 PyTorch 训练脚本；解释 `model.train()` 与 `model.eval()` 的差别；用混淆矩阵判断一个分类器到底靠不靠谱。**课后阅读**：许庆阳《深度学习模型与算法基础》第 71–85 页。

## 6.1　练手房：MNIST 与 CIFAR-10

学开车不必先上高速，学训练也一样。深度学习为“第一次训练”准备了两个经典的“练手房”，小到你的笔记本几分钟就能跑完一次，又大到足以暴露训练的全部问题。

第一个是 **MNIST**（Modified National Institute of Standards and Technology）：7 万张 $28 \times 28$ 的灰度手写数字图片（0 到 9 共 10 类），其中 6 万张做训练、1 万张做测试。它诞生于 LeNet 团队之手——从美国人口普查局和高中生的原始表单中扫描、整理而来，1998 年起成为整个领域默认的入门考题。深度学习圈有句话：**手写数字识别是这个领域的 Hello World**——你以后每换一个新框架、新显卡，第一件事多半仍是在 MNIST 上跑一把。下图是它长什么样：每一张图就是 7 万分之一个样本，人类几乎张张能认，机器当年却要苦练许久。

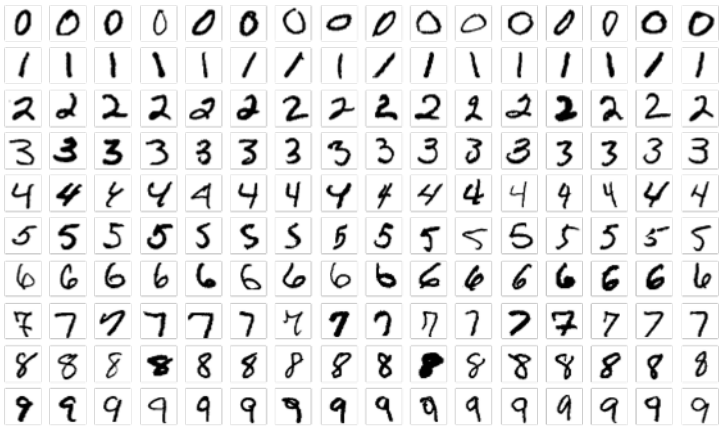


**图 6.1**　MNIST 手写数字样本网格：每张为 28 × 28 的灰度图，同一个人写的“1”千姿百态，歪斜、粗细、断笔都可能是机器要克服的困难

图片来源：LeCun MNIST 数据库 / 维基百科 MNIST database 词条

第二个是 **CIFAR-10**：6 万张 32 × 32 的彩色小图，10 个类别（飞机、汽车、鸟、猫、鹿、狗、蛙、马、船、卡车），5 万训练、1 万测试。它是 MNIST 的“进阶版”——类别多了、彩色了、背景复杂了，同一条卷积路线照用不误；本篇第 7 讲和第 8 讲的课后题都会用到它。

**扫盲：一张图片在机器眼里就是 784 个数**

你眼里“一个手写的 8”，在机器眼里只是 28 × 28 = 784 个 0 到 255 之间的整数：背景接近 0（黑），笔画处接近 255（白）。所谓“训练一个分类器”，就是找一个函数：吃进这 784 个数，吐出 10 个类别各自的得分。这个视角会贯穿全书——第 1 讲的感知器吃 4 个数（鸢尾花的两长两宽），这里的 CNN 吃 784 个数，第 7 讲的 ImageNet 网络吃一百多万个（224 × 224 × 3）。**图像即矩阵**，是每一个初学者必须先建立的直觉。

## 6.2　LeNet-5 精读：第一栋“经典公寓”的户型图

把 LeNet-5 摊开在图纸上，就是下面这栋七层小楼。它不深也不宽，却奠定了此后二十六年所有图像网络的基本户型。

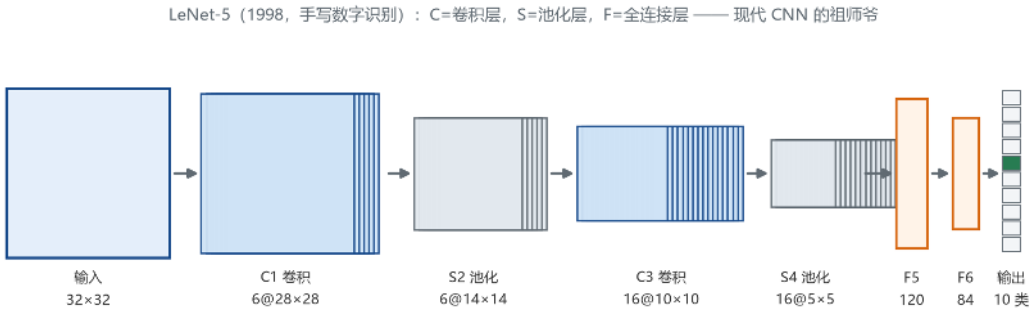



**图 6.2**　图题：LeNet-5 结构横向流程：32 × 32 输入经卷积（6 个 5 × 5 卷积核）、池化、卷积（16 个卷积核）、池化，再经全连接 120、84，输出 10 类得分

逐层数一遍它的“砖头”（下表同时给出每层参数量——请动手验算，公式在下面正文里）：

**表 6.1**　LeNet-5 逐层数字速查表（参数量为近似口径，总数约 6.1 万）

| 层 | 干什么 | 输出尺寸 | 参数量 | 一句话 |
|---|---|---|---|---|
| 输入 | 灰度图 | $32 \times 32$ | 0 | 原始支票局部照 |
| C1 | 卷积，6 个 $5 \times 5$ 核 | $6 \times 28 \times 28$ | 156 | 提取短线段等局部特征 |
| S2 | 平均池化 $2 \times 2$ | $6 \times 14 \times 14$ | 12 | 压缩，抗小幅位移 |
| C3 | 卷积，16 个 $5 \times 5$ 核 | $16 \times 10 \times 10$ | 1516 | 把线段拼成弯钩、圆弧 |
| S4 | 平均池化 $2 \times 2$ | $16 \times 5 \times 5$ | 32 | 再压缩 |
| C5 | 全连接 | 120 | 48120 | 把 400 个数揉成整体判断 |
| F6 | 全连接 | 84 | 10164 | 进一步提炼 |
| 输出 | 全连接 | 10 | 850 | 10 个数字各自的得分 |

参数量怎么算？以 C1 为例：6 个卷积核，每个核是 $5 \times 5 = 25$ 个权重再加 1 个偏置，共 $6 \times (25+1) = 156$ 个。请注意这个数字的分量：假如不用卷积、直接把 $32 \times 32 = 1024$ 个像素拉平接一层 100 个神经元的全连接，光这一层就要 $1024 \times 100 + 100$ 超过 10 万个参数——而参数一多，就需要更多数据、更多算力，还更容易过拟合（第 3 讲）。**卷积的本质优势在 1998 年就已经体现：用“局部连接 + 权重共享”花小钱办大事**。

再把两个“数值算例”做实。**其一，卷积核个数 = 输出特征图张数**：C1 有 6 个核，就输出 6 张 $28 \times 28$ 的特征图，每张对应一个模板的“相似度地图”——表里输出尺寸“$6 \times 28 \times 28$”开头的 6 就是这么来的；同理 C3 的 16 个核出 16 张图。看懂任何 CNN 结构图，先数卷积核个数。**其二，S2 平均池化为什么有 12 个参数？**平均池化本身是零参数的纯操作（第 5 讲），但 LeNet 给每个通道配了一对小旋钮：取完平均后先乘一个可学习的缩放系数、再加一个可学习的偏置，$6 \times 2 = 12$——相当于允许每张图“平均完再调调对比度和亮度”。

C3 这一层还藏着一个更精妙的手工设计：它有 16 个卷积核，按常规每个核都该看 S2 的全部 6 张特征图（$16 \times 6 = 96$ 条连接）；LeCun 却只保留了 60 条——**每张 C3 特征图只挑 S2 的几张做输入**，组合是精心挑选的（见下表）。省参数是一方面，更妙的是逼着 16 张特征图各有分工：前 6 张各看相邻 3 个通道，中间 6 张各看相邻 4 个，后 3 张故意“跳着挑”打破对称，最后 1 张六张全看。这种“手工艺接线”如今已被更通用的 $1 \times 1$ 卷积取代（第 7 讲），但“通道不必全连、可以省着连”的思想留了下来。

**表 6.2**　C3 的 16 张特征图各选 S2 的哪几张做输入（编号从 0 起，依 LeCun 1998 原表）：不是全连接，是精心挑选的组合

| C3特征图 | 选用的 S2 通道 | C3 特征图 | 选用的 S2 通道 |
|---|---|---|---|
| 0 | 0, 1, 2 | 8 | 2, 3, 4, 5 |
| 1 | 1, 2, 3 | 9 | 3, 4, 5, 0 |
| 2 | 2, 3, 4 | 10 | 4, 5, 0, 1 |
| 3 | 3, 4, 5 | 11 | 5, 0, 1, 2 |
| 4 | 4, 5, 0 | 12 | 0, 2, 3 |
| 5 | 5, 0, 1 | 13 | 1, 3, 4 |
| 6 | 0, 1, 2, 3 | 14 | 2, 4, 5 |
| 7 | 1, 2, 3, 4 | 15 | 全部 6 张 |

更重要的是户型本身。看图中从左到右的三段式：**卷积–池化堆叠两轮负责“提取特征”，全连接收尾负责“拍板分类”**。这个模式有个学名上的通俗版——特征提取器加分类头（backbone + head）。它强到什么程度？你将在第 7 讲看到，AlexNet、VGG 直到 ResNet 换了一茬又一茬，户型仍是它；

你项目里六目拼接的九个“参数头”，每个也都是“小特征提取 + 小分类头”的同款结构。1998 年定下的骨架，2026 年还在服役。

**动手时间：纸笔算 LeNet-5**

不看上表，自己算一遍（答案附后）。**(1)** C1 的输出尺寸：$32\times32$ 的图上滑 $5\times5$ 的核、步距 1、不加 padding，输出宽高各是多少？（提示：第 5 讲的公式 $(W-K+2P)/S+1$。）**(2)** C1 的参数量。**(3)** S2 池化后输出多大？**(4)** C3 在 $14\times14$ 上滑 $5\times5$ 的核，输出多大？**(5) 想一想：**若把 C1 的核从 $5\times5$ 换成 $3\times3$ 且不加 padding，输出变成多大？对网络“看到”的信息意味着什么？——答案：(1) 28；(2) 156；(3) $6\times14\times14$；(4) 10；(5) 30。核越小“看得越近”，要看得更远就得堆更多层——这正是第 7 讲 VGG 的出发点。

## 6.3　数据增强：白捡的“新数据”

MNIST 只有 6 万张训练图，够用吗？LeNet 时代的答案是不太够——网络记性好得很，几十个 epoch 之后它就开始**背答案**：训练集上几乎全对，没见过的字却认错，这正是第 3 讲讲过的过拟合。买新数据太贵，于是有了穷人的一百万：**数据增强**（data augmentation）——把已有的图随机变个样，标签却不变，等于白捡新数据。




**图 6.3**　图题：数据增强九宫格：中心为原图，四周是随机旋转、平移、翻转、加噪、亮度变化等增强结果——样子变了，标签仍然是同一个数字

为什么“变了样、标签不变”是成立的？因为**这些随机变换不触及类别的本质**：一个“8”略微歪一点、淡一点、抖一点，它还是 8；一张猫的照片左右镜像，它还是猫。增强没有创造新信息，但它逼着网络**见多识广**——不再依赖“训练集里的 8 都是端正的”这种投机取巧的捷径，而必须学会扭歪的 8、断笔的 8、墨淡的 8 的共性。用 PyTorch 实现只要一行：

```
from torchvision import transforms as T
# 训练时的变换：随机翻转 + 随机小角度旋转 + 转张量
train_tf = T.Compose([T.RandomHorizontalFlip(),
                      T.RandomRotation(10), T.ToTensor()])
# 测试时的变换：什么都不随机，原样评估
test_tf = T.Compose([T.ToTensor()])
```

注意两个细节。**第一**，随机性只属于训练：测试时必须关掉增强、用原图评估，否则成绩无法复

现。**第二**，“标签不变”依赖任务语义，不能想当然：手写数字**旋转 180 度会把 6 变成 9**，上下翻转则把它变得不像任何数字——对 MNIST 这两类变换就不合法；而字母“b”和“d”恰好互为镜像，做 OCR 时连水平翻转都不能用。动手选增强前，先问自己一句：变换之后，**我**还认得出它是几吗？

## 6.4　训练循环：五个动作的摩天轮

万事俱备：有数据（6.1节）、有结构（6.2 节）、有防过拟合手段（6.3 节），现在把第 2 讲的反向传播和第 3 讲的梯度下降装进一个循环。现代深度学习训练就是一个永远在转的摩天轮，每转一圈执行五个动作：

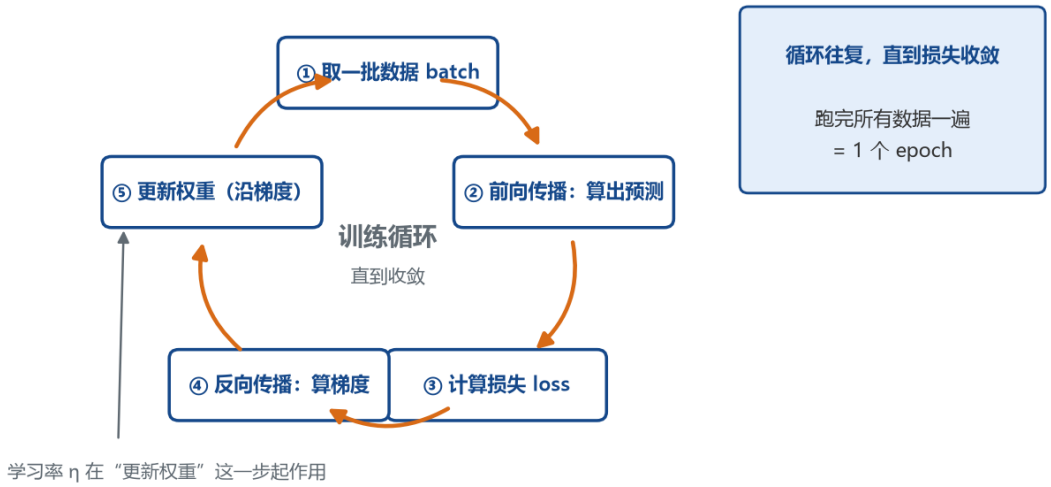



**图 6.4**　图题：训练循环流程环：取 batch → 前向 → 算 loss → 反向 → 更新参数，五步首尾相接循环往复，直到收敛或达到轮数上限

**取 batch**：不一次吃完 6 万张图，而是每次抓一小撮（如 64 张）——一小撮叫一个**批**（batch），整批前向加反向算一步；全部训练数据过完一遍叫一个**轮**（epoch），6 万张除以批大小 64，一个 epoch 约 938 步。**前向**：图像过网络，得到 10 个类别的得分。**算 loss**：用交叉熵损失把得分与标准答案的差距折成一个数（第 3 讲的损失函数家族在分类任务上的主力成员）。**反向**：`loss.backward()` 一行触发反向传播，算出每个参数该背多大责任（梯度）。**更新**：优化器让每个参数沿负梯度挪一小步。转，不停转，直到 loss 不再下降。PyTorch 版全文如下——第 2 讲那个手写循环，翻译成现代框架长这样，骨架一毫米没变：

```
import torch, torch.nn.functional as F

model = LeNetLike()                              # 6.2节那样的卷积网络
opt = torch.optim.Adam(model.parameters(), lr=1e-3)

model.train()                                    # 切到训练模式（见下方知识盒）
for epoch in range(5):                           # 整个训练集过5遍
    for x, y in train_loader:                    # 1) 取一批(batch)数据
        opt.zero_grad()                          #    清掉上一批的旧梯度
        pred = model(x)                          # 2) 前向：得到10类得分
        loss = F.cross_entropy(pred, y)          # 3) 算loss：差距多大
        loss.backward()                          # 4) 反向：算出所有梯度
```

```
        opt.step()                              # 5) 更新：沿负梯度挪一步
    print(f"epoch {epoch}, loss {loss.item():.4f}")
```

五步中只有第 1、2、3、5 行会因任务而变，第 4、5 行的“反向加更新”雷打不动——日后你在三个项目里训练任何模型，看到的都是这个摩天轮。

**知识补齐：训练模式与推理模式——train/eval 一行之差**

PyTorch 的网络对象自带一个开关：`model.train()` 与 `model.eval()`。它不改变参数，改变的是某些层的行为：**Dropout** 训练时随机“关掉”约一半神经元（逼网络不依赖个别神经元，防过拟合），推理时必须全部打开；**Batch Normalization**（下一节主角）训练时用当前批次的均值方差，推理时改用训练期间累计的滑动平均——因为推理时常常一张一张来，没有“批”可统计。忘记切模式是新手第一 bug：训练脚本忘了 `model.train()`，BN 统计错乱、loss 抖成心电图；评估忘了 `model.eval()`，随机关掉的神经元让准确率莫名掉一截，而曲线看起来一切正常。记住：**训练前 train()，评估前 eval()**，一行搞定。

## 6.5　批归一化 BN：给每一层的数据“扶正”

训练一个深网络为什么难？有一半的难处来自一个隐蔽的连锁反应：**每一层都在适应上一层的变化**。第 2 层的输入是第 1 层的输出——训练中第 1 层的参数每个 batch 都在更新，于是第 2 层脚下的大地一直在漂移；它刚适应好，上游又变了，只能跟着重学。层数一深，层层漂移叠加，训练就变得又慢又抖，学习率还不敢开大。2015 年 Ioffe 和 Szegedy 给出的解法朴素而漂亮：**既然每层的输入分布会漂，那就每一层都把它固定住**。这就是**批归一化**（Batch Normalization，简称 BN）。

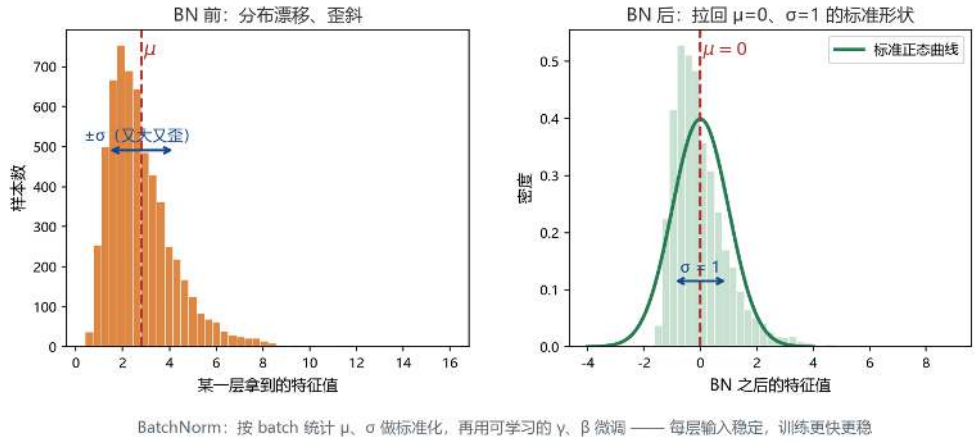



**图 6.5**　图题：BN 效果示意：同一批特征值归一化前（偏斜、范围杂乱）与归一化后（集中在 0 附近、方差稳定）的分布直方图对比

做法分两步。**第一步，拉回标准位**：对每个通道，用当前批次的统计量把数值强行归一到均值 0、方差 1——就是把歪的数据“扶正”（见上图前后对比）。**第二步，还网络一点自由**：再乘一个可学习的缩放系数 $\gamma$（伽马）、加一个可学习的平移 $\beta$（贝塔）——“扶正是原则，至于扶正后想再歪一点还是更正一点，网络自己决定”，万一原始分布本来就更好呢？在代码里，它的位置是约定俗成的三连：

```
self.net = nn.Sequential(
    nn.Conv2d(1, 16, 3),        # 1) 卷积：提取特征
    nn.BatchNorm2d(16),         # 2) BN：把分布扶正（卷积后、激活前）
    nn.ReLU(),                  # 3) 激活：非线性
```

```
)
```

为什么放在卷积后、激活前？因为激活函数各有“舒适区”（第 2 讲），BN 保证送进激活的数值分布稳定，避免大片数据落在饱和区。回报是立竿见影的三件套：**学习率可以开得更大、收敛显著更快、训练更稳**；顺带还有轻微的正则化效果。代价也有一条：训练时它依赖批统计，批太小（比如显存紧张只开到 4）统计就不准——工程上另有小批专用的变体，本课不展开。2015 年之后，几乎所有像样的卷积网络里都躺着一行 `BatchNorm2d`，包括你第 7 讲要见的主角们。

## 6.6　评估：准确率会骗人

训练跑完，测试集准确率 98.5%——可以交差了吗？先别急。**准确率（accuracy）只回答“总体判对的比例”，它会把所有错误一视同仁地藏起来**；要看清一个分类器，你需要一张它的“体检报告”。

**表 6.3**　MNIST 上的混淆矩阵示例（行＝真实类别，列＝预测类别；数字为一组真实训练的近似记录）。对角线是判对数量，其余全是不合格的“岔路”

| 真实 \ 预测 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|---|
| 0 | **956** | 0 | 1 | 0 | 0 | 2 | 4 | 1 | 9 | 7 |
| 1 | 0 | **1112** | 3 | 1 | 1 | 1 | 3 | 4 | 7 | 3 |
| 2 | 12 | 18 | **930** | 18 | 8 | 3 | 8 | 12 | 18 | 5 |
| 3 | 3 | 6 | 26 | **903** | 2 | 25 | 4 | 7 | 23 | 11 |
| 4 | 5 | 3 | 8 | 1 | **920** | 1 | 10 | 4 | 8 | 22 |
| 5 | 13 | 4 | 4 | 24 | 4 | **795** | 13 | 3 | 26 | 6 |
| 6 | 15 | 7 | 7 | 1 | 6 | 10 | **900** | 1 | 9 | 2 |
| 7 | 7 | 12 | 12 | 2 | 6 | 3 | 1 | **957** | 8 | 20 |
| 8 | 12 | 8 | 14 | 18 | 9 | 17 | 12 | 6 | **863** | 15 |
| 9 | 8 | 4 | 5 | 17 | 26 | 9 | 3 | 17 | 14 | **906** |

这张表叫**混淆矩阵**（confusion matrix）：对角线上是各类判对的数量，**对角线以外的亮斑就是“张冠李戴”发生地**。读它的乐趣在于发现规律：上表里最亮的两块误判是 4 认成 9（22 次）、9 认成 4（26 次）——它们只差一笔的闭合；3 与 5 互认（25 与 24 次）——下弧线写得平一点就是对方。人看一眼就服气：“确实像”。所以混淆矩阵不只报成绩，还**告诉你该去补什么样的数据**（多收集连笔的 4 和 9）。

准确率之外，最常用的两个指标是**精确率**与**召回率**。以“找出 4”为例：**精确率**＝模型报出的“4”里有多少真是 4——衡量**报出来的准不准**；**召回率**＝所有真 4 里被模型抓出来多少——衡量**该找的找到没**。两者天然互相牵制：过滤条件卡得越严，报得越准，但漏掉的越多。把它们调和平均一下，就是综合分 **F1**。

**扫盲：一张化验单读懂 TP / FP / FN / TN**

把模型想成化验科医生：真的有病/没病，和化验单报了有病/没病，交叉出四种结局。**TP**（真阳性）＝有病且查出来了；**FP**（假阳性）＝没病却误报——虚惊一场；**FN**（假阴性）＝有病却漏检——**最危险的一种**；**TN**（真阴性）＝没病且没报。于是：精确率 $= \mathrm{TP}/(\mathrm{TP}+\mathrm{FP})$，“报告单的可信度”；召回率 $= \mathrm{TP}/(\mathrm{TP}+\mathrm{FN})$，“漏检率的反面”。在锚链检查这种安全场景（第 二 章），一次 FN 就是漏掉一条缺陷链环——所以水下项目的评测组盯召回率比盯准确率更紧。

最后是一个能救命的提醒。**类别极不平衡时，准确率会撒弥天大谎**：产线质检里 99% 的图是良品、1% 是缺陷，一个把所有图都判“良品”的懒模型，准确率高达 99%——召回率却是 0%，一件缺陷都抓不到，在产线上一文不值。记住这条反例，你就永远不会再被“99% 准确率”单独忽悠：**先看**

**混淆矩阵，再谈准确率**。这正是你若担任项目评测组（第 一 章六目组 5、第 二 章水下组 4）时的日常职业。

## 6.7　过拟合对策清单

第 3 讲那张训练 loss 一路下降、验证 loss 先降后升的双曲线图，从今往后你几乎每周都会重逢：它一出现，就是在喊“过拟合了”。对策是成熟套路，列成清单逐条点名：

**表 6.4**　过拟合对策清单：按“性价比”排序，前三条几乎零成本

| 手段 | 一句话原理 | PyTorch一行 | 成本 |
|---|---|---|---|
| 更多数据 | 见得多了就不会背 | ——（最难的一条） | 高 |
| 数据增强 | 白捡新数据（6.3 节） | `T.RandomRotation(10)` | 低 |
| 早停 | 验证 loss 回升就停 | 盯曲线手动停即可 | 低 |
| L2 正则 | 惩罚过大的权重 | `weight_decay=1e-4` | 低 |
| Dropout | 随机关神经元防抱团 | `nn.Dropout(0.5)` | 低 |
| 批归一化 | 顺带轻微正则 | `nn.BatchNorm2d(64)` | 低 |

使用顺序也有讲究，老手的打法是：**先刻意把网络在小数据上过拟合（证明它有能力学），再逐条上对策把过拟合压下去**。如果连过拟合都过拟合不起来，说明网络太小或代码有 bug，加正则只会雪上加霜。这份清单请存进你的工具箱——实验三（11 月 23 日）训练曲线一旦翘尾，第一反应就该是翻它。

**前沿速递：2026 年，“从零训练”已经是一件稀罕事**

本讲教会你的“拿数据、定结构、转循环”从零养大一个 CNN，在 2024–2026 年的工业界反而成了**稀罕做法**。真实流程几乎都是**预训练–微调范式**（pretrain–finetune）：先在超大通用数据集上训练一个“底座”模型（这步通常由大厂完成并开源），你只需要在底座上用你自己的小数据微调最后几层——用 `timm` 或 HuggingFace 生态，十分钟就能得到一个像样的图像分类器，数据少到每类几十张也行。**从零训练如今主要发生在两类地方：课堂教学（比如本讲）和前沿研究**。这条范式的时间线值得记一笔：第 7 讲的 ImageNet 预训练是它的起点，2021 年起的 CLIP、DINOv2 把“底座”从卷积换成 Transformer，而你在水下项目里“微调 FoundationStereo”正是这一范式的现场直播。所以本讲的价值不在“复刻 1998”，而在让你**看懂整个范式最底层的那台发动机**——微调时转的，仍是本讲这五个动作。

**项目连线：第 6 讲 × 三个项目——同一个摩天轮，三种坐法**

**实验三（11 月 23 日）是“模型训练实践”专场，三个项目的训练全是本讲循环的实例。六目拼接**（第 一 章）：九个参数头各自在标注数据上用本讲循环训练，产出“头名字`.onnx` + 同名`metrics.json`”一对文件，拷入 `build/Release/models/` 生效——`metrics.json` 就是该头的“体检报告”，**逐头指标不劣于规则初值才允许替换**，否则系统自动回退保护。**水下双目**（第 二 章）：训练组的日常是微调 FoundationStereo 时盯着 loss/EPE 曲线下降——EPE 是立体匹配版的“平均错多少像素”，到了实验三的训练现场你天天会见到它。**数据基座**（第 三 章）：真假样本训练分类器，教科书式的两类分类——本讲的混淆矩阵和精确率/召回率就是它的验收语言。最后划重点：**六目组 5 与水下组 4（评测组）的整个职业，就是 6.6 节**——别人训练，你们审判。

本讲课后题：亲手把一个 LeNet 风格的 CNN 养到 98% 以上，并体验“改一个旋钮、看一次变化”的实验感。五要素如下。

**第 6 讲课后编程题（用 AI 实现）**

**题目**：指挥 AI 辅助实现一个 LeNet 风格的卷积神经网络（两层卷积–池化堆叠 + 全连接收尾，参考 6.2 节户型），在 MNIST 上训练并达到**测试准确率 $\geq$ 98%**；然后**自己改动一个超参数**（学习率、batch size、是否加数据增强三选一），用完全相同的方式再跑一遍，把两次的训练曲线（每轮的 loss 与测试准确率）各画成一张图，对比差异并解释原因。

**交付物**：(1) 代码文件 `lenet_mnist.py`；(2) 两张训练曲线截图（基线一次、修改一次，图上标注最终准确率）；(3) 一段 50 字以上的说明，写清“我改了什么、预期与实际效果是否一致”。

**提示词建议**（可直接抄）：“我是零基础 Python 学生。请用 PyTorch 写一个 LeNet 风格的 CNN：两层‘卷积 +ReLU+ 池化’后接全连接 120、84、10，结构对齐教材第 6 讲。在 MNIST 上训练：batch size 64、Adam、学习率 0.001、5 个 epoch；每个 epoch 结束记录测试集 loss 和准确率并存成列表，最后用 matplotlib 画出‘epoch–准确率’曲线并打印最终准确率；每行代码加中文注释；最后告诉我：若准确率到不了 98%，学习率、batch size、数据增强分别该怎么调？”

**验收线**：代码在你自己的电脑上跑通、最终准确率 $\geq$ 98%；随机抽问时能不看资料讲清训练循环五步各自干了什么、`model.train()` 与 `model.eval()` 的差别；至少有一处自己的修改（换超参数、加 BN、加增强都算）。**代码是 AI 写的没关系，讲不清楚就是没做。**

**加分项**：用 6.6 节的方法打印你模型在测试集上的混淆矩阵，找出最容易混淆的一对数字，用一两句话分析原因（它们长得像在哪？），并尝试用一种针对性的数据增强把这个误判次数压下去。

## 6.8　从这一讲到 2026

本讲这个五动作摩天轮，在 2026 年的训练现场转成了什么样？三个事实，替你把“怎么训”的地图更新到今年。

**其一，数据增强已从“技巧”变成“默认配方”**。2024 年的一篇综述把 MixUp、CutMix、RandAugment 等数百种增强方法纳入统一框架，梳理成“训练流水线中的标准模块”，覆盖视觉乃至多模态任务，并维护着持续更新的开源项目——今天训练视觉模型，增强列表与优化器、学习率一样，是开工就要填的默认三件套之一。你课后题里随手加的 `T.RandomRotation(10)`，就是这套配方里的最小一员；比它更凶的 MixUp 干脆把两张图按比例混成一张、标签也按比例混合，逼网络学“类别之间的过渡带”。

**其二，“从零训练”进一步让位给“冻结底座 + 轻量探针”**。本讲前沿速递盒子说过预训练–微调范式，2025 年 8 月 Meta 的 DINOv3 又往前推了一步：官方声称其自监督特征**无需微调**——把整个主干冻住、只训一个线性探针或小检测头，就能在广泛任务上超越此前专门训练的最优模型，并发布了覆盖不同算力档位的模型套件。训练的活儿，正从“调千万个参数”变成“选底座、接探针”。医疗影像那篇系统综述也站在同一边：ViT 能否胜过 CNN，关键变量就是有没有预训练。

**其三，“怎么训”依旧是活跃战场**。2026 年 1 月发布的 YOLO26 带来自研的 MuSGD 优化器、渐进损失（Progressive Loss）与小目标感知标签分配（STAL），五档规模在 COCO 上做到 40.9–57.5 mAP——连最成熟的工业工具链都还在迭代训练配方。与此同时，一条工程底线也已定型：公开配方、固定随机种子、可复现的成绩单，不再是美德，而是纪律。

给 2026 年的你三句话：**把预训练当地基，把增强当配方，把种子和日志当实验纪律**——而本讲的摩天轮，是这一切共同的地板。

**MNIST**

7 万张 $28 \times 28$ 灰度手写数字数据集（6 万训练 + 1 万测试）；深度学习的 Hello World。

**CIFAR-10**

6 万张 $32 \times 32$ 彩色小图、10 类；MNIST 的进阶版，第 7、8 讲课后题的战场。

**批（batch）**

一次前向/反向共用的一小撮样本；批大小是最先要定的训练旋钮之一。

**轮（epoch）**

全部训练数据完整过一遍；训练通常是若干个 epoch 的循环。

**数据增强（data augmentation）**

对样本施加不改变标签的随机变换（旋转、平移、翻转、噪声等），等效于免费扩充数据。

**批归一化（Batch Normalization，BN）**

在卷积后、激活前把每通道数值按批次统计扶正到均值 0 方差 1，再用可学习的 $\gamma, \beta$ 微调；换更快收敛与更稳训练。

**训练模式/推理模式**

`model.train()` 与 `model.eval()` 切换的行为开关：Dropout 与 BN 在两种模式下行为不同。

**混淆矩阵（confusion matrix）**

行为真实类、列为预测类的矩阵；对角线判对，亮斑暴露易混类别对。

**精确率（precision）**

模型报出的正例中真正例的比例——“报出来的准不准”。

**召回率（recall）**

全部真实正例中被模型找出的比例——“该找的找到没”；安全场景的头号指标。

**F1 分数**

精确率与召回率的调和平均，两者的“一票综合”。

**早停（early stopping）**

验证集 loss 回升即停止训练，是过拟合清单里成本最低的一条。

# 第 7 讲　卷积神经网络的演化：加宽、加深与残差

2012年秋天，ImageNet 大规模视觉识别竞赛（ILSVRC）结果公布，排行榜让全场沉默了一瞬：一支几乎没人听说过的队伍，把 top-5 错误率从上一年的 25.8% 一刀砍到 15.3%，比第二名的 26.2% 整整低了近 11 个百分点——在往届每年只挪动一两个百分点的赛场上，这不是进步，是降维打击。冠军队伍是多伦多大学辛顿（Geoffrey Hinton）组的三名成员，训练设备是**两块打游戏用的 GTX 580 显卡**（他们不好意思申报“游戏显卡”，在论文里含糊地写了“GTX 580”）。此后十年，这条“加深、加宽、加残差”的军备竞赛一路把错误率压过人类水平，也把深度学习从冷宫一路抬进了你口袋里的手机。本讲就沿着这条时间线走一遍：每一代网络解决了什么瓶颈、又留下了什么新瓶颈——这是理解你项目里那些 backbone 的必经之路。

本讲地图

**讲什么：**ImageNet竞赛与错误率曲线、AlexNet 的三大法宝、VGG 的 3×3 极简主义、GoogLeNet 的多尺度并行与 1×1 降维、深层网络的“退化”难题、残差连接革命、设计经验法则。**为什么值得学：**六目端到端路线的 backbone 是 ResNet 家族，FoundationStereo 的 ViT 主干里也全是残差连接——你每天都在用 2012–2015 年的遗产。**学完你能：**说出五代网络的接力关系；解释“两个 3×3 抵一个 5×5”的算术；讲清残差为什么让百层网络能训练。**课后阅读：**许庆阳《深度学习模型与算法基础》第 86–100 页。

## 7.1　ImageNet：军备竞赛的“高考考场”

故事要从考场说起。2012 年之前的图像识别研究，各自为战：你用你的几百张图，我用我的几千张图，成绩无法比较，模型也无法拉开差距。斯坦福的李飞飞团队看中了这个死结，从 2007 年起发动一场笨功夫运动：借 WordNet 的概念树，靠众包平台雇了一百多个国家的标注员，一寸一寸给互联网图片打标签。到 2009 年，**ImageNet** 数据库初具规模——累计标注超过 1400 万张图片；从 2010 年起，每年用它举办 ILSVRC 竞赛，固定 1000 个类别，约 **140 万张**带标注图片（其中训练集 120 万张）。

为什么规模如此重要？因为第 6 讲你已经养过 CNN，知道它们的胃口：**参数越多，需要的燃料越多**。过去十几万参数的 LeNet 吃 6 万张 MNIST 尚可，想区分 1000 类猫猫狗狗花花世界，缺的从来不是模型点子——AlexNet 的卷积思想 LeCun 在 1989 年就有了——缺的是 140 万张图这样的燃料，和烧得起这些燃料的炉子。ImageNet 一举补齐了前一半，2012 年的显卡补齐了后一半，两头凑齐，深度学习点火成功。

考场的记分方式也要弄懂：**top-5 错误率**——模型对每张图给出 5 个候选类别，只要正确答案不在 5 个里才算错。给 5 次机会还算错，说明模型错得很离谱。下图是历年冠军的错误率曲线，注意三个刻痕：2012 年悬崖（AlexNet）、2014 年过半山腰（GoogLeNet 6.7%）、2015 年 **3.57%，首次低于人类水平（约 5%）**——此后考题对人类反而太难了，2017 年办完最后一届后竞赛功成身退。

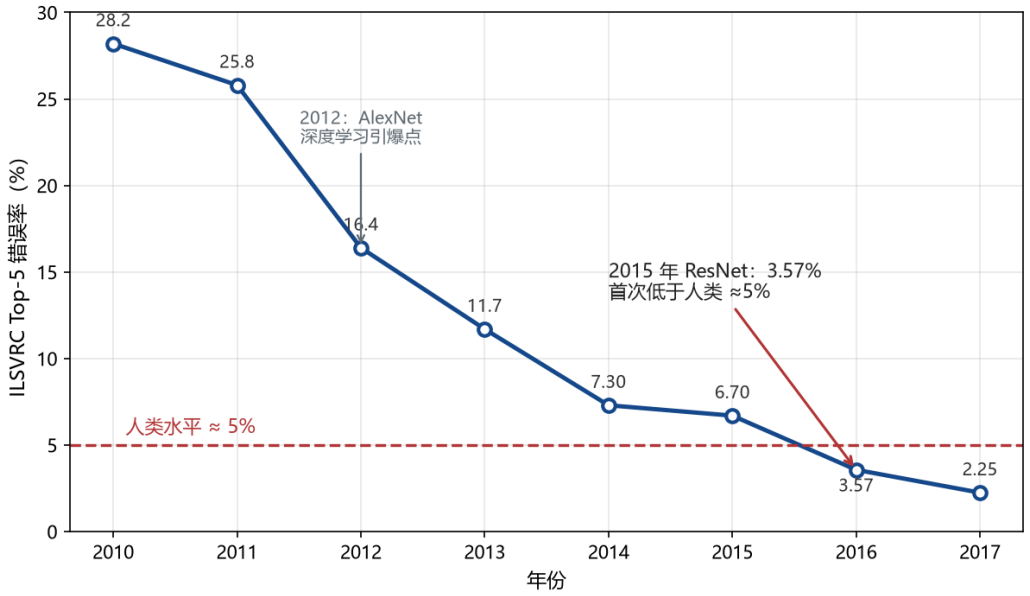



**图 7.1**　图题：ILSVRC 2010–2017 年冠军队伍 top-5 错误率逐年下降折线：2012 年 AlexNet 掀开悬崖，2015 年残差网络 3.57% 首次越过人类水平线（约 5%）

## 7.2　AlexNet：一鸣惊人的三件法宝

冠军网络后来得名 **AlexNet**（一作 Krizhevsky 的名字叫 Alex）。它的结构照抄 LeNet 的户型——卷积池化堆叠提取特征、全连接收尾——只是整栋楼放大了：**8 层**（5 层卷积 + 3 层全连接），6000 万参数，是 LeNet-5 的一千倍；输入也从 $32 \times 32$ 的小图换成 $224 \times 224$ 的彩照。结构上并无革命，**革命在训练配方**。三件法宝至今仍在你的代码里：

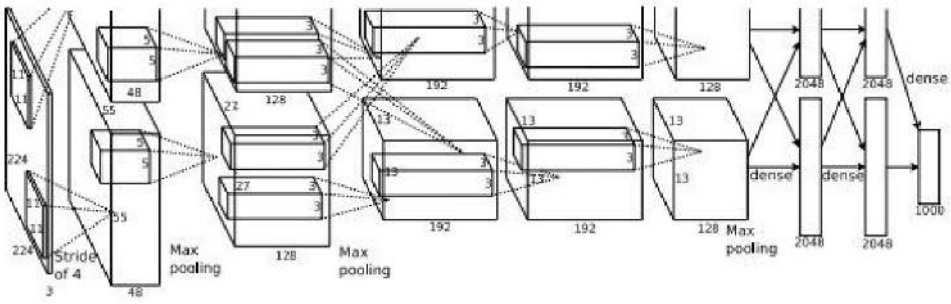



**图 7.2**　AlexNet 网络结构经典图：上下两半对应两块 GPU 各管一半通道、仅在特定层互通；5 段卷积逐级提特征，3 层全连接收尾分类

图片来源：Krizhevsky et al., ImageNet Classification with Deep Convolutional Neural Networks, NeurIPS 2012

**法宝一：ReLU**。第 2 讲见过它：$f(z) = \max(0, z)$，比 sigmoid/tanh 不饱和、梯度不断流。AlexNet 论文实测，同样网络换用 ReLU 后收敛快数倍——没有它，6000 万参数在 2012 年的硬件上根本练不出来。**法宝二：Dropout**。训练时随机丢弃约一半神经元，逼网络不把身家押在任何单个神经元上（第 6 讲过拟合清单里的老朋友，就是它发家的地方）。**法宝三：GPU 训练**。卷积是海量的小乘加，与显卡的本职工作完美同构（见下方扫盲盒）；两块 GTX 580 把六天的训练压到可接受，也把“买显卡炼模型”变成了此后十年的行业仪式。三件法宝合起来的启示，比网络本身重要得多：**当数据与算力到位，老想法也能爆发出新世界**。

法宝之外，论文里还躺着三个如今已有“盖棺定论”的细节，正适合当历史注脚读。**一是 LRN（局部响应归一化，Local Response Normalization）**：让相邻通道的响应互相“别苗头”，谁周围太兴奋就压谁一头——2012 年它是涨分利器，五年后 BN（第 6 讲）登场，干同样的活更简单更好，LRN 从此退居故纸堆：连“怎么归一化”都是演化中的部件。**二是重叠池化（overlapping pooling）**：池化窗口 $3\times3$、步幅却只有 2——相邻窗口叠着滑，信息不至于一步丢太快，论文实测比标准的不重叠池化低约 0.4 个百分点的 top-1 错误率。**三是双 GPU 训练**：当年一块 GTX 580 只有 3 GB 显存，装不下 6000 万参数，于是两块卡各管一半神经元、仅在个别层交换数据——这本质是**模型并行**（一层拆两半）；今天更常用的是**数据并行**（每卡吃不同的批、各自算梯度再汇总）。注意时代背景：2012 年“并行训练”这个词本身还很新鲜，论文得专门画一张双卡结构图解释；如今你实验课上一行 `DataParallel`，就是这段历史的直系后代。

**扫盲：为什么偏偏是游戏显卡改变了 AI**

显卡（GPU）的本职是每秒在几百万个像素上做同一套乘加运算——把每个点的光照、贴图算出来。而卷积恰好就是“用同一个小核对几百万个位置做同样的乘加”。**一个天生并行、重复、规整的计算，撞上一个天生为它而生的硬件**。一块 RTX 4080 的 FP32 算力约 48 TFLOPS（每秒 48 万亿次浮点运算），相当于几万台 1998 年的 PC 同时开动。你项目六目组在 RTX 4080 上训练端到端模型，用的正是这笔红利。

## 7.3　VGG：极简主义——只堆 3×3

AlexNet 之后，各家开始“堆料”：核越用越大、层越加越多，结构花样百出。2014 年，牛津大学 VGG 组（Visual Geometry Group）的 Simonyan 与 Zisserman 反其道而行，交出一份极简主义宣言：**把所有花活全部删掉，只用最小的 $3\times3$ 卷积和 $2\times2$ 池化，像砌砖一样整齐地往上堆**，深度 11 到 19 层。这套毫无惊喜的“砖墙”拿到当年亚军（7.3%），却因为结构干净、易于改造，成了之后数年工程界的默认 backbone，也成了你今天运行 `torchvision.models.vgg16` 的由来。

它的关键洞察是一个算术题，需要先补一个概念——**感受野**（receptive field）：输出上某个神经元“看得见”输入图多大的范围（详细定义见下方知识盒）。两个 $3\times3$ 卷积层串起来，第二层的神经元第一层看 $3\times3$、第二层再看第一层的 $3\times3$，合计正好覆盖 $5\times5$——**两层 3×3 抵得上一层 5×5 的视野**。那么为什么不直接用 5×5？算算参数（设通道数都是 $C$）：

$$5\times5:\ 25C^2 \qquad \text{versus} \qquad 3\times3\times2\text{层}:\ 2\times9C^2=18C^2. \tag{7.1}$$

**人话翻译**：同样的视野，两层 3×3 比 5×5 少 28% 的参数，还白赚一次 ReLU 非线性（两次弯折总比一次表达力强）。三层 3×3 抵一层 7×7，参数从 $49C^2$ 降到 $27C^2$，赚得更多。**用小核堆深度，比用大核更省、更强**——这一条算术直接把“深度”变成了值得军备的标的。

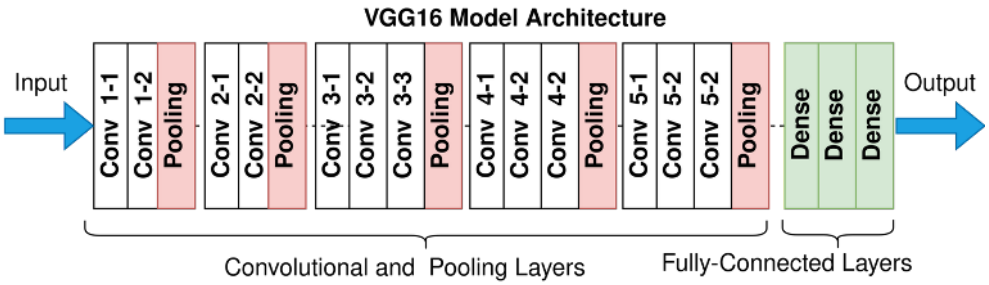



**图 7.3**　VGG16 结构图：整面“砖墙”全部由 $3\times3$ 卷积与 $2\times2$ 最大池化交替砌成，五段卷积逐级加厚通道，末端三层数万个单元的全连接收尾

图片来源：Simonyan & Zisserman, Very Deep Convolutional Networks (VGG), ICLR 2015

下图把这五代网络的深度摆在一起，你能看到军备竞赛的坡度。不过 VGG 也留下两个包袱：一是照这个砌法再往深处堆（20 层以上）就越来越难练（为什么？7.5 节揭晓）；二是那 1.38 亿参数、500 多 MB 的体格——末端三层全连接吞掉了绝大多数参数，妥妥的“虚胖”，这正是第 8 讲轻量化的第一动力。

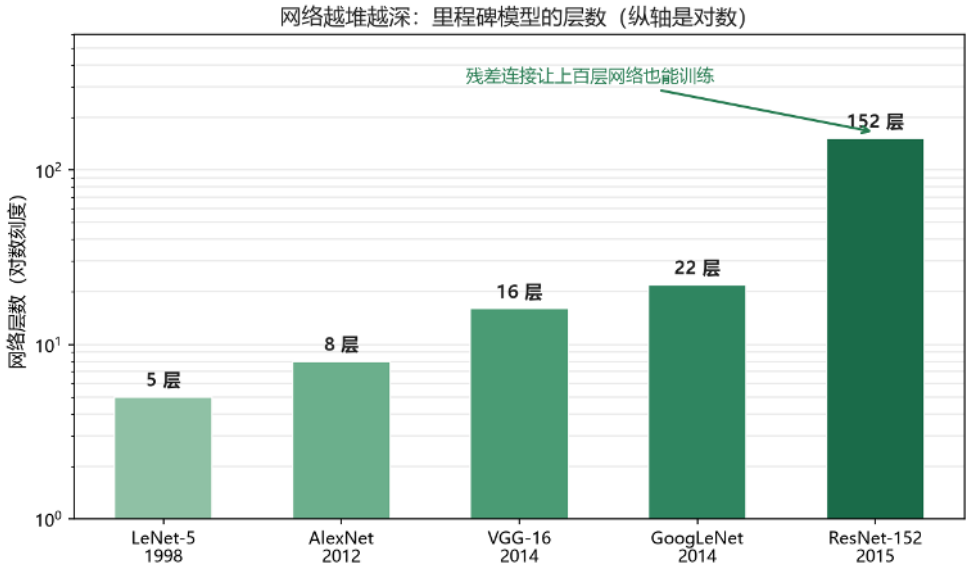



**图 7.4**　图题：网络深度里程碑条形图：LeNet-5（5 层卷积为主）、AlexNet 8 层、VGG16、GoogLeNet 22 层、ResNet 152 层——军备竞赛的“坡度”一目了然

**知识补齐：感受野——神经元的一亩三分地**

把卷积网络想成层层上报的情报系统：底层神经元只盯输入图上 $3\times3$ 的一小块（它的感受野），中层把底层的 $3\times3$ 情报再汇总成自己的 $3\times3$，于是中层的感受野是 $(3+3-1)\times(3+3-1)=5\times5$。每加深一层，感受野按“$k$ + 上层的 $-1$”向外扩一圈。感受野越大，神经元越能综合全局信息做判断——识别一只猫需要看到耳朵、胡须和身子的组合，这就要求深层的感受野足够覆盖整个物体。深度之所以有价值，一半原因就是它用廉价的层层叠加换来了大感受野；8.5 节还会介绍不加深也能扩感受野的“空洞卷积”。

## 7.4　GoogLeNet：多尺度并行与 1×1 省钱术

同年（2014年）的冠军是谷歌的 **GoogLeNet**（名字致敬 LeNet），它提供了另一条军备思路：**不拼纵深拼花样**。它的基本积木叫 **Inception 模块**（名字来自电影《盗梦空间》的梗“we need to go deeper”）：在一层里，$1\times1$、$3\times3$、$5\times5$ 三种卷积**并排**开火，池化也并行一路，四路结果在通道维度拼接——同一层同时提取小、中、大三种尺度的特征，**让网络自己学该信谁**，而不是由设计者拍板选一种核。

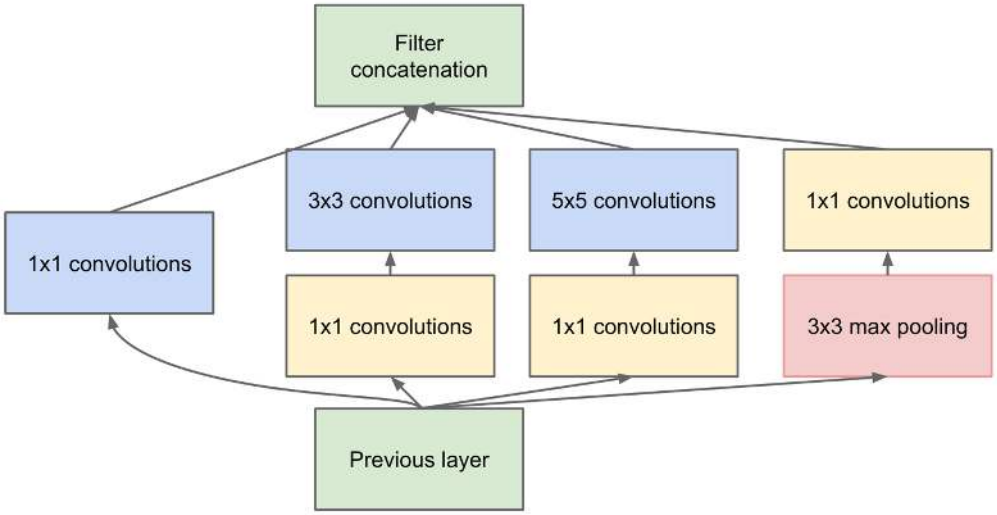



**图 7.5**　GoogLeNet / Inception 模块图：$1\times1$、$3\times3$、$5\times5$ 卷积与池化四路并行，各自接 $1\times1$ 瓶颈降维后拼接

图片来源：Szegedy et al., Going Deeper with Convolutions, CVPR 2015

但并行是有代价的：四路全开，计算量会爆炸，尤其 $5\times5$ 这种大核。救场的是图里那些前置的**$1\times1$ 卷积**——它不扫空间，只在**通道之间做加权组合**，可以理解为“通道维度的全连接”（每个像素位置上，把 256 个通道按权重混合出 64 个新通道）。把它当**瓶颈**（bottleneck）垫在 3×3、5×5 前面，先把 256 通道压到 64 再做大核卷积，该路的计算量立刻降到原来的三分之一以下。这招“$1\times1$ 降维省钱”此后无处不在，成为轻量化设计的标准动作。成绩单：GoogLeNet 深达 22 层，参数却只有约 680 万——不到 AlexNet 的十分之一，还更准。**“更深”与“更省”可以兼得**，前提是会设计。

Inception 家族并没有停在 2014 年。随后三年的迭代堪称一部小型演化史，一张表速览：

**表 7.1**　Inception v2–v4 演化速览：每一代只解决一个明确的瓶颈

| 版本 | 关键改动 | 一句话点评 |
|---|---|---|
| Inception-v2（2015） | 每层卷积后加 BN | 达到同等精度，训练步数降到约 1/14——BN |
| Inception-v3（2015） | 非对称卷积：$1\times3$ 接 $3\times1$ 替代 $3\times3$ | 同样视野、更少参数，还白赚一次非线性 |
| Inception-v4 / Inception-ResNet（2016） | 模块里接入残差捷径 | 与何恺明的革命合流，训练稳定性再上一层 |

表里两处值得多说一句。**非对称卷积**：把一个 $3\times3$ 拆成先竖后横的两小条（$1\times3$ 再 $3\times1$），参数从 9 个降到 6 个，表达能力反而更灵活——你在第 5 讲手算过卷积，可以自己验证“先竖后横”两步的视野确实仍是 $3\times3$。**辅助分类器**：GoogLeNet 在网络中部额外接了两个小分类头、把它们的损失一并算进总损失，本意是给深层“递个梯度”、防梯度消失；后来的复现表明它主要起一点正则作用，对最终精度的帮助很有限，到 v4 时期已经可以放心去掉——又一个“当年灵、后来被更好的招式取代”的注脚，与上一节的 LRN 遥相呼应。

## 7.5　深化的障碍：56 层为什么输给 20 层

按 VGG 的算术，深度这么划算，那一路加深不就天下无敌了？2015 年，时任微软亚洲研究院研究员的何恺明团队做了一个直白的实验：在 CIFAR-10 上训练普通（plain）卷积网络，20 层对 56 层，其他条件全部相同。结果违反直觉——下图（复现何恺明论文的风格）里，**56 层的训练误差反而比 20 层还高**。

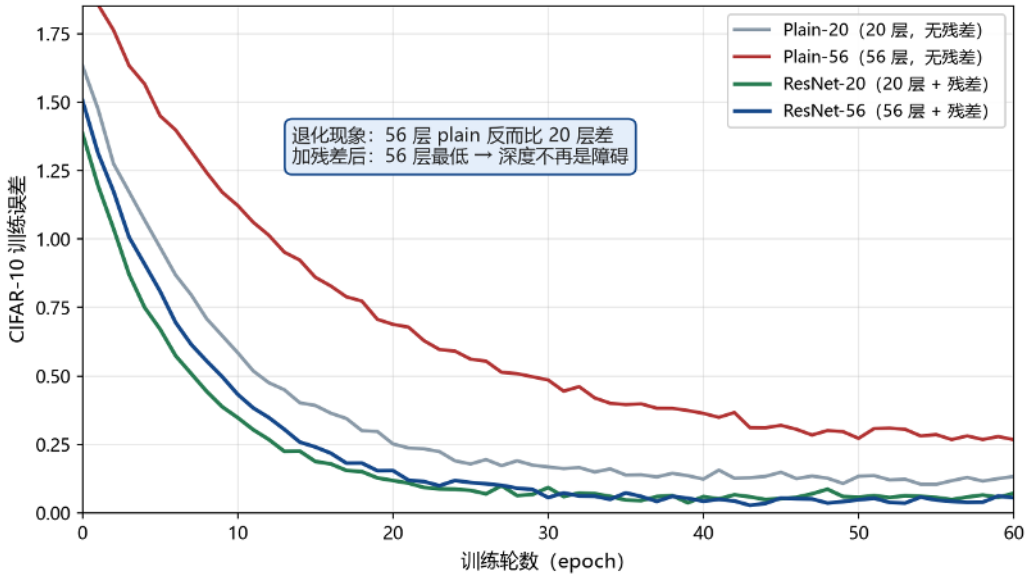



**图 7.6**　图题：复现 He 等人论文的四条曲线：普通网络 20 层与 56 层（56 层训练误差反而更高）、残差网络 20 层与 56 层（越深越好，且训练误差持续下降）

先把两个最常见的误判排除。**不是过拟合**：过拟合是训练误差低、测试误差高（第 3 讲），而这里是**训练误差本身**降不下去——网络连“背诵”都背不会。**也不是梯度消失**：彼时 ReLU 和归一化初始化（第 6 讲的 BN 也算同类）早已普及，20 层都能正常收敛。剩下的真相被称为**退化**（degradation）：**更深的理论上限不低，优化起来却更难**。想通这一点只需一个思想实验：假设 56 层网络的最优解就是“前 20 层照抄、后 36 层什么都不干”——只要后 36 层都学成恒等映射（输入原样输出），它就该和 20 层一样好。但优化器要费老大劲才能把普通卷积层调到“什么都不干”的精度；层层微小的偏差叠加，56 层反而学歪了。**深网络的问题不是能力不够，而是“维持现状”这件简单事对它来说太难。**

## 7.6　残差革命：给梯度修一条高速公路

何恺明的解法堪称优雅：**既然“什么都不干”难学，那就把它变成默认**。给网络加一条**恒等捷径**（identity shortcut）：把输入 $x$ 原封不动抄送到输出端，与卷积层的输出相加：

$$\mathbf{y} = \mathcal{F}(\mathbf{x}, \{W_i\}) + \mathbf{x}, \tag{7.2}$$

**人话翻译**：$\mathbf{x}$ 是这一块的输入，$\mathcal{F}$ 是两三层卷积想学的东西，输出不再是卷积层单独的答案，而是“$\mathcal{F}$ 对 $\mathbf{x}$ 的修改量加在 $\mathbf{x}$ 上”。换个角度看，卷积层学的内容从完整答案 $H(\mathbf{x})$ 变成了修改量 $\mathcal{F}(\mathbf{x}) = H(\mathbf{x}) - \mathbf{x}$——所以叫**残差**（residual，即“剩余的修正量”）。**学“修改量”永远比学“完整答案”容易**：如果这层根本不需要干活，把 $\mathcal{F}$ 压向 0 即可（权重本来就被正则往小拽，见第 6 讲清单），恒等映射白送；如果需要干活，也只需学从“现状”到“目标”的那一小段差。

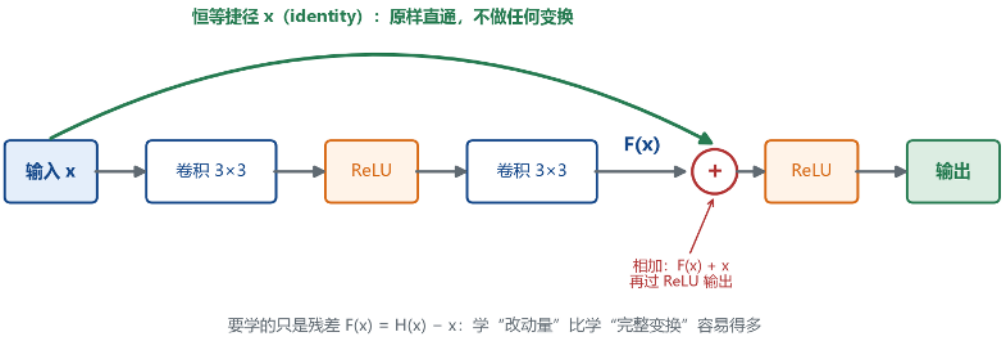



**图 7.7**　图题：残差块：输入 $x$ 兵分两路——主路经两层卷积学修改量 $F(x)$，捷径路原样直通，两路相加后再过激活；标注“学习目标从 $H(x)$ 换成 $F(x)=H(x)-x$”

下图是 ResNet 论文原图，左列的两个小积木就是上图的实际尺寸（两种厚薄规格，深网络用带 $1\times1$ 瓶颈的右款）。别小看这条抄近道的加法边，它同时解决了两件事：**前向**，“什么都不干”成为零成本默认值，百层网络退化问题消失；**反向**，求导时梯度从输出端沿捷径一路“原样 +1”地流回输入端，中间任何一层都拦不住它——相当于给梯度修了一条不设收费站的高速公路，再深的网络也不会在训练初期“断粮”。组装起来就是 **ResNet**：把第 6 讲的卷积块换成残差块，一路砌到 **152 层**仍然稳定训练——2015 年，ResNet-152 以 **3.57%** 的 top-5 错误率夺冠，首次越过人类水平线，这篇论文拿下 CVPR 2016 最佳论文奖，也是中国学者（何恺明，清华本科、港中文博士）在深度学习史上最响亮的一笔。记住这个结构，它被称为深度学习史上最伟大的一个“加法”。

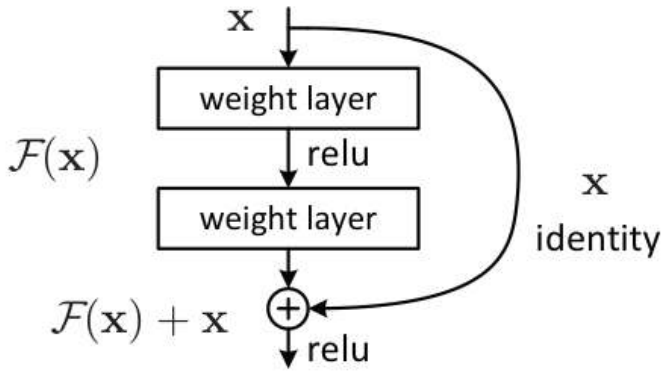



**图 7.8**　ResNet 论文原图：左侧两种残差块规格（普通款与 $1\times1$ 瓶颈款），右侧串联成 34 层网络与 VGG-19、普通 34 层的对比

图片来源：He et al., Deep Residual Learning for Image Recognition, CVPR 2016

## 7.7　设计经验法则：深度、宽度、分辨率的三角

军备竞赛打到 2015 年，兵法已经基本成型。把五代网络的成败浓缩成一张经验法则表，你日后自己选 backbone、改结构时照着想：

**表 7.2**　网络设计经验法则速查：想动哪一格，先看配什么药

| 想加什么 | 必须配的药 | 出处/代表 |
|---|---|---|
| 深度（层数） | 残差连接，否则退化 | ResNet（7.6节） |
| 宽度（通道） | 分组卷积省算力 | AlexNet 双卡 / ResNeXt |
| 感受野（看得远） | 空洞卷积或多加层 | DeepLab（第 8 讲） |
| 省算力 | $1\times1$ 瓶颈降维 | GoogLeNet（7.4 节） |
| 小参数量 | 小核堆深 + 压通道 | VGG 算术 / MobileNet（第 8 讲） |

最后一条全局观：**深度、宽度、分辨率是算力天平上的三个砝码**——输入图从 $224\times224$ 加到 $448\times448$，计算量变 4 倍；通道翻倍，计算量翻 4 倍；层数翻倍，计算量翻约 1 倍。三者同比例放大收益最好（2019 年的 EfficientNet 把这条直觉总结成了“复合缩放”公式）。**结构决定上限，训练技巧（第 6 讲）决定你能不能摸到上限**——两讲合起来，才是完整的“养网络”手艺。

**前沿速递：CNN 死了吗？——2024–2026 视觉骨干的真相**

2017 年 Transformer 进军视觉（ViT）后，“CNN 已死”的标题年年都有。2022 年 ConvNeXt（FAIR，《A ConvNet for the 2020s》）狠狠回击：把 ResNet-50 按 2020 年代的训练配方和结构技巧逐项现代化，性能反超同期的 Swin Transformer——**CNN 不是不行，只是修炼多年的功课被 ViT 一步学走了**。2024–2026 年的现实格局是：**搞通用基础模型，ViT 系当家**——CLIP 图文对齐、SAM 分割、DINOv2 自监督特征，全是 ViT 骨干；**上端侧、跑实时，CNN 仍是主力**——手机、车载、工业相机里的检测分割模型至今多属 CNN/MobileNet/YOLO 家族（第 8 讲）。而对本讲最重要的是：**ResNet 的思想已经统治一切**——每个 Transformer 块写的都是“$x+$ 注意力 $(x)$”，捷径相加原封不动地搬了过去（第 11 讲细看）。学 ResNet 不是学古董，是学所有深度网络的通用语法。

**项目连线：第 7 讲 × 三个项目——你的项目里全是这一讲的遗产**

**六目拼接**（第 一 章）：端到端路线 `ropstitch.onnx` 的 backbone 正是 ResNet 家族——组 7 在 RTX 4080 上用第 6 讲的循环训练它；组 8 接手做量化压缩。而九头路线的“**逐头指标不劣于规则初值才允许替换、否则自动回退**”验收线，就是恒等捷径的**系统级翻版**：新参数头是“修改量”，规则初值是“恒等捷径”——改动只在证明不劣于现状时才被接纳，和 $H(x)=x+F(x)$ 保底不劣于 $x$ 是同一个工程哲学。**水下双目**（第 二 章）：FoundationStereo 的 ViT 主干里，每个块都是“捷径 + 注意力”的残差结构——微调它，就是在 152 层那次革命的地基上盖楼。**数据基座**（第 三 章）：监考员、判官模型的骨干同样多为 ResNet/ViT 家族。一句话：**这一讲不是历史课，是你三个项目的零件说明书**。

军备竞赛的结论之一是“深就是好”，但你也该亲眼看一次**为什么深必须配残差**——课后题让两胞胎网络当场对决赛跑。五要素如下。

**第 7 讲课后编程题（用 AI 实现）**

**题目**：指挥 AI 辅助在 CIFAR-10 上做一场“有无残差”的对照实验：搭两个参数量接近的 6 层卷积网络——A 版为普通串接（plain），B 版在每两层卷积后加一条恒等捷径（shortcut）相加后再激活；两者用完全相同的训练设置（batch size 128、学习率 0.001、Adam、各训 20 个 epoch），把两条训练 loss 曲线与两条测试准确率曲线画在同一张图上对比，并写一段 50 字以上的结论：

残差版收敛是否更快、最终是否更高、差距大不大。

**交付物**：(1) 代码文件 `plain_vs_res.py`（内含两个网络定义与统一训练函数）；(2) 对比曲线截图（至少含训练 loss 与测试准确率两幅）；(3) 50 字以上的实验结论。

**提示词建议**（可直接抄）：“我是零基础 Python 学生。请用 PyTorch 在 CIFAR-10 上写两个 6 层卷积网络做对照：网络 A 为普通串接（卷积 +ReLU 堆 6 层，中间夹两次最大池化）；网络 B 参数量与 A 接近，但每两层卷积后加恒等捷径：输出 = ReLU(卷积输出 + 输入)，若通道数不一致用 1×1 卷积对齐。两者共用同一个训练函数：batch size 128、Adam、学习率 0.001、各训 20 个 epoch；每个 epoch 记录训练 loss 和测试准确率，最后用 matplotlib 把两条 loss 曲线画一张图、两条准确率曲线画一张图；每行加中文注释。”

**验收线**：两个网络都训完 20 个 epoch 且训练设置完全一致；能不看资料向同学解释“捷径相加为什么能让 56 层网络训练误差降下来”；至少有一处自己的修改（加深到 8 层、改学习率、换 batch size 都算）。**代码是 AI 写的没关系，讲不清楚就是没做。**

**加分项**：把两个网络都加深到 10 层再比一轮，观察“差距是否随深度拉大”；或在训练中记录每个 epoch 的梯度范数（提示：遍历 `model.parameters()` 取 `p.grad.norm()`），验证“捷径让梯度流得更通”。

## 7.8　从这一讲到 2026

“CNN 之死”的论战，到 2026 年有了判决书式的答案：**分疆而治，且互相渗透**。四条 2024–2026 年的硬事实，给本讲的军备竞赛续上最新一章。

**ViT 阵营的三大地标**。其一，Meta 的 DINOv3（2025 年 8 月）在 **17 亿张**精选图像上自监督训练出最高 70 亿参数的视觉骨干，靠“Gram anchoring”技术守住了长训练中稠密特征图的质量，官方称冻结后加个小探针即可超越各专门任务的此前最优——军备竞赛的“燃料”逻辑（7.1 节）在今天依然成立，只是考场换成了自监督。其二，SAM 2（2024 年 7 月开源，Apache 2.0）把一张图像当作单帧视频，用**流式记忆**模块统一了图像与视频的可提示分割：最小模型在 A100 上约 91.2 FPS，实时。其三，IDEA 的 DINO-X（2024 年 11 月）用超一亿样本的 Grounding-100M 支撑**开放词汇检测**：文本说得出名字的、乃至说不出名字的物体都能检，零样本 LVIS 稀有类 63.3 AP、超先前最优 5.8 AP——检测从“认训练集里那 1000 类”走向“万物皆可检”。

**卷积阵营的反击**同样有板有眼：2026 年 1 月发布的 YOLO26 免 NMS 端到端、检测头更轻，CPU 推理较上代最高快 43%，实时检测的工业默认仍然姓卷；2024 年 7 月的 Grounding DINO 1.5 Edge 为了冲开放词汇检测的边缘速度，干脆请 EfficientViT 来当图像编码器；ConvNeXt V2 的设计思想则被 MobileNetV4 吸收进搜索空间——连 Transformer 阵营的工具箱里都装着卷积零件。

所以 2026 年的真实图景不是绞刑架，而是**融合**：大模型看懂世界靠 ViT 基座，跑进车间和手机靠卷积，最强的系统往往两者都要。而本讲的残差捷径、$1 \times 1$ 省钱术、多尺度并行，依旧是这场融合里的通用语法——学它们不是学历史，是学现在进行时。

**ImageNet**

李飞飞团队 2009 年起构建的千万级标注图像数据库；ILSVRC 竞赛（2010–2017）用它 1000 类、约 140 万张图的子集，是深度学习的“高考考场”。

**top-5 错误率**

每张图给 5 个候选类、正确答案不在其中才算错的计分方式；ILSVRC 的官方指标。

**AlexNet**

2012 年冠军：8 层、6000 万参数，靠 ReLU、Dropout、GPU 训练三大法宝引爆深度学习革命。

**ReLU**

$f(z) = \max(0, z)$；不饱和、算得快，AlexNet 之后 CNN 的默认激活函数。

**Dropout**

训练时随机丢弃部分神经元以抑制过拟合的技巧；推理时关闭。

**感受野（receptive field）**

某层神经元能“看见”的输入区域大小；随层数加深逐层扩大。

**1×1 卷积**

不扫空间、只在通道间做加权组合的卷积；用作瓶颈降维省算力，是轻量化标配。

**Inception 模块**

GoogLeNet 的积木：多尺度卷积与池化并行再拼接，让网络自选该信哪个尺度。

**退化（degradation）**

普通深网络训练误差反而高于浅网络的现象；源于深网络难优化，而非过拟合或梯度消失。

**残差连接（residual connection）**

输出 = 主路计算结果 + 输入的捷径相加结构；让网络学“修改量”，梯度可沿捷径直通。

**恒等捷径（identity shortcut）**

不做任何变换、把输入原样抄到输出端相加的那条边；ResNet 的灵魂。

**ResNet**

何恺明等 2015 年提出的残差网络，最深 152 层，以 3.57% top-5 错误率首次超越人类水平。

# 第 8 讲　轻量化网络、全卷积网络与卷积操作模式设计

先把两个数字摆在一起感受一下。VGG-16，第 7 讲的功勋网络：1.38 亿参数，按最常用的 FP32 格式存成文件超过 500 MB——而一部普通手机的运行内存是 8 GB，还要留给系统、微信和相机进程——**一个分类模型就要吃掉小半 GB 的内存预算**，更别说再塞一个检测器、一个分割模型。第二个数字：产线上的工业相机一般按每秒 30 帧出图，那就是**每 33 毫秒必须出一次结果**——这段时间里要完成图像搬运、网络前向、结果后处理，留给卷积的预算只有十几毫秒；而一块 RTX 4090 的功耗超过 400 瓦，你手机充电器总共才 20 瓦。**大，是服务器上的特权；小，才是落地的入场券**。本讲就讲“小而美”的手艺：怎么把网络天生设计得小（深度可分离卷积）、怎么把大网络压小（压缩四板斧）、怎么让网络输出到每个像素（全卷积与语义分割），以及怎么把它真正装进设备（ONNX 与推理引擎）——最后一节，就是你们项目交付格式的说明书。

**本讲地图**

**讲什么**：模型压缩四板斧（剪枝/量化/蒸馏/轻量设计）、深度可分离卷积与 MobileNet、其他轻量家族速览、全卷积网络与语义分割、卷积操作模式工具箱（空洞/分组/可变形/转置）、端侧部署三件套（ONNX/推理引擎/INT8）。**为什么值得学**：六目九头是“几十 KB 的小网络”、组 8 的量化对比、数据基座的 llama.cpp 量化部署、实验二的本地部署——全在本讲。**学完你能**：手算深度可分离卷积省了多少；读懂一份“模型体检报告”；说清 FP32/FP16/INT8 的取舍与 ONNX 在交付链中的位置。**课后阅读**：许庆阳《深度学习模型与算法基础》第 101–115 页。

## 8.1　模型压缩四板斧

一个训好的网络，体积来自两个数：参数量和每个参数的存储精度。压缩的一切手段都是在这两个数上做文章，外加一条“从源头就小”的路线。江湖俗称四板斧：

**表 8.1**　模型压缩四板斧总览：目标一致，路径各异，实战中常常组合使用

| 板斧 | 一句话原理 | 典型收益 | 代价/风险 |
|---|---|---|---|
| 剪枝 | 把不重要的权重或通道置零删掉 | 参数减半到十倍级 | 需重训恢复精度 |
| 量化 | FP32 → INT8，精度换体积 | 体积 1/4，还能加速 | 少数样本上精度下滑 |
| 知识蒸馏 | 大老师教小学生 | 小模型逼近大模型 | 训练流程变长 |
| 轻量设计 | 结构上天生省 | 从根源省算力 | 单体精度略降 |

**剪枝**（pruning）像修剪果树：训练完发现大量权重接近零、砍掉也不怎么影响输出，就把它们（或整个通道）删掉。**量化**（quantization）把每个参数从 32 位浮点压到 8 位整数，体积立省 75%，还能借处理器的整数指令提速——8.6 节重点讲。**知识蒸馏**（knowledge distillation）让一个小“学生”模型去模仿大“教师”模型的输出分布，学到的常比直接用硬标签更多——前沿速递盒子里有它的 2026 年新剧情。**轻量结构设计**则是从图纸阶段就按“手机预算”画楼，主角是下一节。下图的散点把常见模型按“大小 vs 精度”铺开，左下角那片“小而准”的区域，就是本讲全部手艺要守住的

阵地。

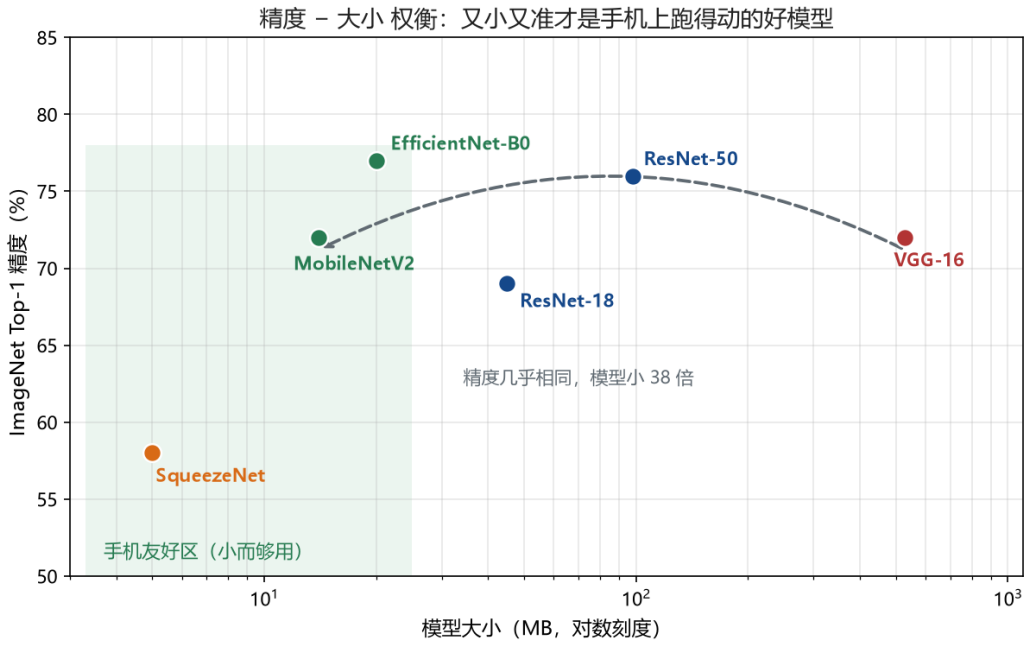



**图 8.1**　图题：模型大小与精度散点图：VGG、ResNet、MobileNet 等各居其位，左下方圈出“手机可跑区”——轻量化的目标是把精度曲线往左下推

## 8.2　深度可分离卷积：轻量化的核心积木

四板斧里，对你们影响最深远的是第四板斧，而它的核心积木只有一个：**深度可分离卷积**（depthwise separable convolution）。先回忆标准卷积在干什么（第 5 讲）：一个 $3 \times 3 \times C_{in}$ 的核，同时干两件事——在**空间**上扫描（左右上下找纹样），在**通道**间混合（把 64 个通道揉成 1 个）。两件事捆绑销售，每个输出像素都要付 $3 \times 3 \times C_{in}$ 的价钱，乘上 $C_{out}$ 个输出通道，账单巨大。

**拆开卖**。深度可分离卷积把这一步拆成两小步：**逐通道卷积**（depthwise convolution）——每个通道自己卷自己的 $3 \times 3$，只在空间上扫描、通道间互不往来；**逐点卷积**（pointwise convolution）——$1 \times 1$ 卷积（第 7 讲见过的老朋友）负责把逐通道卷出的特征在通道间混合。下图把两笔账并排摆开：

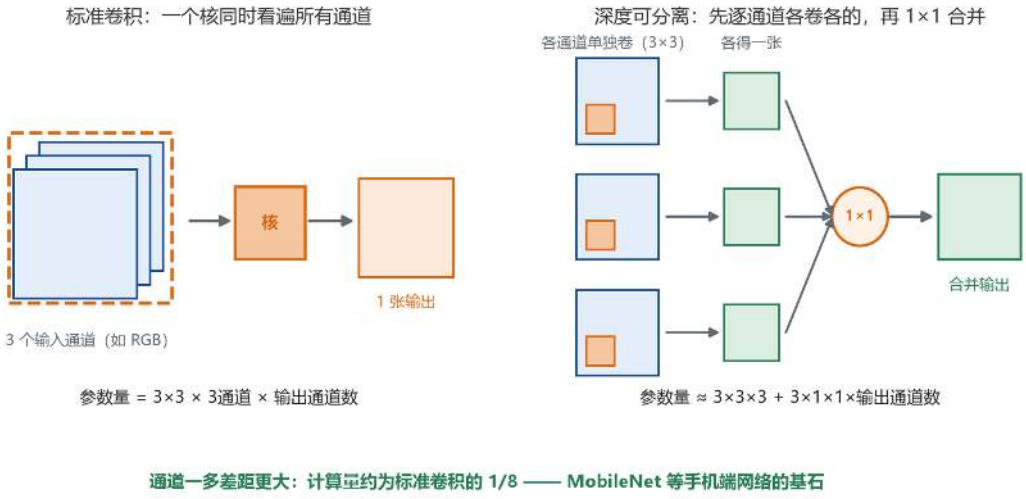



**图 8.2**　图题：标准卷积 vs 深度可分离卷积：同样“空间扫描 + 通道混合”，拆成先逐通道再逐点两步后，计算量降到约八分之一

算一笔账感受差距（这正是本讲课后题的手法）。设输入 64 通道、输出 64 通道、特征图 $56\times56$、核 $3\times3$：

$$\text{标准：}\quad 56\times56\times(3\times3\times64)\times64\approx1.16\times10^{8} \tag{8.1}$$

$$\text{拆分：}\quad \underbrace{56\times56\times(3\times3\times64)}_{\text{逐通道}}+\underbrace{56\times56\times(1\times1\times64)\times64}_{\text{逐点}}\approx1.5\times10^{7} \tag{8.2}$$

**人话翻译**：同样的活儿，拆开干只花约**八分之一**的乘加次数。通式更漂亮：当输入输出通道数都是 $N$ 时，拆分版的计算量约为标准版的 $\frac{1}{N}+\frac{1}{9}$——通道越多越赚，极限逼近九分之一。

2017 年，Google 把这块积木砌成整栋楼，就是 **MobileNet V1**：420 万参数、17 MB 的文件，ImageNet 精度 71.7%——第一次让“手机上跑像样的 CNN”不再是口号。2018 年的 V2 补上关键一刀：**倒残差**（inverted residual）+ 线性瓶颈——把第 7 讲 ResNet 的“宽–窄”残差块反过来，做成“窄–宽–窄”的沙漏：先在窄通道间加捷径，中间用逐通道卷积在膨胀后的宽通道里提特征，最后压缩。宽度上的钱省了，残差的高速公路还在。

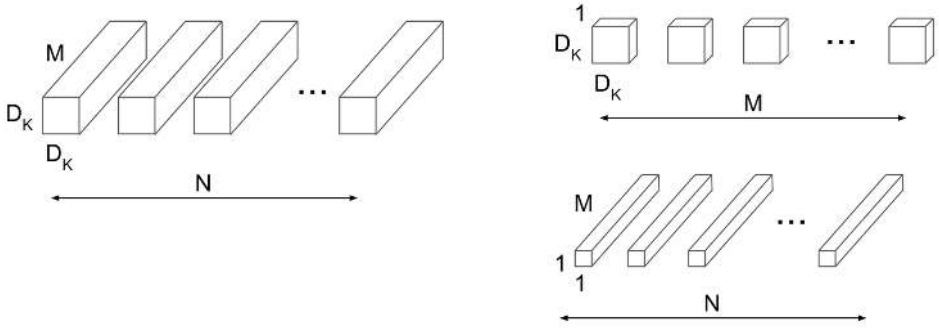



**图 8.3**　MobileNet 论文图：标准卷积（上）与深度可分离卷积（下）的模块对比——先逐通道扫空间、再逐点混通道，计算量大幅下降

图片来源：Howard et al., MobileNets: Efficient CNNs for Mobile Vision Applications, 2017

**动手时间：替你的项目算一笔“卷积账”**

六目项目九个参数头都是小网络，选型时这笔账要会算。设输入 32 通道、输出 128 通道、特征图 $28\times28$、核 $3\times3$：**(1)** 标准卷积的乘加次数是多少？**(2)** 深度可分离卷积（逐通道 + 逐点）呢？**(3)** 比值是多少？更接近 1/8 还是 1/9？**(4) 想一想**：若输出通道翻倍到 256，比值怎么变？为什么通道越多越划算？——答案：(1) $28\times28\times9\times32\times128\approx2.9\times10^7$；(2) $28\times28\times9\times32+28\times28\times32\times128\approx3.4\times10^6$；(3) 约 0.12，介于 1/8 与 1/9 之间、更近 1/8；(4) 比值降到约 0.113——$N$ 越大，$1/N$ 越小，比值越贴着下限 1/9。

## 8.3　轻量家族一分钟点名

MobileNet之外，轻量家族还有几位常客，一表速览——不求都记住，混个脸熟，将来选型时知道去翻谁：

表 8.2　轻量化网络家族速览：各家的“省”法不一样

| 家族 | 省在哪 | 一句话点评 |
|---|---|---|
| SqueezeNet | 先 $1\times1$ 压通道再 $3\times3$ | 2016 年老前辈，AlexNet 精度、五十分之一体积 |
| ShuffleNet | 分组卷积 + 通道混洗 | 组间老死不相往来？洗个牌互通信息 |
| GhostNet | 一半特征用廉价变换生成 | 卷出来的一半是“影子”，影子不必真卷 |
| NAS 自动设计 | 让算法搜结构 | 不人肉画楼，把设计目标写成搜索题 |

表的第一行 SqueezeNet（2016）值得单独两句：它证明了一件震撼业界的事——**AlexNet 的精度，可以用约五十分之一的参数做到**，权重文件仅 4.8 MB。它的积木叫 **fire 模块**：先用一排 $1\times1$ 卷积把通道**压**薄（squeeze，挤一下），再用 $1\times1$ 与 $3\times3$ 两路**并联展开**通道（expand，撑开）——一挤一撑之间，卷积核统统做小，VGG 式的肥大全连接层则根本不设。它没有发明任何新算子，全靠把第 7 讲的 $1\times1$ 省钱术用到极致，是“轻量结构设计”思想的第一次完整示范。

其中两个概念值得多说一句。**分组卷积**（group convolution）：把输入通道分成 $g$ 组，每组各自卷各自、互不往来，计算量直接除以 $g$——它最早出现在 AlexNet，不过是当年两块显卡各存一半通道的无奈之举，如今成了轻量化的正经武器；代价是组与组之间信息不通，于是 ShuffleNet 在分组卷积之间插入“通道混洗”（channel shuffle），把各组的通道打乱重排，让下一轮分组时信息交叉——好比分组讨论到一半强制换一次座位，保证信息在组间流通；靠着“分组 + 洗牌”这两手，ShuffleNet 在精度相当时把计算量压到 MobileNet V1 的几分之一。**NAS**（神经架构搜索，neural architecture search）：干脆不人肉设计，把“精度高、延迟低”写成搜索目标，让算法在海量候选结构里自动淘金——2019 年上线的手机端模型不少出自这类“硬件感知”的自动搜索。你会发现轻量化的思路高度一致：**减少冗余计算，让每一分算力都花在刀刃上**。

## 8.4　全卷积网络 FCN：从“是什么”到“在哪儿”

到目前为止，网络都在回答“这张图**是什么**”。但自动驾驶要知道“可行驶路面**在哪儿**”，医学影像要知道“肿瘤边界**在哪儿**”，锚链检查要知道“缺陷**在**哪个链环**上**”——这类任务叫**语义分割**（semantic segmentation）：**给图里每一个像素发一个类别标签**，相当于把“一个标签”变成“几十万个标签”。

难度立刻暴露：第 6 讲户型图的末端是几层全连接——它们把特征图拉平、揉成 10 个数，**空间位置信息全丢了**，输出尺寸也焊死成类别数，根本到不了“每像素一个标签”。2015 年 Long、Shelhamer 与 Darrell 的 FCN 论文给出漂亮的手术方案，两步：

**第一步，全连接换卷积**。把分类头里的全连接层换成 $1 \times 1$ 卷积（又是它！），网络末端输出的就不再是 10 个数，而是一张与输入位置一一对齐的**热图**（heatmap）——每个像素位置上放着各类别的得分。这种“从头到尾只有卷积层、输入输出都是空间图”的网络，就叫**全卷积网络**（Fully Convolutional Network，FCN）。它还有个可爱性质：不挑输入尺寸，图多大热图就多大。**第二步，上采样回原尺寸**。一路卷积池化下来，热图只有原图的几十分之一大，需要**上采样**（upsampling）放大回原尺寸、逐像素对齐——放大手段从简单到高级有最近邻插值、双线性插值和可学习的**转置卷积**（transposed convolution，俗称反卷积，8.5 节再表）。两步合起来，一个端到端的“图进、像素级标签出”的分割网络就诞生了。

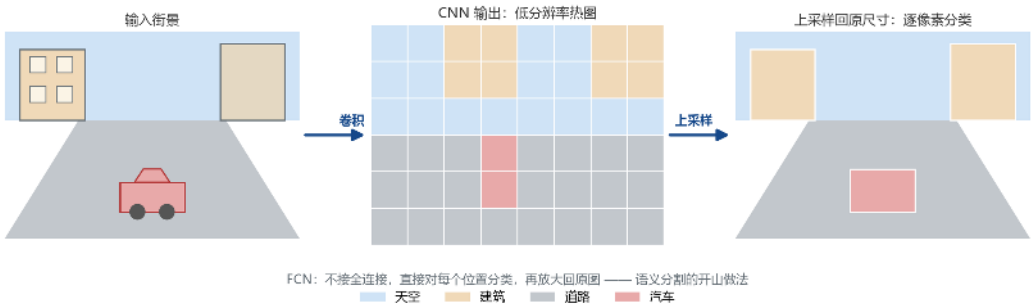



**图 8.4**　图题：FCN 分割示意：输入图像经 CNN 得到低分辨率热图，再上采样回原尺寸，输出与原图对齐的彩色分割图（每个像素一个类别色）

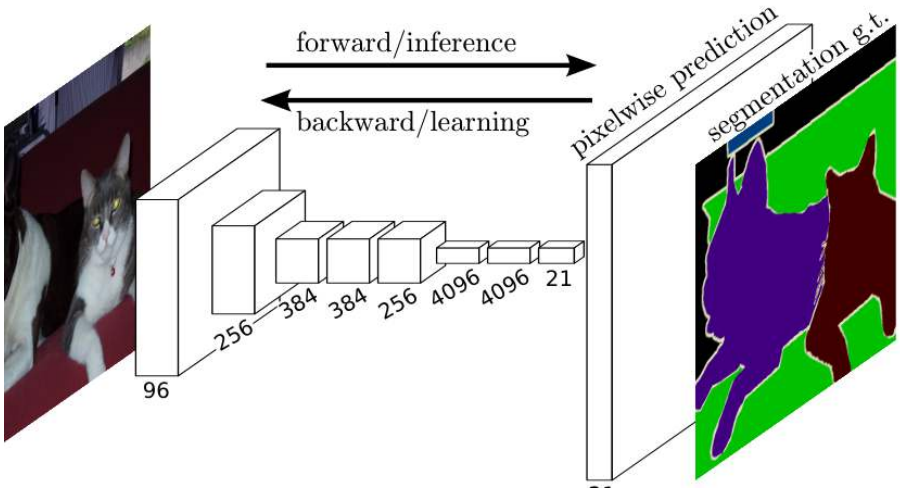



**图 8.5**　FCN 论文分割结果图：输入照片、真值标注与 FCN 预测的分割结果对照——粗糙但成体系，语义分割从此端到端

图片来源：Long, Shelhamer & Darrell, Fully Convolutional Networks, CVPR 2015

原论文里还有点睛的第三笔：**跳级融合**（skip fusion）。直接把最深层的热图放大 32 倍（论文称 FCN-32s），边界粗得像马赛克——深层热图“懂语义”却“丢细节”。于是 FCN 把**浅层**特征图（细节丰富）也接了上来：深层热图先上采样，与浅层特征图**逐点相加**，再继续放大——FCN-16s 融一次、FCN-8s 融两次，分割边界随之越来越细。**浅层管细节、深层管语义，一次相加把两种优势合并**；这记“跳跃连接”的下文马上就到——下一段的 U-Net 会把它用成传奇。

FCN 之后，分割精度靠两个方向持续抬高。一是**看得更大更全**：DeepLab 系列（v3+，2018）用空洞卷积扩大感受野、用多尺度池化并行捕捉大小物体，长期是分割标杆（效果见下图，典型的自动驾驶街景风格）；医学影像领域的 U-Net 则靠“编码–解码 + 跳跃连接”结构成为另一个传奇——你又在跳跃连接，ResNet 的幽灵无处不在。二是**见得更多**：这正是 SAM 登场的剧情，见下下段。分割应用一眼望去：自动驾驶的可行驶区域与车道线、医学影像的器官与病灶勾画、卫星图的建筑物提取、产线质检的缺陷区域框定——**凡是“位置本身就是答案”的任务，都是 FCN 的子孙**。

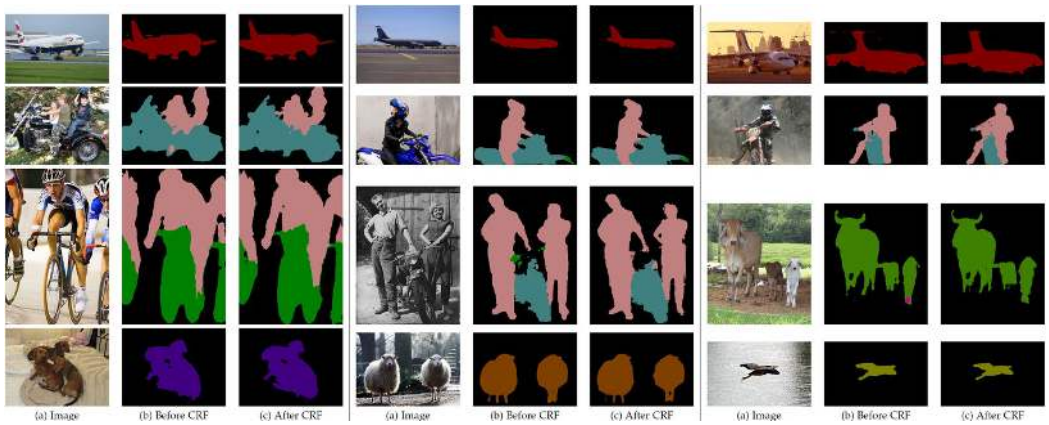



**图 8.6**　DeepLab 语义分割效果图：街景中行人、车辆、道路、天空被逐像素上色——每个像素都有自己的类别标签

图片来源：DeepLab v3+, Chen et al., 2018

2023 年，Meta 发布 **SAM**（Segment Anything，“分割一切”），把分割推入基础模型时代：给它一张新图，你点一个点、框一个框，它就瞬间把对应物体精确抠出——**零样本**，不需要为你的领域重新训练。支撑这个魔法的是一个史无前例的数据集：1100 万张图、11 亿个掩码。它对你还有一个具体的意味：**标注行业的工作方式变了**——过去人工抠一个物体要几分钟，现在 SAM 打初稿、人只做修边检查，效率翻数倍；你水下项目（第 二 章）调研中的“自动标注”路线，走的就是“大模型打初稿 + 人审核（或教师模型打伪标签）”这条线索。

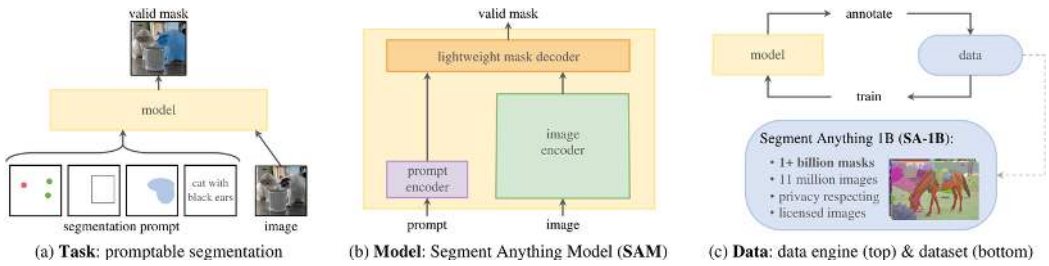



**图 8.7**　SAM（Segment Anything）示例：用户在图上点几个点（或框一个框），模型零样本分割出指定物体，边缘精确到发丝

图片来源：Meta AI, Segment Anything 项目页, 2023

**扫盲：分类、检测、分割——三个任务的差别**

同一张街景图，三个任务三种输出：**分类**输出一个词（“街景”）；**检测**输出若干个框，每个框一个类别（“此处有一车、此处有一人”）；**分割**输出一张与原图同尺寸的“涂色图”，每个像素都有自己的类别。信息量依次暴涨，标注成本也一样——分类一张图一个标签，分割一张图几十万张“小标签”。所以分割数据集总是又小又贵，这也是 SAM 这类零样本分割工具对行业冲击巨大的原因。

## 8.5　卷积操作模式工具箱

走到这里，值得把“卷积”这个词底下的各种玩法收进一个工具箱。它们都是 8.2 节“拆解与重组”思想的不同方向：

**表 8.3**　卷积操作模式速查表：改哪个自由度，换什么能力

| 模式 | 改的是 | 得到什么 | 代表应用 |
|---|---|---|---|
| 标准卷积 | —— | 基线 | LeNet/VGG全家 |
| 深度可分离 | 通道混合与空间扫描解耦 | 省约 1/8 计算量 | MobileNet（8.2 节） |
| 分组卷积 | 通道分组各卷各的 | 计算量 $/g$ | AlexNet/ShuffleNet |
| 空洞卷积 | 核内插“孔”不加大核 | 感受野翻倍、参数不变 | DeepLab 分割 |
| 可变形卷积 | 采样位置可学习 | 贴合物体形变 | 实例分割/姿态 |
| 转置卷积 | 反向的卷积 | 可学习的上采样 | FCN/生成模型 |

**空洞卷积**（dilated/atrous convolution）最值得展开：把 $3\times3$ 核的九个采样点彼此拉开距离（中间插“孔”，间距称膨胀率 rate），核还是九个参数，视野却从 $3\times3$ 扩到 $5\times5$（rate=2）——**不花一个新参数，白得大感受野**。它解决了分割任务的一个死结：分割网络常去掉了池化层（保住分辨率），于是感受野缩水，空洞卷积正好不靠池化就能把视野补回来。**可变形卷积**（deformable convolution）则更进一步：连“采样点摆在哪”都交给网络学，卷积窗口随物体的形状扭曲——猫的尾巴细长，采样点就顺势拉长。机制上，它额外挂一支小型卷积网络，为每个采样点预测一个位移量，整套位移合称**偏移场**（offset field）——连窗口的形状本身都成了可学习的参数。**转置卷积**（transposed convolution）是 FCN 用过的上采样招式：普通卷积把大图卷小，它把小图“逆卷”回大图，且放大方式可学习。看到规律了吗？**卷积不再是一个固定的算子，而是一个可设计的部件家族**——这正是本章标题里“卷积操作模式设计”的含义，也是工业界大量专利与论文的聚集地。

工具箱还该补上 2020 年代最有戏剧性的一章：**大核卷积回潮**。第 7 讲的 VGG 算术——“小核堆叠完胜大核”——统治了此后七年，几乎成了教条；2022 年，回旋镖结结实实飞了回来。清华团队的 **RepLKNet**（CVPR 2022）把卷积核一口气开到 $31\times31$，配合**结构重参数化**（训练时用多条小分支搭出等效结构以稳住训练，部署时全部折叠成一个真正的大核，推理时不多花一分钱），在多项下游任务上明显胜过同体量的小核网络；同期的 **SLaK** 更把稀疏布置的大核推到 $51\times51$。大核带来的不只是感受野的数字，而是**“一眼看到整个物体”的归纳偏置**，代价是必须靠重参数化与精心安排才跑得快。给初学者的启示并非“大核更好”，而是：**算术上的最优解未必是工程上的最优解，任何教条都值得用实验重新问一遍**。

另一个必须点名的是 **SENet**（Squeeze-and-Excitation，2017 年 ImageNet 分类冠军），它给工具箱添上了**通道注意力**（channel attention），三步走：先对每张特征图做全局平均池化，把“这个通道整体有多活跃”压成一个数；再过两层全连接（先收缩、再放宽，中间夹激活），输出每个通道的重要性权重；最后把权重逐通道乘回特征图——**让网络自己决定哪个通道重要**：重要的放大，没用的压暗。整套 SE 模块几乎不加参数、即插即用，今天大量骨干网络里仍躺着它的变体。用本讲的语言总结：深度可分离卷积拆出了“空间”与“通道”两个自由度，SENet 给“通道”这个自由度装上了看内容定权重的智能开关。

## 8.6　端侧部署三件套：ONNX、推理引擎与 INT8

网络训好只是半场，**部署**（deployment）才是把模型真正变成产品的那半场——也是你们实验二（11 月 2 日）“本地部署”的考试范围。三件套按数据流顺序登场。

**第一件：ONNX**（Open Neural Network Exchange，开放神经网络交换格式）。PyTorch、TensorFlow 各说各的方言，直接把模型交给部署端就像把 `.doc` 发给只有 WPS 的同事。ONNX 是行业的“通用普通话”：把网络翻译成一份标准的**计算图**文件（图上每个节点是一个算子、每条边是数据），任何支持它的运行时都能执行。`torch.onnx.export` 一行即可导出。**你们六目项目的九个参数头与端到端 `ropstitch.onnx`，交付格式就是它**——头文件配同名 `metrics.json`，拷入 `build/Release/models/` 即被系统识别。**第二件：推理引擎**。拿到 ONNX 后，谁在设备上执行这张图？**推理引擎**（inference engine）：NVIDIA 显卡上的 TensorRT 会做算子融合、自动半精度等图级优化；手机 CPU 上的 NCNN（腾讯开源）零依赖、对 ARM 指令深度优化；同门还有 TFLite、MNN 等。选引擎先选硬件，这在实验二里你们会亲手体会。**第三件：INT8 量化**，压轴戏，先用知识盒补一个前置概念。

**知识补齐：FP32 / FP16 / INT8——三种记账精度**

计算机存数靠位数，位数就是“账本精细度”。**FP32**（32 位浮点）：1 位符号 + 8 位指数 + 23 位尾数，约 7 位有效数字——像精确到分的账本，训练默认用它。**FP16**（16 位浮点）：1+5+10 位，约 3 位有效数字——账本只记到毛票，但体积减半、很多芯片算它反而更快。**INT8**（8 位整数）：只能记 $-128$ 到 127 的整数——像只划“正”字的计分板。那小数怎么办？**量化的本质**：先量出这批数的范围 $[\min, \max]$，把范围等分成 255 格，每个数只记“落在第几格”，另存一个缩放系数（scale）负责换算回来。精度损失来自“格子变粗”，收益是体积 1/4、整数指令提速、内存带宽省 4 倍。

量化不是免费午餐，权衡如下表。分类任务的实操经验：训练完成后用几百张“校准图”统计范围、直接转 INT8（称**训练后量化**，PTQ），精度往往只掉零点几个百分点；但任务越精密（回归、立体匹配这类输出连续值的任务），INT8 越容易掉链子，必要时得用**量化感知训练**（QAT）——训练阶段就“假装”自己是 INT8，让网络提前适应粗格子。

表 8.4　三种精度档次的典型权衡（分类任务的量级参考，具体任务必须实测）

| 精度 | 体积 | 速度 | 精度损失 | 适用 |
|---|---|---|---|---|
| FP32 | 1× | 基线 | 0 | 训练与精度基准 |
| FP16 | 0.5× | 常快于 FP32 | 几乎无 | GPU部署首选 |
| INT8 | 0.25× | CPU/GPU 均显著提速 | 通常 <1%，个别任务更大 | 产线/端侧大批量 |

所以工程上的铁律是：**量化必实测**——在部署机上用真实任务指标验收，而不是相信“通常掉不到 1%”。你们项目的验收线就是这条铁律的定量版：组 8 的量化版 onnx，重投影误差劣化必须小于 10% 才允许交付。

**前沿速递：端侧 AI 大爆发——大模型正在塞进你的手机**

本讲的手艺正在经历史上最大的一次应用浪潮。**端侧大模型**：2024–2026年，手机厂商与芯片厂把量化后的大语言模型直接装进旗舰机（Google 的 Gemini Nano、苹果的 Apple Intelligence、各家端侧版 Qwen 等），本地就能做摘要、改写、识图——不联网、不泄密、零延迟。支撑它的正是“蒸馏 + 量化”双板斧：几十亿参数的教师蒸馏成 7B 上下的小模型，再经 4 比特量化（GGUF

格式）压到 4 GB 上下，8 GB 内存的手机就装得下；开源社区 `llama.cpp` 一套代码，把量化模型跑在手机、笔记本、开发板上——你们数据基座项目在本地用 llama.cpp 量化部署 Gemma 12B，走的就是这条路。同时，NCNN/MNN/TFLite 构成的端侧推理生态日臻成熟，“手机跑 CV 模型”已是无传感器厂商的标配。更值得年轻人注意的趋势是**知识蒸馏 2.0**：教师不再只是更大的分类网络，而是大语言模型本身——用它海量的“软输出”去教一个专用小模型。你们水下项目“FoundationStereo 当教师、UWStereo 当学生”的伪标签闭环（第 二 章），方法论上与它同宗同源。**本讲学的压缩与部署手艺，是 AI 从机房走进世界的必经窄门。**

**项目连线：第 8 讲 × 三个项目——这一讲就是你们的“部署说明书”**

**六目拼接**（第 一 章）：九个参数头每个都是 ≤ 几十 KB 的轻量小网络，CPU 上实时跑——“轻量”在这里不是选修课，是入场券；8.6 节的 ONNX + `metrics.json` 正是它们的交付格式。组 8（端到端优化组）的本职工作就是本讲 8.6 节的完整实操：把 `ropstitch.onnx` 在部署机上做 **FP32/FP16/INT8 三档**的耗时与重投影误差对比，**INT8 劣化 <10% 才交付量化版**，否则回退 FP32——把本讲的权衡表变成一张真实的验收表。**数据基座**（第 三 章）：Gemma 12B 在本地用 llama.cpp 做量化部署（GPU、上下文 8192、并发 1），量化档位的选择就是精度–显存的实时权衡——大模型的轻量化，与本讲同一套思想。**水下双目**（第 二 章）：部署组在测显存与吞吐、做 FP16 与 ONNX 导出初探时，用的全是本讲工具箱。最后划重点：**实验二（11 月 2 日）“匹配合适程序并执行本地部署”就是本讲的实操考场**——ONNX、推理引擎、量化、llama.cpp 都会在那场实验里与你见面，请带着本讲去上课。

本讲课后题：给你的模型做一次“体检”，把“轻量”从口号变成四个具体的数字。五要素如下。

**第 8 讲课后编程题（用 AI 实现）**

**题目**：指挥 AI 辅助完成一份“模型体检报告”：取 torchvision 自带的两个预训练模型 `resnet18` 与 `mobilenet_v2`，在一个猫狗二分类数据集上（自备几百张猫狗图，或用 CIFAR-10 的猫/狗子集，替换分类头后各微调 2 个 epoch）分别测量四项指标：(1) 参数量（百万）；(2) 权重文件大小（MB）；(3) CPU 上单张图片的平均推理耗时（测 20 张取平均）；(4) 测试集准确率（%）。把结果整理成一张对比表，并写 50 字以上的选型建议：手机端产线质检该选谁？服务器端呢？

**交付物**：(1) 代码文件 `model_checkup.py`；(2) 四项指标对比表截图；(3) 50 字以上的选型建议与“我改了什么”说明。

**提示词建议**（可直接抄）：“我是零基础 Python 学生。请写一个脚本比较 torchvision 预训练的 resnet18 与 mobilenet_v2：(1) 用 sum(p.numel() for p in model.parameters()) 统计参数量；(2) 用 torch.save 保存权重后用 os.path.getsize 报告文件 MB 数；(3) 把模型放到 CPU，用 time.time() 计时对同一张图片连续推理 20 次取平均耗时；(4) 在我提供的猫狗图文件夹上微调 2 个 epoch（batch size 32、学习率 0.001、Adam）后在测试集上算准确率。最后用 print 输出一张对齐的四行对比表；每行代码加中文注释；不要用 GPU。”

**验收线**：四项指标齐全且来自你自己电脑的真实运行；能不看资料解释“MobileNetV2 为什么更小更快、代价是什么”（提示：8.2 节那笔账）；至少有一处自己的修改（换第三个模型、换输入尺寸、换数据集都算）。**代码是 AI 写的没关系，讲不清楚就是没做。**

**加分项**：把其中一个模型导出为 ONNX（`torch.onnx.export`），用 `onnxruntime` 在 CPU 上重测推理耗时，与 PyTorch 原生对比——谁更快？差多少？用一两句话解释推理引擎做了什么优化。

进一步可尝试 FP16 权重再测体积与耗时。

## 8.7　从这一讲到 2026

2026 年的“轻量化”三个字已经换了含义：不再是“把小模型剪到能跑”，而是**“把大模型压进手机”**。五条硬事实，按“架构—引擎—量化”三层摆开。

**架构层**。MobileNetV4 最大的 Hybrid-Large 款在 Pixel 8 的 EdgeTPU 上 **3.8 毫秒**完成一次推理、ImageNet-1K 精度 87%，其面向移动加速器设计的 Mobile MQA 注意力带来 39% 提速，在手机 CPU/GPU/DSP、苹果神经引擎上近乎处处帕累托最优——“为每一种端侧芯片都高效”成了小模型的新标准。分割这边，MIT Han Lab 的 EfficientViT-SAM 用多尺度线性注意力换掉 SAM 的重型编码器，TensorRT 实测较 SAM-ViT-H **提速 48.9 倍**而精度不降，医疗版还拿下 CVPR 2024 笔记本医疗 SAM 挑战赛冠军——分割基础模型的端侧化宣告完成。

**引擎层**。阿里开源的推理引擎 MNN，核心库只有约 **800 KB**，却部署在淘宝、天猫、钉钉等 30 多个应用里，通过 MNN-LLM 与 MNN-Diffusion 让手机**本地**跑大语言模型和扩散模型，3.6.1 版更新增了高通 Hexagon NPU 后端——8.6 节那张“推理引擎”名单，如今的主战场已是大模型。

**量化层**。苹果的 Apple Intelligence 用约 **30 亿**参数的端侧基座在设备上运行，更大的模型留给私有云——手机 NPU 跑基座模型已是消费级现实。获 MLSys 2024 最佳论文奖的 **AWQ**（激活感知权重量化）证明：只需重点保护约 1% 的“显著权重”，就能放心做 W4A16 的 4 比特量化，其 TinyChat 引擎在桌面与移动 GPU 上较 FP16 提速 3 倍以上——**4 比特已成为端侧大模型的默认档位**，你数据基座项目里 llama.cpp 的量化档位选择（第 三 章），正是同一条技术路线的现场版。

一句话总结这三年：**轻量化的终局不是把小模型做小，而是把大模型做进口袋**——量化、蒸馏与端侧引擎，让手机 NPU 成了新的“显卡”。你在本讲学的每一招，都直接通向那道窄门。

**模型压缩**
让网络更小更快的各类技术总称；四板斧为剪枝、量化、知识蒸馏、轻量结构设计。

**剪枝（pruning）**
删除接近零或不重要的权重/通道以缩小模型；通常需微调恢复精度。

**量化（quantization）**
降低参数存储精度（如 FP32 转 INT8）换取四分之一体积与加速；需实测精度损失。

**知识蒸馏（knowledge distillation）**
让小模型模仿大模型的输出分布来学习，“大老师教小学生”。

**深度可分离卷积（depthwise separable convolution）**
逐通道卷积（管空间）+ 逐点 $1\times1$ 卷积（管通道）；计算量约为标准卷积的 $1/N+1/9$。

**倒残差（inverted residual）**
MobileNetV2 的“窄–宽–窄”残差块；捷径连在窄通道之间，中间膨胀后提特征。

**分组卷积（group convolution）**
输入通道分组各卷各的，计算量除以组数；ShuffleNet 用通道混洗打通组间信息。

**语义分割（semantic segmentation）**
给图中每个像素一个类别标签；输出是与原图对齐的“涂色图”。

**全卷积网络（FCN）**

用 $1 \times 1$ 卷积替换全连接、输出热图再加采样的网络；语义分割的端到端开端。

**上采样（upsampling）**

把小分辨率特征图放大回原尺寸的操作；插值或可学习的转置卷积。

**空洞卷积（dilated convolution）**

核内采样点拉开间距以扩大感受野，参数量不变；DeepLab 的主力。

**转置卷积（transposed convolution）**

实现“可学习放大”的算子，俗称反卷积；FCN 上采样的手段。

**ONNX**

开放神经网络交换格式：把模型导出为通用计算图文件；本项目所有头网络的交付格式。

**推理引擎（inference engine）**

在具体硬件上执行模型计算图的运行时（TensorRT、NCNN、TFLite 等）。

**训练后量化（PTQ）/量化感知训练（QAT）**

训完用校准集直接量化；或训练时就模拟量化让网络适应。前者省事，后者更保精度。

# 第 9 讲　循环神经网络 RNN

先做个一秒钟的小实验：打开手机输入法，打下“今天天气”，看它猜的下一个词——多半是“真不错”“怎么样”。这个你每天都在用的功能，背后站着本讲的主角：一个**边读边记**的网络。而在它之前，我们认识的全部网络——感知器（第 1 讲）、多层感知器（第 2 讲）、卷积神经网络（第 5 讲）——都有一条共同的隐含设定：**输入一次给全，答案当场给出**。它们像“一眼定终身”的裁判：照片整张递进来，结论直接吐出去，案子即告终结。没有“上一张照片”，也没有“下一个词”。

可惜真实世界的数据一多半不是这样递过来的。你读一句话是**一个字一个字**读的，读到“它”字要靠前文才知道它指谁；听语音是**一段一段**听到的；相机工作时的匹配数、增益是**一帧一帧**积累出来的。这类“有先后次序”的数据叫**序列**（sequence），对付它，网络必须带上记忆。1980 年代就有了答案的**循环神经网络**（Recurrent Neural Network，RNN），是给网络装上记忆的第一代成熟方案；更重要的是，它是你理解 2026 年一切序列模型（包括大模型）绕不开的地基——第 11 讲你会看到，大模型的看家本领“文字接龙”，原型正是本讲的 9.3 节。

**本讲地图**

**讲什么**：序列数据与时间步；RNN的核心思想（同一个网络反复用，隐藏状态当便签）；按时间展开；用 RNN 做语言模型、玩文字接龙；沿时间反向传播（BPTT）；梯度消失与爆炸；RNN 的应用与局限。**为什么值得学**：第 11 讲的大模型与三次动手实验（10 月 12 日 agent 配置、11 月 2 日本地部署、11 月 23 日离线训练）的主角都是“接龙高手”，接龙的源头在本讲；而相机状态、ROV 遥测这类小数据工业时序，2026 年仍是 RNN 家族的主场。**学完你能**：手算一步 RNN 状态传递；说清“权值共享在时间维”是什么意思；解释梯度为什么会连乘到消失；判断一个任务该不该上 RNN。

## 9.1　序列数据无处不在：世界是按帧来的

**序列数据**（sequential data）就是一串**次序本身携带信息**的数据：词的先后排成句意，帧的先后排成动作，采样的先后排成趋势。判断一类数据是不是序列，有个毒舌但有效的检验——**打乱次序，看信息还剩多少**。把一句话的字序打乱，剩下一堆乱码；把视频的帧序打乱，剩下一堆幻灯片；把心电图按时间乱排，医生当场报警。对比之下，把一张猫照片的像素行随机重排成另一张“图”，对“判断是不是猫”这个任务来说固然也是灾难——但猫图之间彼此独立，而序列里的元素彼此依赖：**每一个元素的含义，取决于它前面的元素**。

下表把四类最常见的序列摆在一起。注意“一个时间步是什么”这一列：同是序列，切法天差地别。

**表 9.1**　四类常见序列：一个时间步各是什么

| 序列类型 | 一个时间步 | 典型长度 | 打乱次序后剩什么 |
|---|---|---|---|
| 文本（一句话/一段话） | 一个字或一个词 | 几十到几千步 | 乱码 |
| 语音 | 一小段声波（约 20 毫秒） | 上万步 | 噪音 |
| 视频 | 一帧图像 | 几十到几千帧 | 幻灯片 |
| 传感器时序（温度/相机状态/遥测） | 一次采样读数 | 几十到上百万点 | 趋势全丢 |

**生活化类比：时间步——给序列装上一台滴答钟**

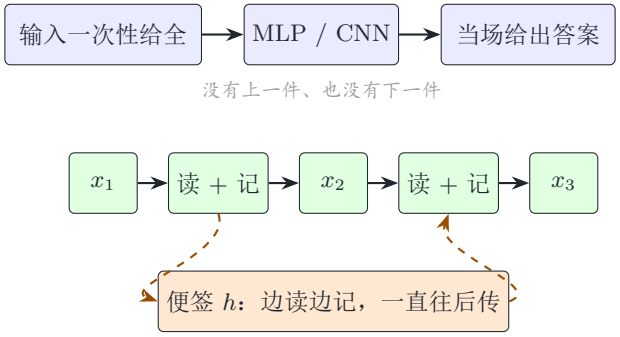
处理序列的第一步，是把连续的流切成一帧帧，切出来的每一份叫一个**时间步**（time step），用 $t$ 编号：$t=1$ 是第一步，$t=2$ 是第二步……就像给流水装上一台滴答作响的钟，钟每响一下，网络吃进一份输入。顺带提前认识一个你在项目里天天会见的形状写法 $[B,T,D]$：$B$（batch）是一批样本数，$T$ 是序列长度（时间步数），$D$ 是每个时间步的特征维数——比如“一个批次的相机状态序列”就是 $[8,32,24]$：8 组样本、每组 32 步、每步 24 个数。以后见到张量形状，先数清这三个字母。

为什么无记忆的网络干不了这活儿？两个理由。其一，**单步看没有答案**：只看“它”一个字，神仙也不知道它指可丽饼还是指天气；答案藏在上下文里，而上下文就是“前面读过的所有步”。其二，**序列长度不定**：MLP和 CNN 的输入尺寸是固定的，而句子有长有短、音频有长有短，总不能每换一个长度就重造一个网络。于是需求非常明确：造一个**能一步一步吃输入、还记得自己吃过什么**的网络。

在 RNN 登场之前，人们怎么让机器“读懂”一句话？补一段前史，帮你看清 RNN 到底新在哪。第一条路线叫**词袋**（bag of words）：把句子拆成一袋词，只统计“哪些词出现过、各出现几次”，词的先后顺序整个扔掉——“我打了他”和“他打了我”在词袋里一模一样，谁动了谁无从谈起。第二条路线叫 **N-Gram**：学乖了一点，把相邻 $N$ 个词的搭配存进一张大表——2-Gram 存“我昨天”“昨天上学”这样的词对，3-Gram 存“我昨天上学”这样的三连词组，预测下一个词就靠查表。可哪怕“我昨天上学迟到了”这样一句家常话，想接得准也已经要看三四个词的搭配；而理论上搭配的数目是**词表大小的 $N$ 次方**——$N$ 每加大 1，表中条目就要乘上一整个词表：2-Gram 尚可硬存，3-Gram 已是海量，$N$ 再往上去，存储与统计双双**指数爆炸**。何况 $N$ 再大也有限，$N$ 个词之外的上下文它一概不知。两条路线的共同病根是**死记硬背**——背的是组合，不是规律。所以最终胜出的，是下一节这位带着记忆、边读边消化的选手。

## 9.2　RNN 的核心思想：一个网络用到底，一张便签传下去

RNN的设计朴素得惊人，两句话说完。**第一句：同一个网络反复用**——每个时间步都用同一套权重读一份新输入。**第二句：留一张便签当记忆**——网络维护一个向量，叫**隐藏状态**（hidden state），读每一步时先瞄一眼便签上记着什么，再把新读到的内容写进便签，然后传给下一步。下图把“无记忆”与“有记忆”两种工作方式摆在一起对比：上面一行看完就散，下面一行靠着那张橙色便签把信息一站站传下去。




写成公式只有一行。第 $t$ 步，网络拿新输入 $x_t$ 和旧便签 $h_{t-1}$，算出新便签 $h_t$：

$$h_t = \tanh(W_x x_t + W_h h_{t-1} + b) \tag{9.1}$$

人话：新来的输入 $x_t$ 乘矩阵 $W_x$（“这次来的事怎么记”），旧便签 $h_{t-1}$ 乘矩阵 $W_h$（“以前记的事怎么留”），两股信息相加、再加偏置 $b$，最后过 tanh（第 2 讲激活函数之一）把每个数压进 $-1$ 到 $1$——防止便签越写越爆。$t=1$ 时 $h_0$ 取全零（一张空白便签）。若每步还要输出（比如猜下一个字），再接一层 $y_t = W_y h_t + b_y$ 即可。$W_x$、$W_h$、$W_y$ 这几套矩阵，就是 RNN 要学的全部参数——注意它们**与序列长度无关**：句子再长，参数一个不多，这正是“同一个网络反复用”的底气。

把上面的循环在纸上按时间“拉开”，就是下图：同一个模块（灰色块，权重完全相同）被复制三份排开，隐藏状态像接力棒横向传递。这张“展开图”是理解 RNN 的一切钥匙——本讲的训练与病灶分析全在它上面进行。

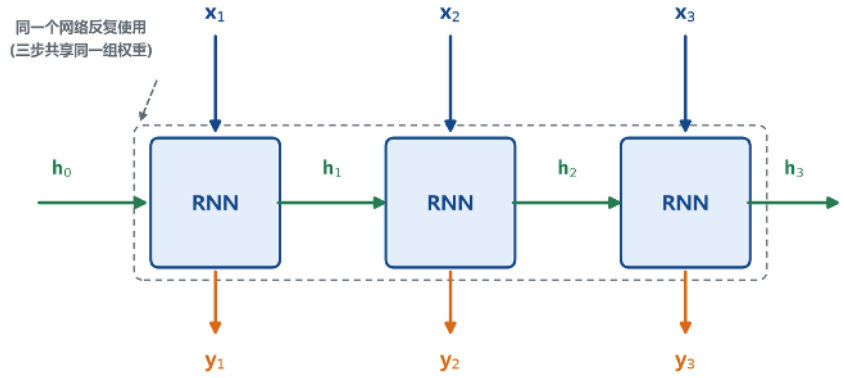



**图 9.1**　RNN 按时间展开：同一个模块复制三步，输入 $x_1, x_2, x_3$ 逐步入场，隐藏状态 $h$ 沿横向箭头一站站传递，每步可给出输出 $y$

同一个循环，Christopher Olah 2015 年那篇著名博客画得更写意——下面这张图后来成了 RNN 的“标准照”：左边的折叠形态里，网络块自带一条打弯的自环箭头；沿着时间拉开，就是右边的展开形态。看懂那条打弯的箭头，就看懂了“循环”二字的全部：

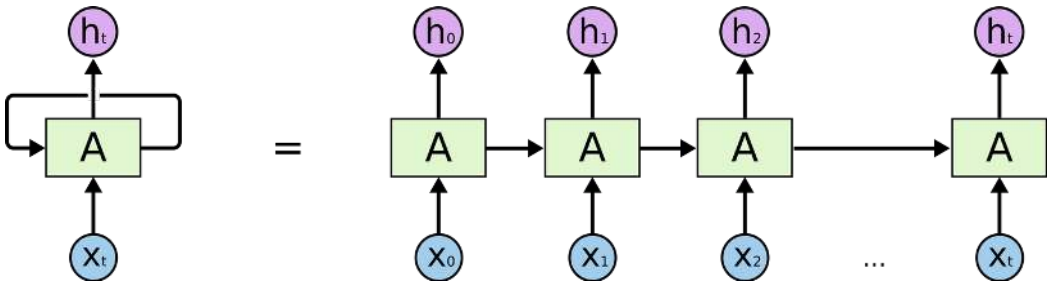



**图 9.2**　RNN 按时间展开的经典图：左侧折叠形态（模块带自环箭头），右侧为沿时间展开后的形态，隐藏状态逐站横向传递

图片来源：Christopher Olah, ”Understanding LSTM Networks” 博客文章, 2015

再品一层：**权值共享**。第 5 讲 CNN 的卷积核是“在空间维共享”——同一块模板扫遍全图每个位置；RNN 的 $W_x, W_h$ 是“在时间维共享”——同一套“怎么读、怎么记”的规则用在第 1 步、第 7 步、第 100 步。两者的动机一模一样：**规律不挑位置**。猫耳长在图里哪个角落都还是猫耳，“主语到谓语”的语法距离十个字还是一百个字都成立。共享让参数量与数据尺寸脱钩，这是深度学习两大支柱（CNN 与 RNN）共用的记账智慧。

**动手时间：亲手当一次 RNN——三步手算看“便签饱和”**

取最简单的情形：$x_t$ 和 $h_t$ 都是单个数，$W_x=1$，$W_h=0.5$，$b=0$。连喂三个 1：$x_1=x_2=x_3=1$。拿草稿纸（或手机计算器）逐步算 $h_t=\tanh(1+0.5\,h_{t-1})$，tanh 值可查表或按 $\tanh(u)=\dfrac{e^{2u}-1}{e^{2u}+1}$ 现算：

- $h_1=\tanh(1+0)=0.76$；
- $h_2=\tanh(1+0.5\times 0.76)=\tanh(1.38)=0.88$；
- $h_3=\tanh(1+0.5\times 0.88)=\tanh(1.44)=0.89$。

盯着这串数看两件事：其一，三次输入完全相同，便签却一次比一次大——**新输入在不断覆写旧记忆**，$h_1$ 里的“0.76”到 $h_3$ 时只剩一点影子；其二，便签在向 1 饱和，tanh 的压缩让 $h$ 装不下更多东西。再顺手用计算器算 $0.8^{20}\approx 0.012$、$1.2^{30}\approx 237$ 记在草稿边——9.5 节讲梯度消失与爆炸时，这两个数就是全部剧情。

收好草稿纸，最后认识 RNN 这块积木的两种经典拼法，各记一句话，第 10 讲在 PyTorch 里改一行参数就能打开。其一，**堆叠 RNN**（stacked RNN）：把两层乃至更多层 RNN 像积木一样竖着摞——下层逐字读出“字面”记忆，上层在下层的记忆基础上再加工，学更高层、更抽象的模式，与多层感知器（第 2 讲）层层堆出“从边缘到零件再到整脸”是同一个道理。其二，**双向 RNN**（bidirectional RNN，简称 Bi-RNN）：把序列正着读一遍得到一份记忆，再倒着读一遍得到另一份记忆，两份拼接起来用。它为什么有用？想想“完形填空”：只看左半句你只能蒙，前后文都看全才有把握——词性标注、命名实体识别这类“整句读完才好下结论”的任务里，双向几乎成了标配。

## 9.3　用 RNN 做语言模型：文字接龙

RNN最经典的应用是**语言模型**（language model）：给“接下来的字”算概率。玩法见下图：把一句话逐字喂进 RNN，每读一个字，网络的输出层（softmax，第 6 讲）就给出一张“下一字概率榜”——比如读完“今天天气”，榜上“真”得 0.31、“很”得 0.22、“不”得 0.09……训练目标简单到令人发指：**语料里下一个字是谁，就把榜上谁的分推高**。整句读完，每一步的预测误差加起来算损失，误差经反向传播把 $W_x, W_h, W_y$ 修一点。

**知识补齐：最小语言模型实例——一句话如何变成一道数学题**

“给接下来的字算概率”说起来抽象，现在拿一句最短的语料（整个训练集就这一句）把它变成一道看得见的数学题：**“我每天要吃千层面”**，8 个字。**第一步，编词典**：给语料里每个字发编号——我 =0、每 =1、天 =2、要 =3、吃 =4、千 =5、层 =6、面 =7。**第二步，向量化**：把每个字变成 **one-hot 向量**——长度等于词典大小、只有自己那一格是 1 的向量（“吃”就是第 4 格为 1、其余全 0 的 8 维向量）。**第三步**：RNN 每读入一个字的 one-hot，softmax 就输出一张长度为 8 的概率榜，训练目标不变：榜上“下一个字”那一格的分被推高。训练时网络“吃的题”与“对的答案”一字排开，就是下表：

| 时间步 | 输入（字 → one-hot） | 标签（下一个字） | 这一步教会网络什么 |
|---|---|---|---|
| $t=1$ | 我（第 0 格为 1） | 每 | “我”后面接“每” |
| $t=2$ | 每（第 1 格为 1） | 天 | “每”后面接“天” |
| $t=3$ | 天（第 2 格为 1） | 要 | “天”后面接“要” |
| … | … | … | … |
| $t=7$ | 层（第 6 格为 1） | 面 | “层”后面接“面” |

one-hot 的毛病也当场现形：真实语料的词典动辄几万字，每个字就是一根几万维、只亮一格的“又高又瘦”的稀疏长条——浪费，而且任意两字“人人等距”，谁跟谁更像完全看不出。所以工程上 one-hot 只当临时工，转身就换成可学习的稠密词向量（embedding，第 11 讲的 Transformer 正式用它）。但骨架你已经全部看清：**词典编号 → 向量化 → 逐字预测下一字 → softmax 概率榜**——第 11 讲那些万亿参数的大模型玩“文字接龙”，第一天也是用这张 8 行小表练出来的童子功。

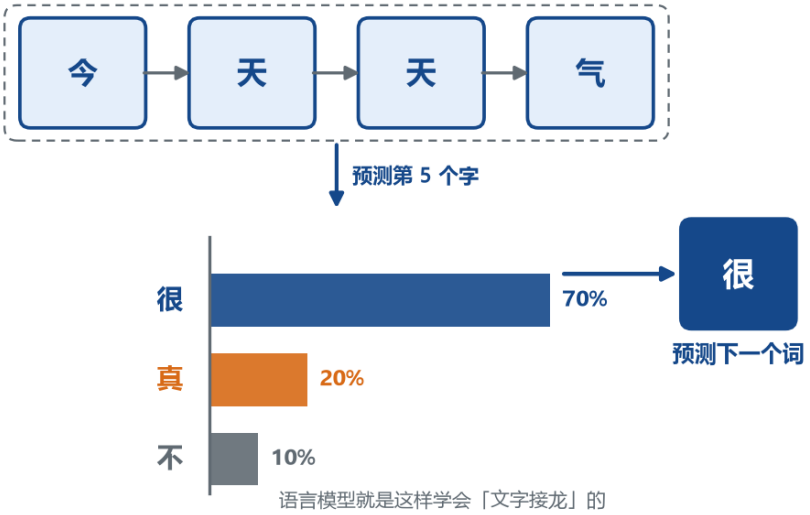



**图 9.3**　用 RNN 玩文字接龙：“今天/天气/很/好”逐字读入，每步输出一张“下一字”概率榜（右侧条形），榜上取一个字拼回去，接着猜再下一个

生成文字就是把接龙玩起来：榜上抽一个字，拼回输入，再猜下一个字。几行伪代码就能说清这个循环：

```
h = zeros                # 便签清零（h0 = 0）
for ch in "今天天气":     # 逐字读入序列
    h = rnn_step(ch, h)  # 每读一字：h = tanh(Wx*x + Wh*h + b)
scores = softmax(Wy @ h) # 便签给出"下一字"概率榜
next_ch = sample(scores) # 榜上抽一个字，比如 "真"
print(next_ch)           # 拼回去，接着接龙
```

第 3 行就是本讲主角公式，第 4 行把便签翻译成概率榜。请把**文字接龙**这个词用荧光笔标出来——它将一路贯穿到第 11 讲：GPT 的看家本领，本质仍是“读完上文，预测下一个词”，四十年未变的任务设定；变的只是接龙的网络（RNN 换成了 Transformer）与语料的规模。生成式预训练变换

器（Generative Pre-trained Transformer）这个名字里的 G（生成式），说的就是“从概率榜里采样接龙”这件事。你在这几行代码里看到的循环结构，就是 2026 年最强大模型的直系祖先。

顺带回答一个自然会冒出的疑问：采样时要不要总取榜一？总取榜一，生成的文字四平八稳且容易复读；偶尔按概率抽冷门字，文字才活。这个“抽签温度”的旋钮，留到第 11 讲再拧。

## 9.4　BPTT：把展开图当普通网络训练

训练 RNN 的思路就一句话：**把按时间展开的那张图当成一个普通深层网络，直接用第 2 讲的反向传播**。展开图里输入是 $x_1, \dots, x_T$，“层”就是一个个时间步，“深度”等于序列长度 $T$；唯一的区别是各层参数相同（共享的 $W_x, W_h, W_y$）。损失取每个时间步预测误差之和：$L = L_1 + L_2 + \cdots + L_T$（每步的 $L_t$ 是交叉熵，第 6 讲）。这套流程有个专门的名字：**沿时间反向传播**（Backpropagation Through Time，BPTT）——听着唬人，展开图一画，其实就是老朋友反向传播换了个时间维的舞台。

对共享参数求梯度有个小讲究：$W_h$ 出现在展开图的**每一层**，所以它的梯度是各时间步贡献的**总和**（$\partial L/\partial W_h = \sum_{t=1}^{T} \partial L_t/\partial W_h$），实现时把各步算出的梯度累加即可。真正的大坑在误差信号本身：要让第 $T$ 步的预测错误“教育”到第 1 步的读法，误差得沿横向箭头倒着走 $T-1$ 步——而链式法则规定，每倒一步都要乘一次因子。

**知识补齐：梯度为什么会连乘——链式法则又立功了**

回忆开课导引（第 0 讲）里那张链式法则小图：$x \to u \to y$ 复合函数的导数，等于沿箭头把各段导数**相乘**。反向传播（第 2 讲）把这条规则用到整个网络上：误差从输出层倒回第 1 层，沿路逐段相乘。普通 20 层 MLP 倒回 20 段；而 RNN 展开后深度就是序列长度，一段 100 字的语料就是 100 层。更要命的是，RNN 每段乘的因子里都含同一个 $W_h$——倒走 $k$ 步，就要乘 $W_h$ 的 $k$ 次幂的量级。**连乘**与**同一个数的幂**，这两个词凑在一起，指数级的放大或缩小就在路上了。9.5 节把这笔账算给你看。

## 9.5　梯度消失与爆炸：记忆的大敌

9.2节动手算的那两个数现在派上用场。误差信号从第 $t$ 步倒回 $k$ 步之前，幅值大约要乘 $\gamma^k$，其中 $\gamma$ 是 $W_h$（连同导数项）的“平均放大倍数”。下图画出了两种命运：$\gamma > 1$ 时梯度随距离**指数爆炸**（$1.2^{30} \approx 237$ 倍，数值冲出浮点范围，训练直接变成 NaN）；$\gamma < 1$ 时梯度**指数消失**（$0.8^{20} \approx 0.012$，走 20 步只剩 1%，走 50 步连浮点数都懒得表示它）。

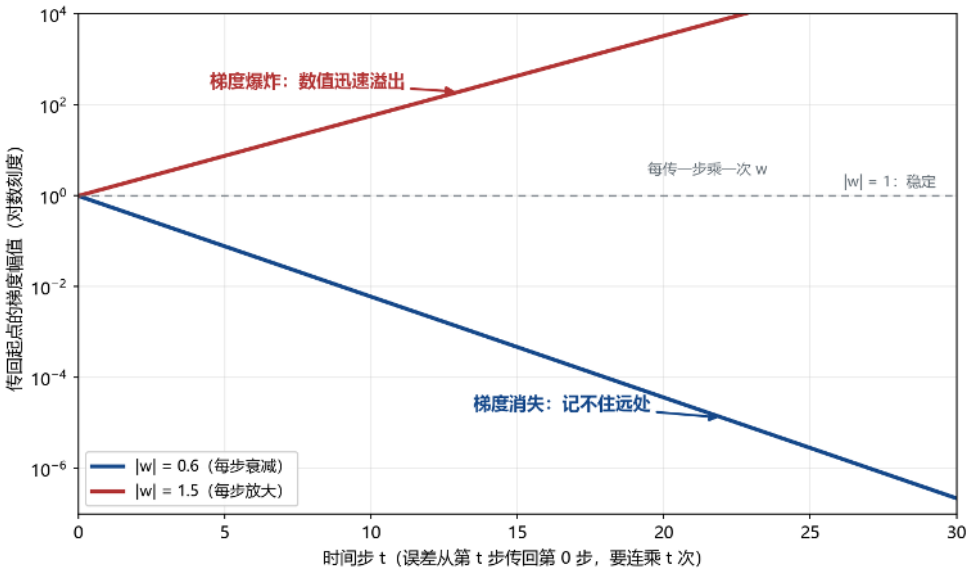
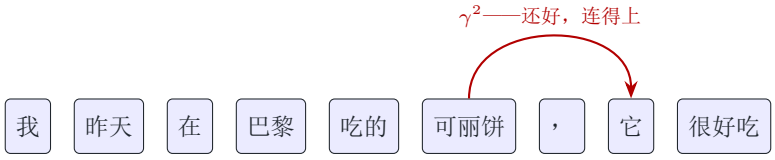



**图 9.4**　长序列上的两种命运：梯度幅值随回传步数指数变化——$\gamma > 1$ 连乘爆炸（扶摇直上），$\gamma < 1$ 连乘消失（归零）；横轴是误差要倒走的步数

两个病，一个轻一个重。**爆炸**好治，一句话的药方：**梯度裁剪**（gradient clipping）——每次更新前看一眼梯度向量的长度，超过阈值就按比例缩短，方向不变。相当于给误差信号装了个限流阀，工程上一行代码的事。**消失**难治，而且它砸的正是 RNN 的立身之本——记忆。“消失”翻译成人话：**网络记不住远处的词**。看这句话：




“它”指谁？指可丽饼。人读一遍就懂，因为你的短时记忆把“可丽饼”稳稳托到了“它”出现的那一刻。RNN 想学会这条规则，训练时误差就得从“它很好吃”倒着传回“可丽饼”，把“该把可丽饼写进便签”的读法奖励一遍。本例只隔 2 步，$\gamma^2$ 还撑得住；可真实文本里指代隔十几步、时序数据里的因果关系隔几十步是常事——按 0.8 连乘，20 步后误差信号只剩约 1%，50 步后约 $10^{-5}$：**到不了，等于没学**。这解释了 9.3 节那个现象的深层版：字符 RNN 接龙超过几十字就忘了开头设定，不是它懒，是梯度根本走不到那么远。

病根还有一个帮凶：唯一的便签 $h_t$ 每一步都被整页重写（9.2 节手算你亲眼见了），旧信息每过一站就稀释一层。想治它，就得给记忆**修一条不被反复覆写的专用通道**——这正是下一讲 LSTM 登场的理由。1997 年的两位研究者为此设计的“传送带加三扇门”，将统治序列建模二十年。

## 9.6　RNN 的应用与局限，以及 2026 年为什么还学它

把 RNN 家族能干的活儿摆一张表：

**表 9.2**　RNN 的典型应用：输入输出都是“按时间步对齐”的活儿

| 应用 | 输入 → 输出 | 例子 |
|---|---|---|
| 语言模型 | 上文 → 下一字概率榜 | 输入法联想、接龙生成 |
| 时序预测 | 过去 $T$ 步 → 下一步 | 负荷预测、温度趋势、相机参数预测 |
| 序列标注 | 每步输入 → 每步标签 | 分词、词性标注、设备日志标异常 |
| 语音与翻译 | 声波/文本 → 文本 | 2014–2016 年神经机器翻译的主力（下一讲） |

局限也在表外明摆着两条。**第一，慢**：第 $t$ 步必须等第 $t-1$ 步的便签传过来才能开算，串行天注定——第 5 讲卷积是“每个位置同时算”，GPU 几千个核心一起开火；RNN 的 GPU 只能眼睁睁看时间步一个个排队，训练快不起来。**第二，难长记忆**：上一节的梯度消失，50 步外的依赖基本学不到。这两条罪状，正是 2017 年后 Transformer（第 11 讲）取而代之的资本：全并行训练 + 注意力直达任意距离。

那为什么 2026 年的你还必须学它？三个理由，一个比一个实际。其一，**它是地基**：第 11 讲的注意力、本章末尾要讲的 Mamba，全都在跟“循环 + 状态”这组概念对话，跳过本讲去读大模型材料，处处是看不懂的黑话。其二，**小数据工业时序是它剩下的主场**：大模型要海量语料，而工厂里的传感器序列、设备状态、相机参数往往一天就攒几千步样本——这种小数据、低算力、要低延迟的场景，LSTM/GRU 至今是产线主流（下一讲给你选型表）。其三，**状态恒定**：RNN 无论读多长，随身只带一张便签，内存不随序列变长而涨——凡是“边收数据边出结果”的流式系统（逐帧相机、多轮对话 agent），都贪图这个便宜。

**前沿速递：RNN 的 2024–2026 文艺复兴——循环思想没死，换了个马甲**

Transformer 有个与生俱来的代价：注意力要两两比对，序列长 $T$，计算量就是 $T^2$（平方复杂度）——读 10 万字的文档，账单是 1 万字的 100 倍。于是 2023–2026 年，**状态空间模型**（State Space Model，SSM）高调回归：S4（2022）先用“循环 + 压缩状态”在超长序列上崭露头角，**Mamba**（2023 年底，Gu & Dao）靠“选择性状态空间”一战成名，2024 年的 Mamba-2 把理论地基夯实。看它的推理方式：每来一个新词，用一个固定大小的状态做一次递推更新——**这不就是带门控的 RNN 吗？**。Mamba 的计算量随序列线性增长（$O(T)$），状态恒定、推理如飞，2024–2026 年在基因组（百万级序列）、长文档、音频上全面开花；同门的 RWKV 则干脆长着 Transformer 的脸、跑着 RNN 的心。业内一句行话送给你：“Attention 之外，循环未死——它只是学会了先压缩、再记忆。”你在本讲学到的 $h_t$ 递推，正是这场文艺复兴的语法起点。

**项目连线：你的三个项目里，处处是时间步**

**六目拼接**：九个可学习“参数头”网络（第 一 章）里有一个叫 `temporal_state`：输入形状 $[B, T, 24]$——$T$ 个时间步、每步 24 维相机状态（匹配数、增益、运动估计等），输出预测**下一步**该走的参数。把 32 步状态序列切成帧、逐步入网、预测下一步——这正是本讲 9.1 节的 $[B, T, D]$ 与 9.2 节的循环结构在工业现场的微缩版；$T$ 只有几十步，一个小 RNN 单元绰绰有余。**水下双目**：ROV 下潜拍回的是**时序帧流**（第 二 章），伪标签过滤的重重关卡里有一关叫“时序一致”：相邻帧的视差与深度不该跳变，跳了就说明上一帧的标签不可信——判断“一致与否”，前提是记得上一帧，这就是循环网络“带便签上班”的思想。**数据基座**：对话式问数（第 三 章）要记住上下文——多轮对话不串题，靠的是一套“哪些记、哪些忘、哪些此刻说”的上下文管理机制，A 组

“结果固定不串题”的验收线，考的正是下一讲门控思想的工程版。三次实验里跑起来的 agent 与训练系统，接上时序数据时的选型判断，就从本讲与下一讲来。

## 9.7　从这一讲到 2026

9.6 节的前沿盒子讲了循环思想的文艺复兴，这里把 2026 年的战报再往前递一页，三件事值得你记住。**第一，长序列上首次反超**：FlashAttention-2 是注意力阵营引以为傲的推理标配引擎，而 Mamba-2 在超过 8K token 的长序列上，推理速度与显存效率双双反超它（2025 年的对比实验给出了硬数据）——“线性复杂度干翻平方复杂度”第一次从口号变成实测。**第二，老牌选手重装上阵**：1997 年提出 LSTM 的霍赫赖特团队 2024 年推出 xLSTM，2025 年进一步放出 70 亿参数的 xLSTM-7B——生成质量与同规模的 Transformer 语言模型打得有来有回，短板只剩训练与推理速度还落后 Mamba 系一截。**第三，最扎实的战场在基因组学**：一条 DNA 序列动辄百万级碱基对起步，注意力那本平方账在这里直接爆表；Caduceus、Hawk 等状态空间模型凭线性复杂度把超长 DNA 当序列整段“读”进去，长程基准 LRB 专门为这类生物长序列任务评测它们的线性扩展能力——“边读边记”的循环思想，正在帮生物学家读生命的天书。

所以，别把本讲当成“被淘汰者”的传记。你在 9.2 节亲手拨弄过的那张便签 $h_t$，2026 年正以 Mamba、xLSTM 之名，在 DNA 测序数据与超长文档里每天运行——**循环思想没有死，它在等算力与硬件追上它的数学**。等你在第 11 讲看完注意力的繁华，不妨回来再翻一遍本讲的展开图：两条路线争的其实是同一道题——**用有限的状态，装下无限的历史**。

本讲课后题，是亲手把“便签”跑起来——字符接龙或正弦预测二选一，顺便亲眼看一次梯度消失。五要素如下。

**第 9 讲课后编程题（用 AI 实现）**

**题目**：指挥 AI 辅助，实现一个最小 RNN 时序实验，二选一。**(A) 字符级文字接龙**：找一段 5000 字以上你喜欢的中文文本（小说、新闻、歌词皆可）存为 `corpus.txt`，训练一个字符级 RNN（隐层 64–128 足够，CPU 也跑得动），训练完从种子字“今”开始采样生成 100 个字。**(B) 正弦波预测**：生成 600 点序列 $y=\sin(2\pi t/50)$ 加少量噪声，用滑动窗口“前 20 步预测第 21 步”，训练后从序列中段起递推预测 50 步，画出“预测 vs 真值”曲线。实现可用 PyTorch 的 `nn.RNN`，但代码里必须注释标清：$h$ 在哪一步被更新、“按时间展开”发生在哪里。

**交付物**：(1) 代码文件 `rnn_toy.py`；(2) 生成样本截图或“预测 vs 真值”曲线 png 一张；(3) 50 字以上说明：模型学到了什么（接龙的语感？正弦的周期?）；(4) 一段 50 字以上的“我改了 AI 代码里的什么、为什么”。

**提示词建议**（可直接抄）：“我是零基础学生，正在学循环神经网络。请用 Python + PyTorch 帮我实现字符级 RNN 文字接龙：(1) 读入本地 corpus.txt（UTF-8），按字符建立词表；(2) 用 nn.RNN（hidden_size=128）搭一个‘输入当前字、预测下一字’的模型，交叉熵损失，Adam 学习率 0.003；(3) 训练 20 轮左右，打印每轮 loss；(4) 从种子字‘今’开始按概率采样生成 100 个字并打印；(5) 每行中文注释，并特别注明：隐藏状态 h 在哪一步被更新、‘按时间展开’发生在代码的哪里；(6) 最后用三句话回答：为什么生成到几十字以后，模型会开始忘掉开头?”（选 B 的同学把语料部分换成生成正弦序列与滑窗即可。）

**验收线**：代码能跑、loss 曲线整体下降；能对同学讲清 $h_t=\tanh(W_x x_t+W_h h_{t-1}+b)$ 里每个符号是什么、“展开”是什么意思；能当场回答两问——“RNN 为什么慢”（时间步串行，GPU 吃

不满）与“为什么记不住 50 步前的字”（梯度连乘消失，9.5 节）；至少一处自己的修改（换语料、改窗口长度 $T$、改隐层大小、把 tanh 换成 ReLU 看看会不会更糟，都算）。**代码是 AI 写的没关系，讲不清楚就是没做。**

**加分项**（任选其一）：(1) 画出梯度范数随回传步数的衰减曲线（把 9.5 节的“两种命运”亲手复现一遍）；(2) 把窗口长度从 20 加到 100，对比预测曲线，亲眼看一次“RNN 记不住长序列”；(3) 把语料中每个字的隐藏状态取出来做二维散点图（第 4 讲隐空间的手法），看看标点、数字、汉字是否各占一簇。

课后阅读：教材《深度学习模型与算法基础》第 167–186 页，与本讲内容对应；读到 RNN 展开图时，用本讲的“便签”语言把它自己讲一遍；读到梯度消失一节，对照 9.5 节的两条曲线。

**序列数据（sequence）**

次序本身携带信息的一串数据：文本、语音、视频、传感器读数；打乱次序即毁掉信息。

**时间步（time step）**

把序列切成的“一帧”，用 $t$ 编号；张量形状 $[B, T, D]$ 里的 $T$ 即时间步数。

**循环神经网络（RNN）**

同一个网络逐时间步反复使用、用隐藏状态传递记忆的序列模型；参数与序列长度无关。

**隐藏状态（hidden state）**

RNN 的“便签”：$h_t = \tanh(W_x x_t + W_h h_{t-1} + b)$，每步被整页重写。

**按时间展开（unfold）**

把循环沿时间拉开成深层网络图：深度即序列长度，是分析与训练 RNN 的通用视角。

**权值共享（时间维）**

同一套 $W_x, W_h$ 用于每个时间步；与 CNN“模板扫遍全图”的空间维共享同理。

**BPTT（沿时间反向传播）**

把展开图当普通网络做反向传播；共享参数的梯度为各时间步贡献之和。

**语言模型 / 文字接龙**

逐字读入上文、预测下一字概率分布的任务；GPT 的看家本领与其一脉相承。

**梯度消失/爆炸**

误差沿时间回传要连乘 $W_h$ 类因子：大于 1 连乘爆炸（NaN），小于 1 连乘消失（记不住远处的词）。

**梯度裁剪（gradient clipping）**

治爆炸的限流阀：梯度向量超过阈值就按比例缩短、方向不变。

# 第 10 讲　长短时记忆网络 LSTM 与 GRU

1991 年，慕尼黑工业大学有个叫霍赫赖特（Hochreiter）的学生，在毕业论文里把上一讲那个“RNN 为什么记不住远处的词”的病根算了个体无完肤——梯度沿时间连乘，小于 1 就指数消失。1997 年，他和导师施密德胡伯（Schmidhuber）发表了对症的新结构：**长短时记忆网络**（Long Short-Term Memory，LSTM）。名字起得很妙——“长的短时记忆”：还是短期记忆那一套用法，但这次能维持**很长**时间。此后二十年，这套设计统治了序列建模：2014 年神经机器翻译起飞靠它，2016 年 Google 上线神经网络翻译、错误率直降约 60%，底层是一摞 LSTM；你手机上的语音助手，2014–2017 年间几乎清一色 LSTM。2017 年 Transformer（第 11 讲）夺走了王座，但本讲你会看到，LSTM 发明的思想从未离场——它们只是换了住处。

上一讲的病历写得清楚：唯一一张便签 $h_t$ 每步整页重写，tanh 还把数值摁在 $-1$ 到 1，老信息每过一站稀释一层；梯度想跨越几十步去奖励“该记住的读法”，路上连乘到归零。LSTM 的药方两条，本讲逐条讲：**把记忆拆成两份**（一条不被反复覆写的传送带 + 一张便签），**给记忆装上闸门**（读什么、记什么、说什么，都由可学习的开关控制）。

**本讲地图**

**讲什么**：细胞状态传送带；三扇门——遗忘门、输入门、输出门；LSTM六行公式逐行人话翻译；精简版 GRU（更新门、重置门）与选型表；RNN vs LSTM 实战对比；2026 年什么场合还用 RNN 家族。**为什么值得学**：三个项目直接踩在门控思想上——六目的 `temporal_state` 头被要求“考虑门控结构”，水下伪标签的“时序一致”过滤是门控的朴素版，数据基座的多轮对话记忆管理是门控的工程版；小数据工业时序在 2026 年仍是 LSTM/GRU 的主场。**学完你能**：把 LSTM 每行公式翻译成人话；说清它凭什么记得住远处的词；在 LSTM/GRU/Transformer 之间做选型并给出理由。

## 10.1　细胞状态：一条直通远方的传送带

先复盘病根。普通 RNN 只有一份记忆 $h$，被迫身兼两职：既是**工作记忆**（算输出时要用），又是**长期档案**（要替远处留存线索）。两个职务打架：每来一个新输入，$h_t = \tanh(W_x x_t + W_h h_{t-1} + b)$ 整个被重算一遍——相当于档案每翻一页就得全文重抄一遍，抄十几页后，第一页的内容早被抄走样了。梯度消失的微观机制也在这：信息在 $h$ 里每站都要过一次 tanh 与矩阵乘，误差回传时同样的路再走一遍，连乘不歇。

LSTM 的第一刀切得极准：**把两份职责拆开**。新增一条独立的记忆主干，叫**细胞状态**（cell state），记作 $c_t$——你可以把它想象成工厂里那条**传送带**：默认动作是**原样前移**（$c_t$ 基本等于 $c_{t-1}$），只在有闸门指令时，往带上放一点新货、取走一点旧货。于是“昨天在巴黎吃的可丽饼”一旦上了传送带，就一路平移直通“它”字出现的那一刻，中途不再被抄写稀释。接下来的公式与图里你会反复看到这个分工：$c$ 走传送带，$h$ 走便签；$h$ 随时可丢可换，$c$ 只在门控批准时增删。

这一招还有个数学红利，值得你用红笔圈住：传送带的更新是**加法**——$c_t = c_{t-1} +$（小改动），而不是重写。误差沿加法通道回传，几乎原封不动地直达远方，梯度消失被釜底抽薪。眼熟吗？第 7 讲 ResNet 的残差捷径正是同一个智慧：**给信息修一条加法高速公路**。1997 年的 LSTM 与 2015 年的 ResNet 隔着十八年殊途同归——“深层网络里，让该原样通过的东西走加法通道”这一句话，值两

座图灵奖级别的洞察。

**生活化类比：便签、传送带与三道闸**

想象一条工厂流水线：底部一条**传送带**（细胞状态 $c$）载着货箱长途运输；传送带上方坐着一名质检员，手里一张**便签**（隐藏状态 $h$），随手记录当前工位的情况。流水线沿线有三道闸：第一道管旧货**取走多少**（该忘的忘），第二道管新货**上带多少**（该记的记），第三道管质检员**从带上抄多少**去写便签、对外报告（该说的说）。三道闸的开度全由质检员根据“新到货 + 便签”现场决定——记住这幅图，10.3 节的六行公式你一条都不会背错。

单元全貌见下图：那条横穿单元、几乎不打弯的长箭头就是细胞状态传送带，沿途三道闸门依次放行、写入、抄送；所有支路都从传送带上分叉或汇入，这是 LSTM 结构图最醒目的特征。

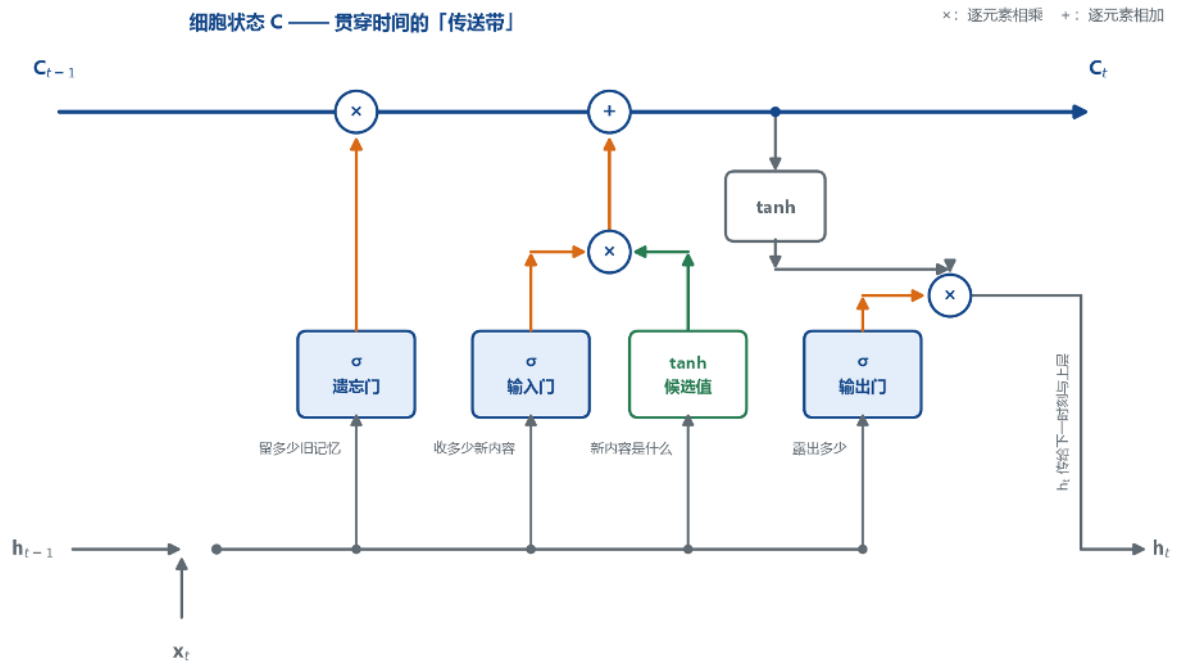



图 10.1　LSTM 单元内部全貌：底部粗箭头是细胞状态传送带（$c$ 原样前移），沿途三扇门——遗忘门（乘 $f$ 取走旧货）、输入门（乘 $i$ 放入新货 $\tilde{c}$）、输出门（乘 $o$ 抄送出 $h$）；$h$ 与 $x$ 一起决定每扇门的开度

这套“传送带 + 三扇门”的画法早已是序列建模的通用语言。下面这张经典图出自 Christopher Olah 2015 年的博客文章——它把上面那张工程版翻译成了更直观的彩色版，一代工程师是读着它入门的：

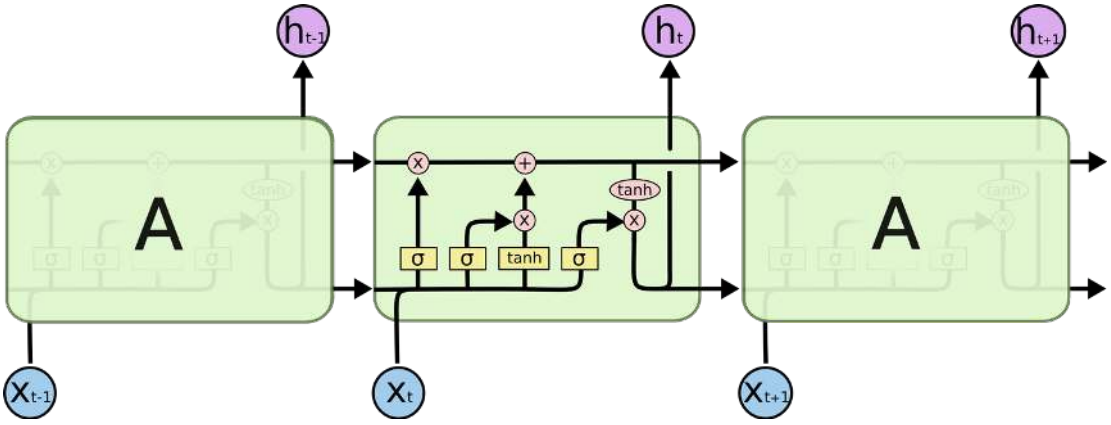



**图 10.2**　LSTM 门结构的经典画法：上方水平主线即细胞状态传送带，三列黄色模块依次是遗忘门、输入门、输出门，各门输出与传送带逐元素相乘

图片来源：Christopher Olah, "Understanding LSTM Networks" 博客文章, 2015

## 10.2　三扇门：该忘的忘，该记的记，该说的说

先说什么是**门**（gate）。每扇门就是一个 sigmoid 小网络（一层全连接）：拿两样东西当输入——**新来的输入** $x_t$ 和**旧便签** $h_{t-1}$——输出一个 0 到 1 之间的数。这个数的含义是**闸门开度**：0 全关，1 全开，0.5 放半。因为 sigmoid 连续可导，"开多大"本身是可以被梯度学出来的：网络训练时自己决定每一步该开几分。三扇门各司其职：

- **遗忘门**（forget gate）——**该忘的忘**。像水库的**水闸**：传送带上哪些旧货价值耗尽了（话题换了、任务完成了），开闸放走；开度接近 1 就是一点不忘，接近 0 就是全忘。
- **输入门**（input gate）——**该记的记**。像**安检门**：新信息先在门口打分，值钱的才放上传送带、上多少份量；无用的杂讯在门口就被拦下。
- **输出门**（output gate）——**该说的说**。像**录音笔的红灯**：档案里什么都记着，但此刻要不要放出去、放多少，红灯说了算——细胞状态满载，便签上只写当前需要的那部分。

**知识补齐：sigmoid 是"软开关"——为什么不用阶跃**

门的本质是开关，可开关的数学版——阶跃函数（0 或 1 二选一）——在第 2 讲就见过它的死穴：不连续、不可导，梯度传不过去，学不了。sigmoid（第 2 讲激活函数）是阶跃的"软化版"：平滑地从 0 爬到 1，处处可导。于是"开或关"变成"开几分"，硬决策变成软比例，梯度能顺着门开度一路回传，把"什么时候该开大、什么时候该关小"学成参数。深度学习里反复出现这一手：想要离散决策，就用 sigmoid/softmax 的连续版来近似，训练完再（需要时）取整。

还有个细节别漏：门是**逐维**的。$h$ 和 $c$ 都是向量，sigmoid 输出的开度也是同长度的向量，逐个格子对应——同一时刻完全可以"可丽饼那一格全开、天气那一格全关"。这正是下一节公式里那个圈点符号 $\odot$ 的用武之地。

**动手时间：你就是 LSTM——手动拨三扇门**

手机语音助手连听五句话。请给每一步的三扇门填开度（0–1），并写一句理由，感受"门控 = 记忆管理"：

| 步 | 你听到的话 | 遗忘门（旧货取走多少） | 输入门（新货上带多少） | 输出门（此刻说多少） |
|---|---|---|---|---|
| 1 | “记住：明天十点开会” | （空带上没旧货） | | |
| 2 | “顺便记一下：买咖啡” | | | |
| 3 | “开会的事取消啦” | | | |
| 4 | “咖啡钱记我账上” | | | |
| 5 | “今天就这些，谢谢” | | | |

参考要点：步 1 输入门全开（时间信息上带）；步 2 遗忘门全开（买咖啡是另开一格新货，开会那格别动——回忆“门是逐维的”）；步 3 遗忘门对“开会”那一格开大（这条任务作废），对“买咖啡”保持全关；步 4 输入门开、输出门开始放行；步 5 任务清场，遗忘门整体开大。做完这道题，你已经用自己的手执行了一遍 LSTM——剩下的只是把这双手翻译成矩阵。

## 10.3　LSTM 逐步计算：六行公式，六句人话

上闸门了。约定符号：$\sigma$ 是 sigmoid（读“西格玛”），$\odot$（读“圈点”）是**逐元素相乘**——两个同长度向量对应格子相乘，正是“开度乘货量”。

**知识补齐：Hadamard 积 $\odot$——“开度乘货量”的数学写法**

公式里的圈点有个学名：*Hadamard* 积（哈达玛积，Hadamard product），规矩只有一条——**同形状，对应格子相乘**。三条具体规则各一句话：**向量 $\odot$ 向量**：长度相同，对应格相乘，结果还是同长度向量（本节最常用：门开度 $\odot$ 货量）；**矩阵 $\odot$ 矩阵**：行列数完全相同，对应格相乘；**向量 $\odot$ 矩阵**：按 NumPy 的“广播”规则把向量沿行（或列）铺成同形状的矩阵，再逐格相乘。注意它与矩阵乘法 $AB$（行乘列、乘完还要逐对求和）是两码事：$\odot$ 不求和、不改形状，纯粹“点对点”相乘——所以每扇门“该开哪几格、各开几分”，一个 $\odot$ 就写得明明白白。

每扇门都由同一个配方生成：$\sigma(Wx_t+Uh_{t-1}+b)$，不同的门用不同的权重。下面六行就是 LSTM 的全部，每行配一句人话：

第一组，三扇门的开度：

$$f_t=\sigma(W_f x_t+U_f h_{t-1}+b_f) \qquad \text{人话：遗忘门开几成？} \tag{10.1}$$

$$i_t=\sigma(W_i x_t+U_i h_{t-1}+b_i) \qquad \text{人话：输入门开几成？} \tag{10.2}$$

$$o_t=\sigma(W_o x_t+U_o h_{t-1}+b_o) \qquad \text{人话：输出门开几成？} \tag{10.3}$$

第二组，备货与更新传送带：

$$\tilde{c}_t=\tanh(W_c x_t+U_c h_{t-1}+b_c) \qquad \text{人话：候选记忆——这步“值得记的内容”先打个草稿} \tag{10.4}$$

$$c_t=f_t\odot c_{t-1}+i_t\odot \tilde{c}_t \qquad \text{人话：传送带 = 旧货按遗忘门比例保留 + 新货按输入门比例上带} \tag{10.5}$$

第三组，对外输出：

$$h_t=o_t\odot\tanh(c_t) \qquad \text{人话：便签 = 把传送带过一遍 tanh，按输出门开度抄出来} \tag{10.6}$$

六行里最核心的是第 5 行：$c_t=f_t\odot c_{t-1}+i_t\odot\tilde{c}_t$。**这就是“记语文”：旧记忆的每一格乘一个 0–1 的保留比例，新草稿的每一格乘一个 0–1 的写入比例，两者相加。**想让某个格子永志不忘？把

那格的 $f_t$ 学成 1、$i_t$ 学成 0——于是 $c_t$ 的那格 $= c_{t-1}$ 的那格，原样平移，一路走到天涯海角。想让某格清零？$f_t$ 学成 0 即可。梯度回传时沿这条加法通道几乎无衰减地倒走几十上百步——上一讲的 $\gamma^k$ 连乘惨案，在这里被 $f_t \approx 1$ 的通道强行叫停。

和普通 RNN 对个账：普通 RNN 一层 tanh、一份记忆；LSTM 三层 $\sigma$ 加一层 tanh、两份记忆（$c$ 与 $h$），权重矩阵从 2 套涨到 8 套——**参数约为普通 RNN 的四倍**，每步计算量同比例变贵。贵得值不值？10.5 节见分晓。训练流程与上一讲完全一样：展开、逐时间步算损失、BPTT 回传——变的只是每个时间步里发生的事。

## 10.4　GRU：两扇门的精简版

三扇门是标配吗？2014年，曹（Cho）等人在做机器翻译时提出**门控循环单元**（Gated Recurrent Unit，GRU），砍掉一扇门、取消独立传送带，性能却与 LSTM 大体相当。它的手术刀切在两处（为省版面，下列公式从略偏置项）。

第一处：把遗忘门和输入门**合并成一扇更新门**（update gate）$z_t$，并让两者**此消彼长**——旧记忆保留 $1 - z_t$ 份，新内容写入 $z_t$ 份，一个旋钮管两头：

$$z_t = \sigma(W_z x_t + U_z h_{t-1}) \qquad \text{人话：更新门——“留旧 } vs \text{ 迎新”的比例拨到几成？} \tag{10.7}$$

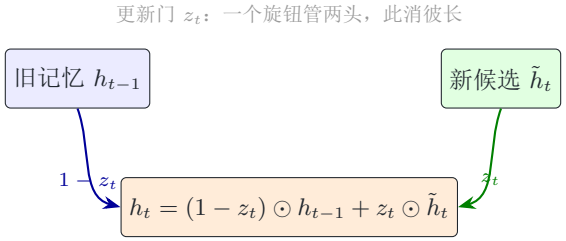
第二处：加一扇**重置门**（reset gate）$r_t$，只管一件事——生成新候选草稿时，回头看多少旧便签：

$$r_t = \sigma(W_r x_t + U_r h_{t-1}) \qquad \text{人话：重置门——打新草稿时，旧便签参考几成？} \tag{10.8}$$

$$\tilde{h}_t = \tanh(W_h x_t + U_h (r_t \odot h_{t-1})) \qquad \text{人话：候选记忆——在“被重置门过滤过的旧记忆”基础上写新货} \tag{10.9}$$

$$h_t = (1 - z_t) \odot h_{t-1} + z_t \odot \tilde{h}_t \qquad \text{人话：新记忆 } = \text{旧记忆按比例混入新货} \tag{10.10}$$

看最后一行（几何直观见下图）：$h_t$ 是“旧记忆”与“新候选”的**加权平均**，落在两点连线上的某处——$z_t = 0$ 则整根记忆原样平移（GRU 版的“传送带”，只是焊在了 $h$ 上），$z_t = 1$ 则彻底改写。没有独立的 $c_t$，记忆的“长途档”与“工作档”合并在一根 $h$ 里，全靠更新门拨比例。



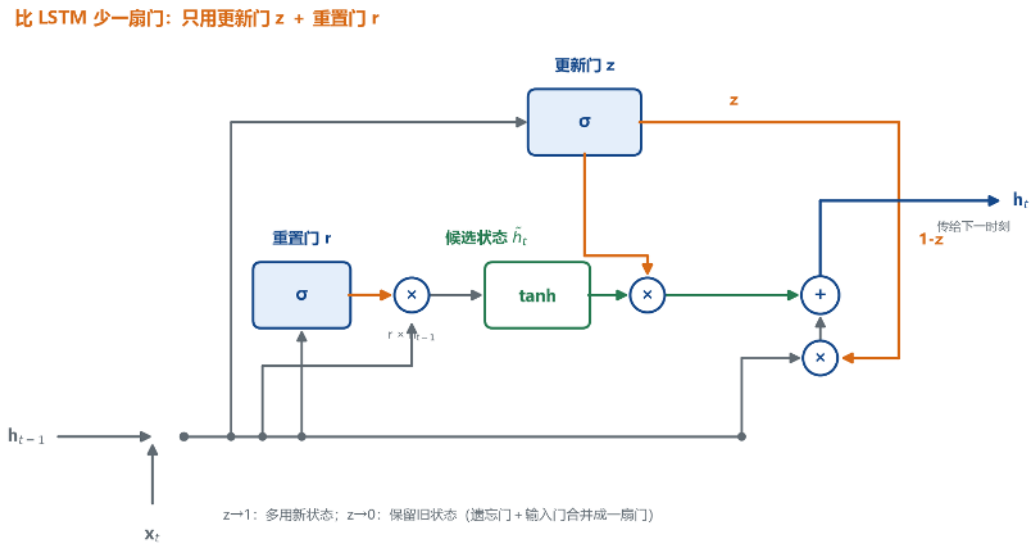



**图 10.3**　GRU 单元：只剩更新门与重置门两扇门，记忆的更新是“旧记忆与新候选的加权平均”，结构比 LSTM 简洁一截

账面变化一目了然：权重矩阵从 LSTM 的 8 套降到 6 套，**参数少约 25%**，训练更快、更不容易在小数据上过拟合。效果呢？大量实证研究的结论朴实无华：**多数任务上两者打平**，谁也不总赢。给你一张选型表，三条经验加一条最诚实的：

**表 10.1**　LSTM vs GRU 选型速查：够用就好

| 你的处境 | 倾向 | 一句话理由 |
|---|---|---|
| 数据少、想快点出结果 | GRU | 参数少 25%，小数据上更快收敛、不易过拟合 |
| 序列长、依赖远 | LSTM | 独立细胞状态的传送带更稳，长程记忆略占优 |
| 要上嵌入式、低算力 | GRU | 参数少推理快，状态内存恒定 |
| 拿不准 | 都跑一遍 | 成本低到不值得犹豫：两个都训，谁的验证集好选谁 |

表里的建议是“按处境选人”，再把两兄弟的**结构账**并排算一遍——同一道“记忆管理”的题，LSTM 与 GRU 给出了两种答案：

**表 10.2**　LSTM vs GRU：同一道题的两种答案

| 对比项 | LSTM | GRU |
|---|---|---|
| 记忆的产生方式 | 独立细胞状态 $c_t$ 当传送带，长途记忆走“专线” | 无独立细胞状态，长途档与工作档合并在一根 $h$ 里 |
| 新状态的产生 | 遗忘门 + 输入门两扇门分工：一扇管留旧、一扇管迎新 | 更新门一扇门管两头：留旧 $1-z_t$、迎新 $z_t$，此消彼长 |
| 输出调节 | 有输出门：先想好“此刻说多少”，再决定便签 $h_t$ 抄多少 | 无输出门：新便签 $h_t$ 直接全量对外 |
| 参数量与速度 | 三扇门加一份候选，共 4 套权重（约 8 套矩阵） | 两扇门加一份候选，共 3 套权重（约 6 套矩阵），少约 25%，更快 |
| 适用场景 | 序列长、依赖远时更稳 | 数据少、要快、要上嵌入式时更划算 |

一张表收拢成一句话：**LSTM 用“两扇门 + 独立传送带”换长途稳定性，GRU 用“一扇门 +**

**一根记忆”换轻快**——记住这句话，上面选型表里最难决的“拿不准”一行，你也就有了自己的判断依据。

## 10.5　实战对比：一张图看懂差距

口说无凭，上实验。同一个序列预测任务（带噪声的正弦波，模型要隔着很长的窗口抓住周期），普通 RNN 与 LSTM 各训练一次，递推预测几十步，结果见下图：普通 RNN 的预测曲线前几步还像样，越往后越飘，最后干脆自说自话地发散；LSTM 全程贴着真值走。这与上一讲的理论严丝合缝：递推预测时误差要跨越的步数越来越长，普通 RNN 的便签早已把“正弦的相位”稀释殆尽，LSTM 的传送带却把相位信息原样送到了几十步之外。

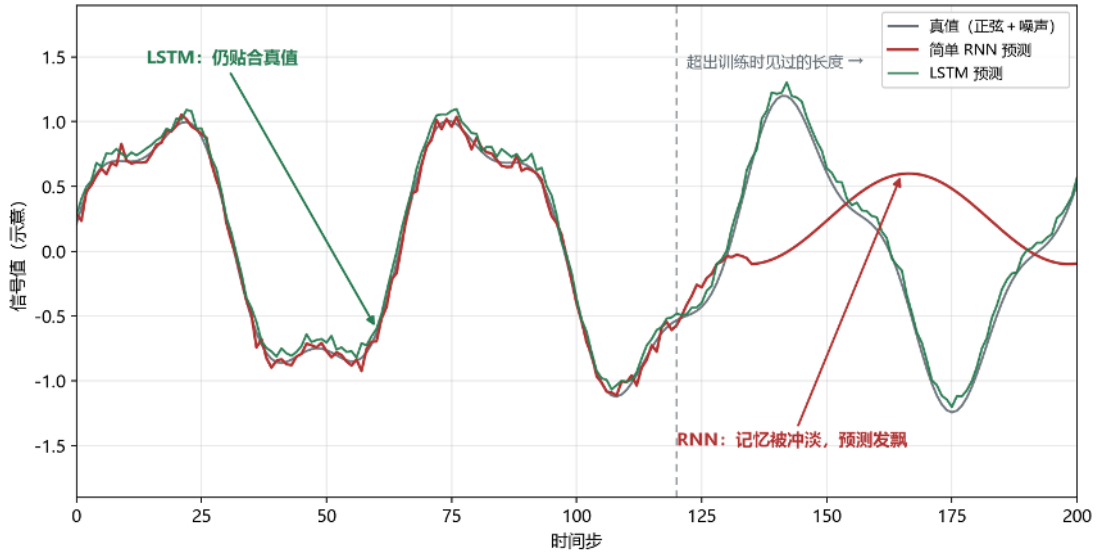



**图 10.4**　同一个长序列预测任务：普通 RNN 的预测曲线几步之后迅速漂移、发散，LSTM 全程贴合真值——传送带加门控的价值一眼可见

短窗口、短依赖的任务上，两者差距其实不大——别养成“无脑 LSTM”的迷信，10.4 节选型表才是正解。代价也要念清楚：LSTM 参数与单步计算约是普通 RNN 的四倍，嵌入式部署时这笔账不能不算。

回看上一讲引用过的那张经典展开图，现在你该看出新的门道了：

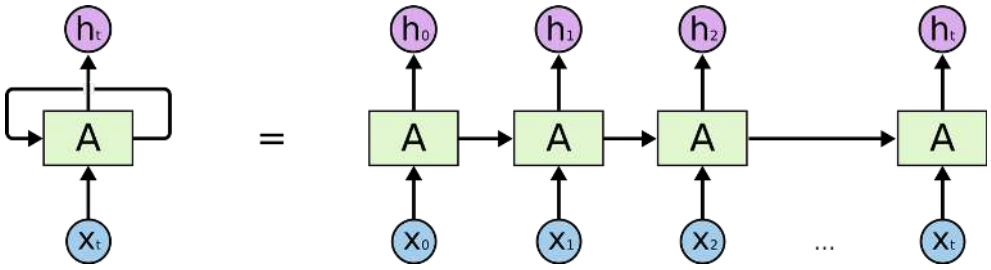



**图 10.5**　RNN 按时间展开的经典图：隐藏状态 $h$ 沿横向箭头逐站传递——普通 RNN 的这条箭头每站都被重写，换成 LSTM 后它背后多了传送带 $c$

图片来源：Christopher Olah, ”Understanding LSTM Networks” 博客文章, 2015

第 9 讲你第一次见它时，那条横向箭头只是一根“接力棒”；今天你知道：普通 RNN 把全部记忆押在这一根箭头上，LSTM 给它配了一条不被覆写的传送带当靠山。工程上还有两个常见加强，各

一句话：**双向**（bidirectional）——序列正着读一遍、倒着读一遍，两份便签拼接，适合“整句都读完了再做标注”的离线任务；**多层堆叠**——把 LSTM 像积木一样竖着摞，深层抓抽象规律。PyTorch 里一行 `nn.LSTM(input_size, hidden_size, num_layers)` 全都有，调用返回的输出里就带着 $(h, c)$ 两个状态——课后题让你亲手跑。

## 10.6　2026 年的选型决策：什么时候还用 RNN 家族

诚实的时局图先画出来：2017年 Transformer 登场后，**大规模 seq2seq 任务——翻译、摘要、对话、文本生成——已被注意力机制全面接管**（第 11 讲）；今天的旗舰大模型里找不到一根循环箭头。但“被接管”只发生在数据富饶的文本大陆。在数据贫瘠的工业地带，2026 年的版图是这样的：

**表 10.3**　2026 年序列模型选型速查：按你的数据与部署条件对号入座

| 场景 | 首选 | 理由 |
|---|---|---|
| 海量语料的文本生成/翻译/对话 | Transformer | 数据越多注意力越强，训练全并行 |
| 小数据工业时序（传感器/设备状态/遥测） | LSTM/GRU | 样本少时结构先验宝贵；模型小、训练便宜 |
| 嵌入式/端侧/在线低延迟 | GRU | 参数少 25%，单步快，状态内存恒定 |
| 流式处理（边收边算、状态要携带） | RNN家族 | 无论多长随身只带一份状态；Transformer 每步重看全历史 |

更该记住的是：**LSTM 的三个发明没有死，只是搬了家**。门控——现代架构里的门控线性单元（GLU）当激活函数用；传送带——“用加法通道保梯度”早已是标配设计（ResNet 直接受其启发）；“状态即记忆”——大模型推理时的 KV 缓存本质上就是一份可携带的状态。看 2024–2026 年挑战 Transformer 平方复杂度的那些挑战者：Mamba 的选择性状态空间，就是“带门控的循环递推”。学 LSTM 不是学古董，是学序列建模的通用语法——语法不过时，过时的只是某种句式。

**前沿速递：xLSTM 与状态空间模型的攻防战（2024–2026）**

2024 年 1 月，对，就是 1997 年那位霍赫赖特——他的团队提出 **xLSTM**：给老 LSTM 装上指数记忆容量与新式可并行的单元（mLSTM/sLSTM），公开叫板 Transformer；2025 年团队放出 70 亿参数的 xLSTM-7B，在同等规模上与 Transformer 语言模型打得有来有回。另一条战线，Mamba（2023 底）与 Mamba-2（2024）用“选择性状态空间”把线性复杂度的序列建模做成 2024–2026 年最热的方向之一，基因组、长文档、音频等超长序列任务上全面开花。2026 年中立的观察是：**超大模型与通用文本，Transformer 仍是霸主；中小数据、超长序列、端侧部署，循环/SSM 系在持续反攻**。而工业界每天在产线上跑的传感器时序预测——风场、油田、ROV 遥测、相机状态——至今大量是 LSTM/GRU。你手里的 `temporal_state` 序列与遥测行，正是这几条路线竞争的主战场。

**项目连线：门控思想的三张项目入场券**

**六目拼接**：任务书对 `temporal_state` 头的明确要求是“考虑门控结构”（第 一 章）——输入 $[B, T, 24]$ 的相机状态序列（匹配数、增益、运动估计等），预测下一步参数。落地时你会遇到本讲全部选型题：$T$ 取几步？LSTM 还是 GRU？门控怎么帮忙忽略“误匹配突刺”那几步（输入门把异常帧的新货拦在门外）？8 个组各自的选择与理由，就是一场现成的选型表实操。**水下双目**：伪标签过滤的“时序一致”一重（第 二 章）是门控的朴素版——上一帧标签可信度低（遗

忘门开大、旧货放走），这一帧的“新知识”才获准进来；更进一步，ROV 的 OSD 遥测行是典型时序文本，用 GRU 给遥测趋势建模，反过来能给伪标签的“稳定性”打分。**数据基座**：对话式问数的记忆管理（第 三 章）把三扇门翻译成了工程语言——遗忘门：把三分钟前的废话题降权出上下文；输入门：把本轮确认的数据表名、口径写进上下文；输出门：只暴露与当前问题相关的记忆给回答模块。A 组“多轮对话不串题、结果固定”的验收线，考的就是这套机制想不想得清楚。实验四（12 月 21 日）做在线优化时，你的系统要边收新数据边更新自己的认知——那一刻你就是在给系统拨门。

## 10.7　从这一讲到 2026

LSTM 的王座 2017 年交给了 Transformer，但它的“退居二线”退得很有尊严。2024–2026 年，时序预测领域冒出一批**时序基础模型**：谷歌的 TimesFM（思路就是“时序版 GPT”——解码器结构，用上千亿个真实时间点喂出来的）、亚马逊的 Chronos 家族（2025 年 10 月刚更新到 Chronos-2）、Salesforce 的 Moirai（原生支持温度、节假日这类外部协变量，工业数据场景常被首选）。它们的共同卖点是**零样本预测**——没见过你的数据，拿来就能预测，在多项公开数据集上已开始追平甚至超越传统模型；学术界 2025 年也出现了拿它们与“含 LSTM 的传统基线”正面对比的评估研究，工业过程建模的专用基础模型到 2026 年仍在活跃发展。

可把镜头切回工厂车间，画风立刻不同：传感器数据流、嵌入式设备、低功耗端侧，2026 年大量服役的仍是 LSTM/GRU，理由朴素得就像 10.4 节那张选型表——**小、快、省**。研究与工业实践共同给出的选型判断，一句话说完：**数据多、算力足、要跨域泛化，上时序基础模型；数据少、要在单片机上实时跑，GRU 依然是首选**。所以“该记的记、该忘的忘”这门门控手艺，2026 年还在风场、油田、ROV 遥测和你的单片机里每天运行——学 LSTM 不是学古董，是学小数据时序问题的看家本领。

本讲课后题，是让普通 RNN 与 LSTM 在同一条赛道上打一场擂台赛——你当裁判。五要素如下。

**第 10 讲课后编程题（用 AI 实现）**

**题目**：指挥 AI 辅助做一组对比实验。生成 600 点序列 $y = \sin(2\pi t/50)$ 加 8% 噪声；滑动窗口“前 40 步预测第 41 步”；分别训练**简单 RNN**（`nn.RNN`）与 `LSTM`（`nn.LSTM`）各一次，隐层同为 64、学习率同为 0.001、训练轮数相同；然后各自从序列中段起**递推预测 60 步**（预测出的点拼回输入接着预测），画出“真值 vs RNN vs LSTM”对比曲线一张、两条训练 loss 曲线一张，并用 50 字以上写结论：LSTM 赢在哪、代价是什么（参数量、耗时各打印出来对比）。

**交付物**：(1) 代码文件 `rnn_vs_lstm.py`；(2) 对比曲线 png 与 loss 曲线 png 各一张；(3) 100 字以上结论（谁贴合、谁漂移、差多少）；(4) 一段 50 字以上的“我改了 AI 代码里的什么、为什么”。

**提示词建议**（可直接抄）：“我是零基础学生，正在学 LSTM。请用 Python + PyTorch 写一个对比实验：(1) 生成 600 点序列 y = sin(2*pi*t/50) + 0.08 倍高斯噪声；(2) 滑动窗口：用连续 40 个点预测下一个点，整理成训练数据；(3) 分别用 nn.RNN 与 nn.LSTM（hidden_size=64，Adam 学习率 0.001，训练 50 轮）各训练一个模型；(4) 从第 300 点起各自递推预测 60 步（把预测值拼回输入继续预测）；(5) 画两张图：预测与真值对比曲线、两条训练 loss 曲线，保存为 png；(6) 打印两个模型的参数量与训练耗时；(7) 每行中文注释，并用 50 字解释：为什么递推预测越长，简单 RNN 漂得越远？”

**验收线**：两张 png 生成且现象与 10.5 节一致（LSTM 贴合真值、RNN 漂移或幅度跑偏）；能对着打印出的参数量回答“LSTM 多在哪”（三扇门多出的那几套矩阵）；能对同学讲清“递推预测”与“单步预测”的区别；至少一处自己的修改（把窗口从 40 加到 80、换成 GRU 三方对比、去掉噪声看两条曲线怎么变，都算）。**代码是 AI 写的没关系，讲不清楚就是没做。**

**加分项**（任选其一）：(1) 加入 GRU 打三方擂台，三色曲线一张图，用 10.4 节选型表的语言点评裁判结果；(2) 窗口长度取 10/20/40/80/160 各跑一次，画“窗口越长、RNN 退化越狠”折线，给上一讲的梯度消失一张实验证据；(3) 把噪声从 8% 加到 30%，观察哪个模型先崩，并用门控的语言解释为什么。

课后阅读：教材《深度学习模型与算法基础》第 187–203 页，与本讲内容对应；读 LSTM 结构图时，用“传送带 + 三扇门”的语言把每条箭头讲一遍；读 GRU 一节时，对照 10.4 节的加权平均图自己推一遍公式。

**LSTM（长短时记忆网络）**

用“细胞状态传送带 + 三扇门”治住梯度消失的循环网络，1997 年由 Hochreiter 与 Schmidhuber 提出。

**细胞状态（cell state）**

LSTM 的记忆主干 $c_t$：默认原样前移、只在门控批准时增删，信息可平流直达远方，梯度沿加法通道几乎不衰减。

**门（gate）**

一个 sigmoid 小网络输出 0–1 的“开度”，逐元素控制记忆的取走/写入/放出；连续可导，开度本身可学习。

**遗忘门（forget gate）**

决定旧记忆每格保留几成；$f_t \approx 1$ 永志不忘，$f_t \approx 0$ 一笔勾销。

**输入门（input gate）**

决定候选记忆每格写入几成——该记的记，杂讯拦在门外。

**候选记忆（candidate）**

本步“值得记的内容”草稿 $\tilde{c}_t$，经输入门审核后才上传送带。

**输出门（output gate）**

决定细胞状态向外（给 $h_t$）暴露几成——档案全记着，但此刻只说该说的。

**GRU（门控循环单元）**

LSTM 精简版：更新门一肩挑“留旧/迎新”（加权平均），重置门管草稿参考多少旧记忆；参数少 25%，效果大体相当。

**更新门（update gate）**

GRU 里此消彼长的一扇门：旧记忆乘 $1-z$、新货乘 $z$，一个旋钮管两头。

**重置门（reset gate）**

GRU 里管“写新草稿时回看多少旧便签”的门。

# 第 11 讲　注意力模型

**本讲地图**

**本讲讲什么：**注意力机制（Attention）与 Transformer——今天所有大模型共用的那颗“心脏”。**为什么值得学：**你天天在用的 GLM、DeepSeek、ChatGPT，以及你们三个项目里的大模型和立体匹配模型，核心零件都是本讲这一个。**学完你能做什么：**看懂 Q/K/V 与自注意力的每一步矩阵运算，读懂 Transformer 结构图，理解大模型为什么能当 agent 配工具；课后编程题带你把注意力画成热图“亲眼看见”。**课后阅读：**许庆阳《深度学习模型与算法基础》第 204–229 页。

如果你在手机上玩过文字接龙，就知道规则：给几个字，往下补一个最像样的字。其实大模型的底牌就是文字接龙——第 9 讲我们已经见过“今天/天气/很/好”式的逐字预测：模型每一步都在给候选字排概率。但把“接龙”从能接三个字，接到能写代码、做翻译、陪你聊天的 ChatGPT，中间隔着一座大山。这座山的名字，叫**注意力（Attention）**。

2017 年，八位谷歌工程师联名发表了一篇论文，标题读起来就像向旧时代宣战：*Attention Is All You Need*——“注意力就是你所需要的一切”。论文宣告：RNN 也好、第 10 讲苦心设计的三扇闸门 LSTM 也好，做序列任务都可以不要了；一台纯粹由注意力搭起来的机器（就是本讲的主角 **Transformer**），当年就把机器翻译做到了世界第一。近十年后的今天，从你手机里的输入法联想，到数据中心里上万张显卡训练的万亿参数大模型，全是这台机器的徒子徒孙。这一讲，我们就把它的心脏剖开看个明白。

## 11.1　为什么要注意力：从 Seq2Seq 的“失忆症”说起

先回忆第 9 讲的**序列到序列（Seq2Seq，Sequence to Sequence）**架构：编码器把整句中文“读”进去，压成一个固定长度的向量；解码器再从这个向量出发，一个词一个词地“写”出英文。这套架构漂亮，但有个先天缺陷：**不管输入是 5 个词还是 500 个词，统统要压进同一个固定大小的向量**。5 个词还好，500 个词就像让你听完一整节课、只许记一张便签纸——开头讲了什么，写到后半段早忘了。我们管这叫长句的“失忆症”，它的病根和第 9 讲讲过的梯度消失一脉相承：信息要一个时间步一个时间步地传，传得越远，丢得越多。

再看人是怎么做的。让你把“我昨天在实验室修好的那台相机”翻译成英文，写到 *camera* 这个词的时候，你的眼睛会怎么办？你会**回头看一眼原句**，目光直接落在“相机”两个字上——你并不是靠死记硬背整句话，而是“写到哪儿、回头看哪儿”。人处理长句靠的是这种随取随用的回瞄，而不是一张越塞越满的便签。

2015 年，Bahdanau 等人第一次把这种“回头看”教给了机器：解码器每写一个词，都回头对原句所有词“扫一眼”，按相关度分配注意力，把相关词的信息多拿一点、无关词少拿一点。这一招让长句翻译质量猛涨。但它当时是挂在 RNN 身上的外挂——真正的大戏在 2017 年：既然“回头看”这么好使，**干脆把 RNN 也扔了，只留注意力**。这就是《Attention Is All You Need》的野心。

先补一口数学。好消息是：整台 Transformer 的核心运算，你在第 0 讲就学过了。

**先补一口数学：点积、矩阵乘法与 softmax**

翻回开课导引（第 0 讲）的向量一节：两个向量的**点积（dot product）**＝对应元素相乘再相加，它衡量两个向量“方向有多一致”——方向越接近，点积越大；互相垂直，点积为零。本讲只需要再往前走一小步：

**矩阵乘法就是“把很多点积排成一张表”。**$A$ 是 $m\times n$ 矩阵、$B$ 是 $n\times p$ 矩阵，则 $AB$ 是 $m\times p$ 矩阵，其中第 $(i,j)$ 格 $=A$ 的第 $i$ 行与 $B$ 的第 $j$ 列的点积。所以“一批向量跟另一批向量两两点积”这种事，一次矩阵乘法就全算完了。

**softmax** 是第 3 讲分类输出层的老朋友：把任意一组分数压成一组和为 1 的正数（“百分比”），分数高的占比大。请记住本讲的一个惊人事实：**自注意力从头到尾，就是几次矩阵乘法加一次 softmax，没有别的魔法。**

## 11.2　Q/K/V 三兄弟：一次“查字典”式的信息检索

注意力把“找重点”这件事拆成了三个角色，江湖人称 Q/K/V 三兄弟：

- **查询（Query，简称 Q）**：我在找什么？——当前这个词发出的“提问”。
- **键（Key，简称 K）**：我有什么标签？——每个词挂在门口的“索引卡”。
- **值（Value，简称 V）**：我实际装的内容。——真正要被取走的信息。

**查字典：Q/K/V 到底像什么**

你拿着一个问题去翻字典。你的问题是 **Q**；字典里每个词条头上的“词条名”是 **K**；词条正文是 **V**。你不会从 A 到 Z 每条都读——你按“词条名跟我的问题有多相关”来分配眼神：越相关的词条看得越久、记得越多，无关的一扫而过。最后你脑子里的收获，是各条正文按相关度**加权混合**的结果。注意力干的就是这件事，只不过字典不是按字母排序查的，而是按**向量相似度**查的——相似度怎么算？点积。全书到这儿不知第多少次出现点积了，它就是“两个向量有多像”的万能尺子。

具体怎么算？以“小猫追蝴蝶”里的“追”为例：它拿自己的 Q 去和句中每个词的 K 做点积打分，再用 softmax 把分数变成一组和为 1 的“关注度”，最后按关注度对所有词的 V 加权求和——求和结果就是“追”在这个语境下更新后的表示。三步流程如下图。

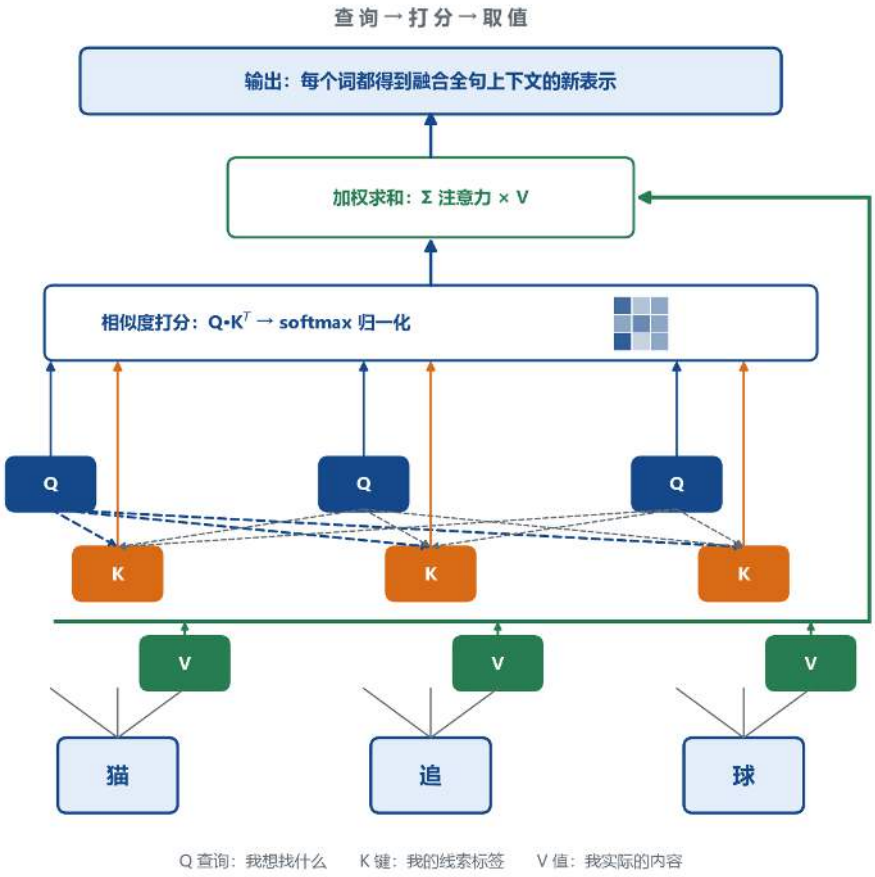



**图 11.1**　图题：注意力 Q/K/V：查询、键、值三路打分，softmax 归一化后对值加权求和

写成公式只有一行。设 $Q$、$K$、$V$ 分别是把全体词的查询、键、值向量各自摞成的矩阵，$d_k$ 是键向量的维度：

$$\text{Attention}(Q,K,V) \;=\; \text{softmax}\!\left(\frac{QK^{\top}}{\sqrt{d_k}}\right)V \tag{11.1}$$

**人话翻译：**$QK^{\top}$ 一次性算出“每个词对每个词”的相似度大表；除以 $\sqrt{d_k}$ 是给分数降温；softmax 把每行分数变成关注度百分比；最后乘 $V$，按百分比把大家的内容调和成自己的新表示。一个细节：为什么除以 $\sqrt{d_k}$？维度一大，点积动辄几十上百，softmax 会被“喂饱”成接近 0 或 1 的极端分布，梯度几乎消失；除以 $\sqrt{d_k}$ 把分数拉回温和区间，一句话，防止打分两极分化。

顺带认一下注意力的两大门派。上面这个“点积打分”的版本叫**点乘注意力**（dot-product attention），如今 Transformer 全家用的都是它；而 2015 年 Bahdanau 初版用的则是**加性注意力**（additive attention）——把 Q 和 K 拼在一起，喂给一个小的前馈网络，让它吐出一个相关度分数。打个比方：加性是“请一位小裁判逐对打分”，分打得不算差，但只能一对一对慢慢算；点乘是“全体两两点积，一次矩阵乘法一气呵成”——GPU 天生爱吃这种能全并行的菜。速度碾压之下，点乘成了现役标准，加性则留在教科书和考古现场。

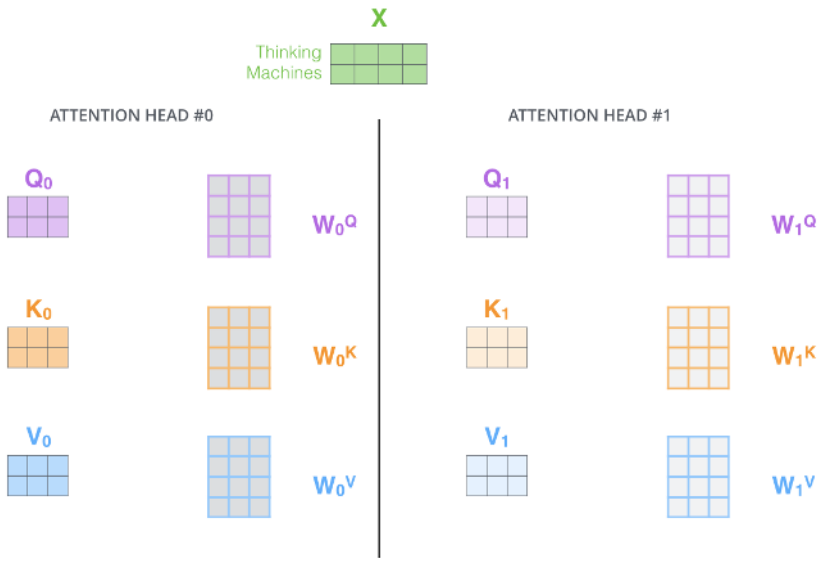



**图 11.2**　图题：缩放点积注意力的计算流程：打分、掩码、softmax、加权求和

图片来源：来源：Jay Alammar, The Illustrated Transformer 博客文章, 2018

用代码写出来不过十来行，你完全可以亲手跑一遍：

```python
import numpy as np

def softmax(x):
    e = np.exp(x - x.max(axis=-1, keepdims=True))  # 减最大值防溢出
    return e / e.sum(axis=-1, keepdims=True)

def attention(Q, K, V):
    d_k = Q.shape[-1]
    scores = Q @ K.T / np.sqrt(d_k)   # 第1步：点积打分（@就是矩阵乘法）
    A = softmax(scores)               # 第2步：softmax变成"关注度"
    return A @ V, A                   # 第3步：对V加权求和

# 三个词，词向量取2维（示意数据）
Q = np.array([[1, 0], [0, 1], [1, 1]], dtype=float)
K = Q.copy()                          # 自注意力：键就是词自己
V = np.array([[1, 2], [3, 4], [5, 6]], dtype=float)
out, A = attention(Q, K, V)
print(A.round(2))                     # 3x3注意力矩阵，每行和为1
```

跑完你会发现：$A$ 是一个方阵，第 $i$ 行第 $j$ 格就是“第 $i$ 个词分给第 $j$ 个词的关注度”，每行加起来等于 1。下一节我们把这个矩阵画成热图——那是理解注意力最直观的一张画。

**纸笔算一次注意力：三个词的“圆桌会议”**

不用电脑，拿张草稿纸。设某句话有三个词，查询向量 $q=(1,0)$，三个键、值向量如下：

| 词 | 键 $k_i$ | 值 $v_i$ |
|---|---|---|
| 小猫 | $(1,0)$ | $(1,0)$ |
| 追 | $(0,1)$ | $(0,1)$ |
| 蝴蝶 | $(1,1)$ | $(1,1)$ |

第一步，算点积打分：$q\cdot k_1=1$，$q\cdot k_2=0$，$q\cdot k_3=1$（本题取 $d_k$ 缩放不改变相对大小，从略）。

第二步，softmax：$\mathrm{softmax}(1,0,1) \approx (0.42,\ 0.16,\ 0.42)$。第三步，加权求和：$0.42 \times (1,0) + 0.16 \times (0,1) + 0.42 \times (1,1) = (0.84,\ 0.58)$。做完请回答：如果“追”的查询和“小猫”的键更垂直，输出会更靠近谁？这个三步流程，和上面 Python 代码的哪三行一一对应？

**知识补齐：训练好的 Q/K/V 到底学到了什么**

可能有人要追问：Q、K、V 这三个变换矩阵是哪来的？答案朴素得可能让你失望：**训练开始之前，它们就是三个随机矩阵**——三个刚入职的新员工，谁提问、谁打标签、谁发货，全凭胡来。所谓训练，就是让预测误差当考官：该关注哪个词没做对，就反过来微调这三个矩阵。练成之后，分工才慢慢长出来：每个词的 $Q$ 变成“这个词在查询什么语义”——“它”的 Q 专门发出“我在找我能指代的名词”的询问；每个词的 $K$ 变成“我能提供什么标签”——“猫”的 K 明晃晃写着“我是名词、我是实物、我在前半句”；每个词的 $V$ 则是“我的实际内容”——被别人取走之后，真正融进对方新表示的那部分信息。一句话：**Q/K/V 的角色分工不是设计者规定的，是梯度从数据里练出来的**——下一节热图里“它”指向“猫”的那格最亮，就是练成的证据。

## 11.3　自注意力：一句话内部的“圆桌会议”

上面的流程里藏着一个关键设定：Q、K、V 全都来自**同一句话**的词向量——每个词既是提问者，又是被查的字典，还是被取走的内容。所以这一招叫**自注意力（self-attention）**：一句话里的所有词同时互相打分、互相取经，像开了一场圆桌会议，每人发言一次，所有人都能按相关度吸收别人的信息。

自注意力一出场，两个老大难当场解决。**第一，距离不再是问题**：第 1 个词和第 100 个词直接握手，不像 RNN 要一站一站传话传 99 站（第 9 讲那张梯度一路衰减的图还历历在目）。**第二，指代消解（coreference resolution）**——代词“它/他/她”到底指谁——变成了一张看得见的表。看下图：“它”那一行，最亮的格子正好落在“猫”那一列：模型不知道语法书上的条文，但注意力分数诚实记录了“它”在向“猫”取经。

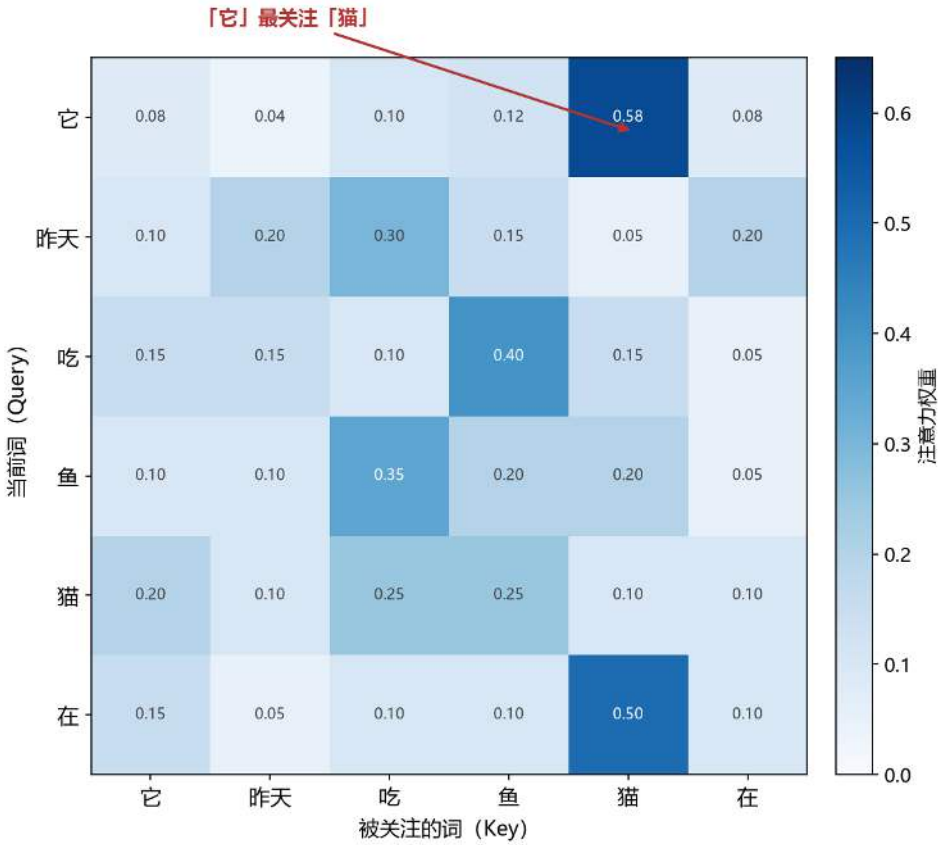



图 11.3　图题：注意力权重热图：“它”那一行最亮的格子指向“猫”，指代消解被看得见地记录下来

这类热图也是调试序列模型的重要工具：哪两个词的联系没被模型抓住，热图上一眼就能看出来。课后编程题就让你亲手画一张。

## 11.4　多头注意力与位置编码：一支足球队而非一个球星

### 11.4.1　多头注意力：八支荧光笔各画各的重点

只做一次注意力，等于只用一支荧光笔在句子上画重点——画得再好也只是“一种眼光”。**多头注意力（multi-head attention）**的思路：把词向量切成 8 份，每份独立做一遍 Q/K/V 注意力，这叫 8 个“头（head）”；最后把 8 个头的输出拼接起来再做一次线性变换。实践中发现，各个头真的会**自发分工**：有的头盯着语法结构（主语配谓语），有的头盯着语义指代（“它”配“猫”），有的头只关心相邻词。像你做阅读理解时同时用“找主语”“找指代”“找关键词”三支荧光笔——多支笔并行，互不干扰，最后一起交卷。

### 11.4.2　位置编码：给天生的“脸盲”贴工位号

但注意力有个天生的缺陷：它**不认顺序**。把“猫追狗”打乱成“狗追猫”，两句话算出来的注意力分数集合一模一样——因为在注意力的世界里，每个词只是一张写着内容的卡片，谁来谁往它根本不关心。这可不行，“猫吃鱼”和“鱼吃猫”是天壤之别。解决办法是给每个词贴一张**位置编码（positional encoding）**的“工位号”，把位置信息掺进词向量再进注意力。三代方案一句话说完：2017 年原版用不同频率的正弦、余弦函数生成位置信号；BERT、GPT-1 用的可学习位置向量（每个位置一个可训练向量，简单粗暴）；如今主流大模型（LLaMA、Qwen、DeepSeek 等）标配的 **RoPE（Rotary**

**Position Embedding，旋转位置编码**）把位置“旋转”进 Q 和 K 的乘法里，让模型外推到更长的上下文时更稳——知道名字、知道它管什么即可。

## 11.5　Transformer 全家福：用注意力搭一整栋楼

把上面的零件装配起来，就是 2017 年论文里的 **Transformer**，整体结构见下图。

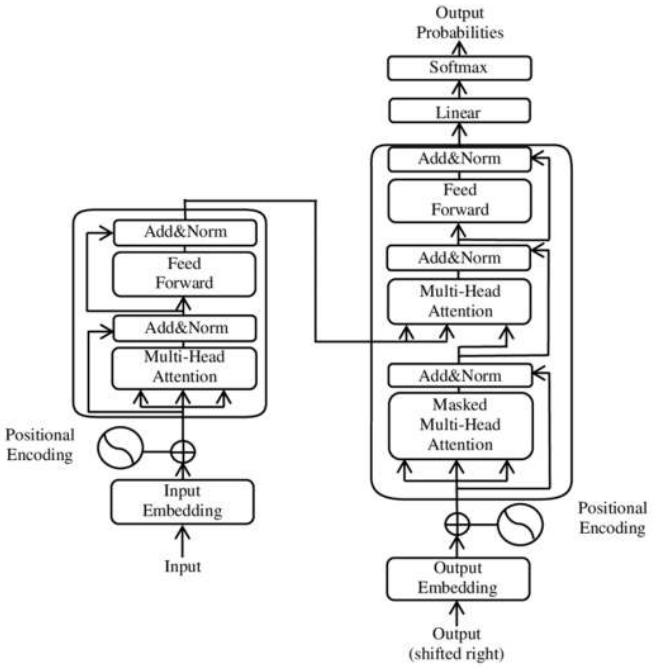



**图 11.4**　图题：Transformer 整体结构：左侧编码器栈与右侧解码器栈，各由多层相同的块堆成

图片来源：来源：Vaswani et al., Attention Is All You Need, NeurIPS 2017

左边一摞是**编码器（encoder）**，右边一摞是**解码器（decoder）**，各叠 6 层，每层是一个标准化的“块（block）”。拆开一块看看，你会发现全是老朋友：一个多头自注意力子层，加一个逐词的前馈网络 MLP（第 2 讲的全连接），每个子层外面还套着**残差连接**与**层归一化（LayerNorm）**——残差捷径正是第 7 讲 ResNet 那条“高速公路”，让几十层堆得起来；层归一化是第 6 讲批归一化的表亲，改为沿特征维度归一化、不依赖同批样本，对序列任务更友好。一句话记住它：**Transformer 块 = 注意力 + MLP + 两条捷径**。

**知识补齐：LayerNorm vs BatchNorm——一个按人、一个按科**

第 6 讲学过批归一化（BatchNorm，简称 BN），现在让它和层归一化（LayerNorm，简称 LN）当面比一比。想象一张 $3 \times 6$ 的成绩单：3 名学生，每人 6 门课。**BN 是“按科”统计**：对每一门课，跨 3 名学生算均值和方差，再把这门课的成绩拉齐——它回答“这门课在全班里算高还是算低”。**LN 是“按人”统计**：对每一名学生，跨自己那 6 门课算均值和方差，给自己的成绩单单独拉齐——它回答“这个人的整体水平如何”。差别看着温和，后果却要命：BN 的统计量依赖“同科的一批人”，**batch 一小就失灵**——批里只剩一两个人，“全班均值”毫无意义，推理时单条样本更是无班可依；LN 对每个样本自给自足，**batch=1 也照常工作**，句子长短不齐也毫不在乎。NLP 恰恰是“句子变长变短、batch 时大时小”的重灾区——这就是 Transformer 全线选 LN 的原因。一句口诀收尾：**BN 沿 batch 维统计（按科），LN 沿特征维统计（按人）**。

解码器比编码器多一样东西：**交叉注意力（cross-attention）**——它的 Q 来自解码器自己，K 和 V 来自编码器的输出，相当于翻译时“回头看”原文的那一眼。这一眼的神奇之处在于“对齐”是自己长出来的：把 *Hello World* 翻译成“你好，世界”，解码器写到“世界”这个词时，交叉注意力的目光会稳稳落在 *World* 上，写到“你”时落在 *Hello* 上——传统机器翻译要靠语言学专家手工整理“词对齐表”，这里一套注意力自己就学会了。另外解码器的自注意力是“因果”的：生成第 $t$ 个词时不许偷看第 $t$ 步之后的词，不然就是抄答案。实现上是一张**下三角掩码**：把所有“未来位置”的注意力分数统统写成 $-\infty$，softmax 一算（$e^{-\infty}=0$），这些位置的关注度恰好归零——像一场只许翻看已答页面的考试，后页一律盖住。

按“拆掉哪半栋楼”，后续模型分成三大家族，你天天听说的模型都能对号入座：

**表 11.1**　Transformer 三大家族：都拆自同一栋楼

| 家族 | 代表模型 | 注意力方式 | 擅长任务 |
|---|---|---|---|
| 只用编码器 | BERT | 双向，前后都看 | 理解：分类、检索、匹配 |
| 只用解码器 | GPT系列 | 因果，只看上文 | 生成：写作、对话、代码 |
| 编码器–解码器 | T5、原版 | 编码读入 + 交叉注意力 | 翻译、摘要等“转换” |

ChatGPT、GLM、DeepSeek 走的都是“只用解码器”路线：模型本质在玩文字接龙，接得足够好，智能就涌现出来了（马上讲到）。

Transformer 还有一张压过 RNN 的王牌：**并行**。RNN 第 $t$ 步必须等第 $t-1$ 步算完，天然串行，GPU 有一身力气没处使；自注意力把“所有词对所有词”的打分变成一两次矩阵乘法，整句话同时算完，显卡满载奔跑。**并行意味着训练快，训练快意味着敢把模型做大**——这就引出了本讲最壮观的情节：堆规模。

## 11.6　规模定律：从 GPT-1 到 2026 年的大模型

2018年起，OpenAI 沿“只用解码器”路线一路堆料：GPT-1 有 1.17 亿参数，GPT-2 涨到 15 亿，GPT-3 直接跳到 1750 亿，下图是对数坐标下的这条增长线——注意纵轴每格代表 10 倍，图上看着平缓，实际是天文数字式的爆炸。到 2026 年，前沿大模型的参数规模已进入万亿级（且多以混合专家方式组织，见下面的前沿盒子）。

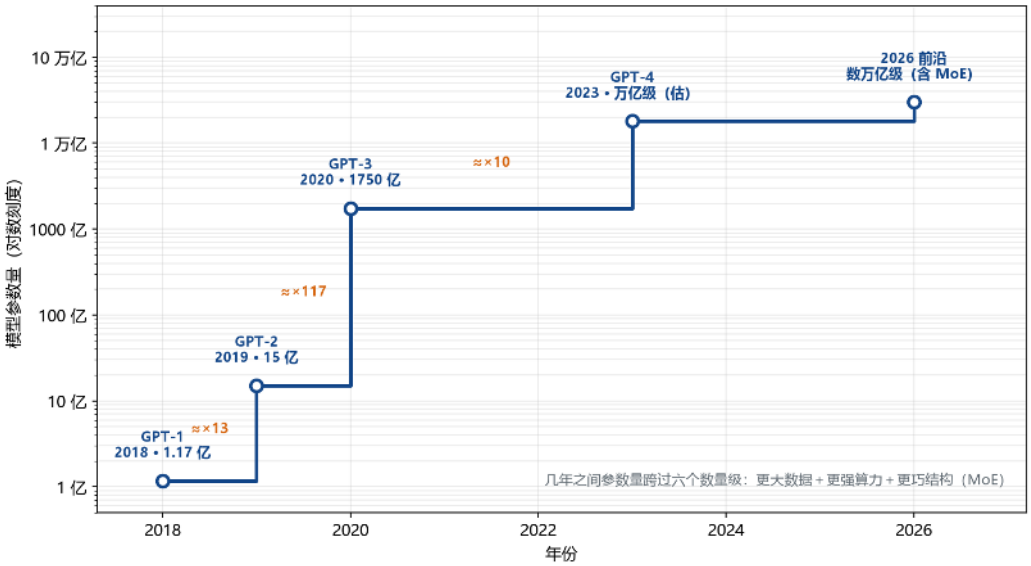



**图 11.5**　图题：大模型参数量的对数增长时间线：GPT-1 到 2026 年前沿模型

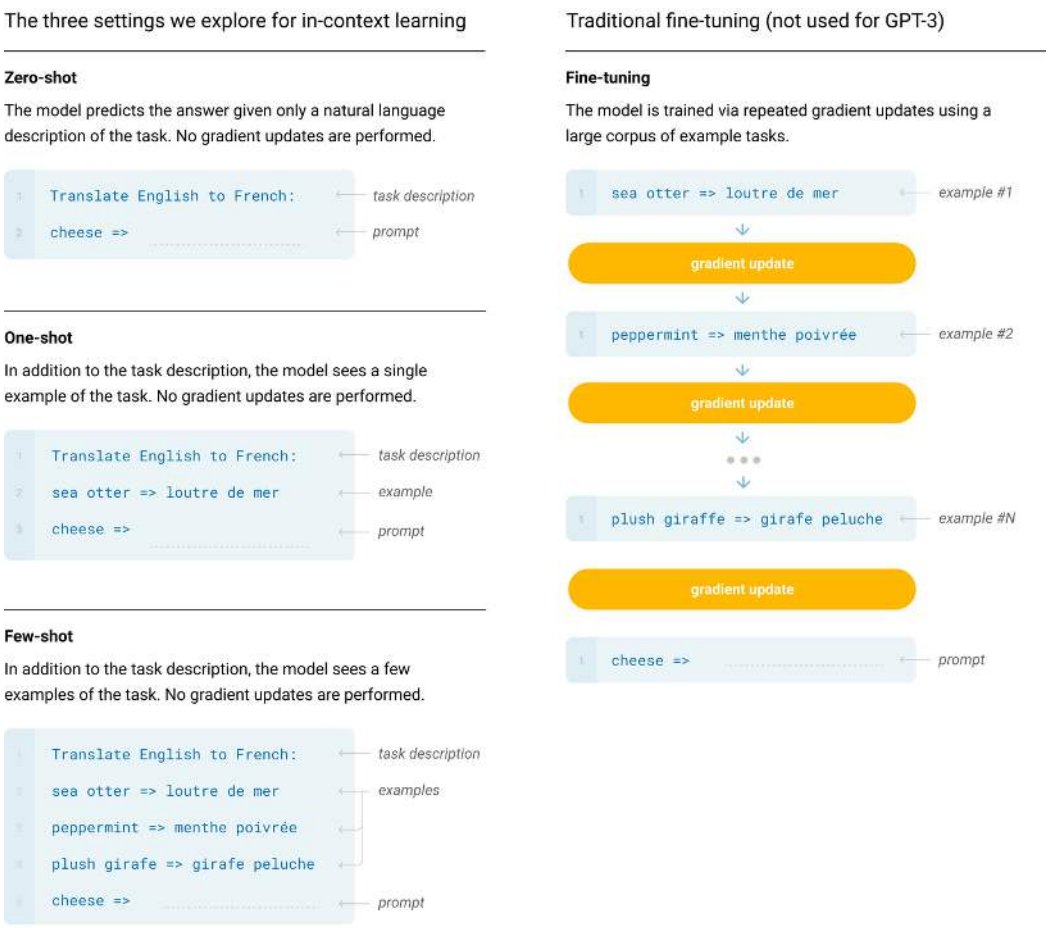



**图 11.6**　图题：GPT 系列的演化：结构一脉相承，规模逐代暴涨

图片来源：来源：Brown et al., Language Models are Few-Shot Learners (GPT-3), NeurIPS 2020

光堆参数还不够，还得喂够数据和算力。2020 年的规模定律（scaling laws）给出一条冷静的经验规律：只要模型、数据、算力按比例一起加大，模型损失会沿幂律曲线**可预测地**下降——“大力”

真的能出“奇迹”，而且出多少奇迹可以提前算出来。这给了资本下注的底气。更戏剧性的是**涌现能力（emergent abilities）**：参数规模跨过某个坎后，模型突然会做小模型完全不会的事（比如多步推理），量变到质变的机理至今仍是研究热点。

GPT-3 论文还带来一个使用方式的革命：不用再微调，把几个例子直接写在提示词里，模型“照猫画虎”就能上手新任务——这叫少样本学习（few-shot learning）。你在实验课里写的那些提示词，理论源头就是这篇论文。

文本这边的故事讲完了，图像怎么搭上这班车？答案是 2021 年的 **CLIP**。它用 4 亿对“图片 + 文字说明”做对比学习：配对的图和文在向量空间里拉近，不配对的推远。练成之后，CLIP 拥有一种奇特能力——把一张图变成一个和“词语”同类的向量。从此**图像第一次以“词”的身份进入了注意力的世界**：它可以跟文字直接做点积比相似度（零样本分类），可以被当成多模态大模型的一个“词”读进去。今天文生图模型的文本理解、多模态大模型的视觉部分，血统都能追到 CLIP。它是从本讲通向第 12 讲文生图的桥梁。

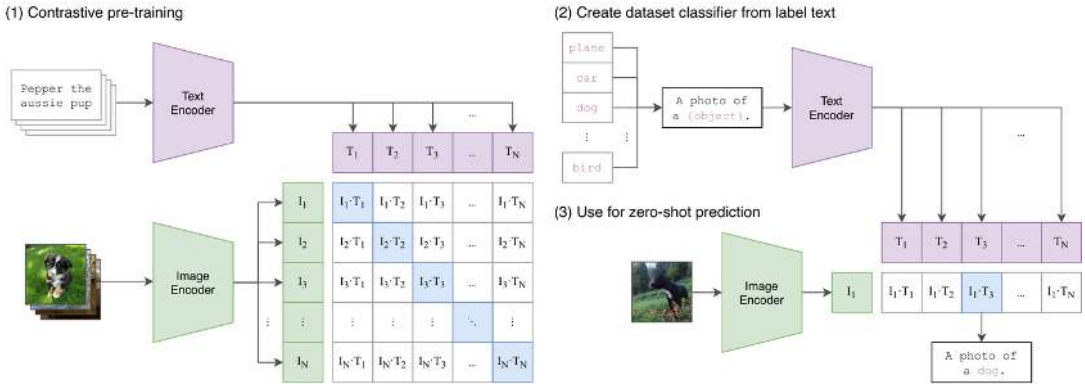



**图 11.7**　图题：CLIP 的图文对齐：同一空间里“图向量”与匹配的“文字向量”互相靠近

图片来源：来源：Radford et al., Learning Transferable Visual Models From Natural Language Supervision (CLIP), 2021

## 11.7　Agent：注意力 + 工具 = 会干活的模型

大模型光会“说”还不够，得会“干活”。让模型干活的主流方案叫 **agent（智能体）**：给模型配一套工具（读文件、改代码、执行命令、查数据库），让它在“思考”与“行动”之间循环。以你在实验一（10 月 12 日）要配置的 VS Code 编程助手 Cline 为例，它的 agent 模式是一个循环：**思考**（读任务，规划第一步）→ **调用工具**（输出“我要读某文件”）→ **观察**（框架执行后把结果塞回对话）→ **再思考**（根据观察决定下一步）……直到任务完成。

这个循环为什么成立？关键就在注意力：每一轮决策，模型都要在“系统提示 + 几十轮历史对话 + 工具返回的一大段结果”里**挑出真正要紧的几段**来决定下一步——干的正是自注意力的本职工作。工具清单写进系统提示，模型按格式输出“工具名 + 参数”，框架负责真去执行。所以 agent 没有任何超自然成分：**它就是注意力 + 一份工具说明书 + 一个循环**。第 0 讲那张“思考–行动–观察”循环图，现在你能从原理层面读懂它了。

**前沿速递：2024–2026，注意力还在进化**

- **长上下文**：前沿模型的上下文窗口已做到百万 token 级，一次读进几十本书。瓶颈在注意力本身：词数翻倍，计算量翻**四倍**（$n^2$ 增长），于是 FlashAttention、稀疏注意力等工程化改进成了标配（名词级了解即可）。

- **混合专家（MoE，Mixture of Experts）**：把一个巨型模型拆成很多“专家”小网络，每个词只激活其中一两个。例如 DeepSeek-V3 总参数 6710 亿，每个词只激活约 370 亿——**模型更大，跑起来反而更省**，是 2025–2026 年前沿模型的标配架构。
- **推理模型**：OpenAI的 o1（2024）、DeepSeek-R1（2025）学会“先想后答”：回答前先生成很长的思维链草稿，用推理时的算力换准确率，数学与编程成绩直逼顶尖人类选手。
- **多模态统一**：文本、图像、音频、视频逐渐被统一成“token”喂进同一个 Transformer——GPT-4o、Gemini 能听能看能说；视频生成模型（如 Sora）把视频切成时空小块当词来“接龙”。

你这门课用的 GLM、DeepSeek，全都长在本讲的族谱上。学完这一讲，你手里等于有一张大模型的心脏解剖图。

**项目连线：你的三个项目里全是注意力**

- **水下双目（二）**：项目采用的 FoundationStereo（CVPR 2025 最佳论文提名）是一个“用注意力看立体图对”的立体基础模型——它的骨干用注意力在左右两目特征之间反复“找对应点”，正是本讲思想与第 11 讲之外立体匹配任务的合体；你们要做的教师–学生伪标签流水线，蒸馏的正是这个注意力骨干的本事。
- **数据基座（三）**：Gemma 12B本地大模型当问数智能体，挂 8 个白名单工具（查表、算指标、画图……）——架构上就是 11.7 节 Cline 的同款“思考–调用–观察”循环；Granite 语义向量模型把文档变成向量做检索，而“把一切变成向量”正是注意力时代的通用语言。
- **六目拼接（一）**：九个“参数头”轻量路线之外，端到端路线的 RopStitch 引擎内部就含 Transformer 骨干——拼接问题被表述成“六路图像 token 互相做注意力”，全景图是注意力“开会”开出来的。
- **实验一**：你在 10 月 12 日亲手配置的 Cline agent，就是 11.7 节的落地现场。配完回头再看本节，你会有“原来如此”的快感。

## 11.8　从这一讲到 2026

2017 年那篇论文的标题，今天读来像句预言：注意力确实“就是你所需要的一切”。2026 年的 GPT、Claude、Gemini、Qwen、DeepSeek 全部构建在这一讲的自注意力之上，并且沿着三条线狂奔。**第一条线：上下文长度**。百万 token 上下文从论文指标变成了生产就绪的标配：Gemini 2.5 Pro 标配 100 万 token（企业版 200 万），Claude Sonnet 4 也达到 100 万——一次读进几十本书、一整库代码；撑起它的是前沿盒子里提到的 FlashAttention 这类注意力工程，正是它们把 $n^2$ 的账单压到付得起。**第二条线：Agent 化**。2024 年底 Anthropic 发布开放协议 **MCP（Model Context Protocol，模型上下文协议）**，业界戏称“AI 的 USB 口”——不管什么大模型、什么工具（读文件、查数据库、调接口），都按同一套插口标准对接，一个完整的 Agent 生态沿这个口长了出来；你在实验一（10 月 12 日）配置的 Cline 编程助手，就是这条线上的一员。**第三条线：把钱花在哪**。Agent 干活时，一轮就要在上下文里背上几万 token——单轮上下文可超 5 万 token、不缓存时一轮成本 0.05–0.20 美元；所以“上下文管理”与“缓存”成了大模型工程的核心手艺，相当于 11.7 节那个思考–行动循环的水电费账本。

你此刻与任何一个大模型的每一次对话，都在这一讲的数学里运行。这也是全书最值得你“常回来看看”的一讲——大模型的下一页历史，多半还写在这颗心脏上。

**第 11 讲课后编程题（用 AI 实现）**

**题目：**做一个“注意力可视化器”，亲眼看见自注意力。取句子“小明把书放回书架因为它看完了”，用 HuggingFace 的 `transformers` 库加载 `bert-base-chinese`，抽取某一层某几个头的自注意力权重（没有网络条件时，可让 AI 生成一个合理的 $6 \times 6$ 模拟注意力矩阵代替），用 matplotlib 画成热图（行列标注分词结果），找出“它”那一行最亮的格子，并用 50 字以上解释：这一格为什么亮？它符合什么语言直觉？（提示：指代消解。）

**交付物：**代码文件 `attn_vis.py` + 热图运行截图 + 50 字以上“我改了什么、为什么”。

**提示词建议：**

```
请帮我用 Python 写一个"注意力可视化器":
1. 用 transformers 加载 bert-base-chinese，输入句子
   "小明把书放回书架因为它看完了";
2. 取第 8 层的自注意力权重（outputs.attentions[7]）;
3. 挑一个注意力头，用 matplotlib 画 6x6 热图，
   行列标签用 tokenizer.tokenize 的分词结果;
4. 自动找出"它"那一行权重最大的格子，打印对应的词对。
请用中文注释解释每一步，把功能拆成小函数，方便我修改句子和层数。
```

**验收线：**代码能跑出热图；你能向同学讲清楚 Q、K、V 在这段代码里各对应哪个张量、热图的一行和一列各是什么意思；至少有一处你自己的修改（换句子、换层、把最亮格子框出来）。“代码是 AI 写的没关系，讲不清楚就是没做。”

**加分项：**换三个不同的句子（比如“它”真正指“书”的句子），对比最亮格子的移动；或把 12 层注意力逐层画出来，观察“它”的指向在第几层变得清晰。

### 注意力机制（Attention）

让模型对输入的不同部分按相关度分配“关注份额”的机制，本质是相似度打分、归一化、加权求和三步。

### Query / Key / Value

注意力的三个角色：查询（我在找什么）、键（我的标签）、值（我实际的内容），类比问题、词条名、词条正文。

### 缩放点积注意力

用点积打分、除以 $\sqrt{d_k}$ 防止 softmax 饱和的标准注意力公式，Transformer 的基本零件。

### 自注意力（Self-Attention）

Q、K、V 全部来自同一序列，句子内部互相打分取经；并行计算，任意距离直接握手。

### 多头注意力（Multi-Head Attention）

把向量分组并行做多遍注意力，各个头自发分工盯语法、语义、邻近等不同角度。

### 位置编码（Positional Encoding）

给无序的注意力补上顺序信息，方案有正弦编码、可学习向量与主流的 RoPE 旋转位置编码。

### Transformer

完全由注意力 + MLP + 残差与层归一化搭成的网络，分编码器、解码器两摞；今天所有大模型的底座。

**token 与上下文窗口**

大模型处理文本的最小单位（词或字块）；上下文窗口是一次性能“看进去”的最大 token 数。

**规模定律（Scaling Laws）**

模型、数据、算力成比例放大时，损失沿幂律可预测下降的经验规律，是“大力出奇迹”的理论依据。

**涌现能力（Emergent Abilities）**

规模跨过某个坎后突然出现的小模型不具备的能力，机理仍是开放问题。

**混合专家（MoE）**

巨型模型拆成多个专家网络，每个词只激活一小部分，做到“总参数巨大、单次开销很小”。

**CLIP**

用 4 亿图文对训练的图文对齐模型，把图像变成与文字同空间的向量，是多模态时代的桥梁。

# 第 12 讲　生成对抗模型

**本讲地图**

**本讲讲什么**：生成对抗模型（GAN）与它的“后浪”扩散模型——AI从“认东西”走向“造东西”的两代引擎。**为什么值得学**：文生图、文生视频、深度伪造与鉴伪、你们数据基座项目里每周在跑的“造假者–监考–判官”对抗训练，全是本讲思想的现场。**学完你能做什么**：看懂 GAN 的极小极大博弈与训练拉锯，说出 GAN 家族的演进脉络，理解扩散模型为什么接管了生成任务，并对“眼见为实”保持工程师式的警惕。**课后阅读**：许庆阳《深度学习模型与算法基础》第 230–248 页。

先讲一个老故事。伪钞犯印刷假钞，越印越像；警察学习鉴定，越查越严。每破获一批假钞，伪钞犯就改进工艺；每升级一次工艺，警察就升级装备——**双方在对抗中共同进化**，最后假钞逼近真钞，警察练就火眼金睛。2014 年，蒙特利尔大学的博士生 Ian Goodfellow 把这场军备竞赛写进了一个损失函数，论文叫 *Generative Adversarial Nets*（生成对抗网络，缩写 GAN）。据他本人后来在访谈里说，这个想法诞生于和同学在酒吧的一次神聊——“让两个网络互相欺骗，不就都能变强吗？”——他回家当晚就把代码写了出来，第一晚就跑通了。深度学习巨头 Yann LeCun 曾评价它是“近二十年深度学习最酷的想法”。

这一讲是全书的压轴之一。前十一讲里的网络几乎都在“认”世界：给图，吐类别。从这一讲起，网络要学会“造”世界：给噪声，吐图像。而“造假”与“鉴伪”这对矛和盾，会一路延伸到你毕业时都将面对的现实问题——深度伪造。

## 12.1　生成模型与判别模型：无中生有 vs 看图说话

先把两类模型分清楚。**判别模型（discriminative model）**做的是“看图说话”：输入一张图，输出类别——它是第 0 章到第 10 章的全部主角。它在学“图和类别的对应关系”，数学上记作 $p(y\,|\,x)$：给定图 $x$，类别 $y$ 的概率。**生成模型（generative model）**做的是“无中生有”：输入一个随机的噪声向量，输出一张图。它要学的是数据本身的样子，即 $p(x)$——世界上所有猫照片共同服从的那张“概率分布地图”，然后从地图上任一处“采样”出一张全新的、训练集里从来没有过的猫。

**什么叫“学会数据的分布”**

说模型“学会了猫的分布”，意思是：它脑子里有一张“猫概念地图”，地图上每个坐标对应一张可能的猫图（耳朵角度、毛色、姿态……），靠近地图中心的坐标是“最典型”的猫，边缘是罕见的猫。所谓“生成一张图”，就是在地图上随手戳一个点，把那个点“翻译”成像素。“这张图像不像真的猫”，就看它是否落在地图上概率高的区域。生成模型千千万，目标都是一张：把这张地图学到手。

这套“学分布”的思路在数学上站得住吗？站得住。神经网络理论里有一条万能逼近式的底气：**任一概率分布，都可以被一个足够大的可微函数逼近到任意程度**——分布无非是一张从“样本坐标”到“概率密度”的对应关系，而“逼近对应关系”正是深度网络的本行。所以“学一个网络去生成符合分布的样本”不是玄学，是数学上有保底的工程路线——本讲的 GAN、稍后的 VAE 与扩散模型

走的全是这条路，区别只在用什么手段逼网络把分布学像。

表 12.1　判别模型与生成模型对比

| | 判别模型 | 生成模型 |
|---|---|---|
| 一句话 | 看图说话 | 无中生有 |
| 输入 → 输出 | 图像 → 类别 | 噪声向量 → 图像 |
| 学习目标 | 类别边界 $p(y\mid x)$ | 数据分布 $p(x)$ |
| 代表 | CNN、ResNet、ViT | GAN、扩散模型 |
| 典型用途 | 识别、检测、分割 | 文生图、数据增广、鉴伪攻防 |

其实你在第 4 讲已经和生成模型打过照面：自编码器的解码器能从隐向量重建图像。但它只会“照着练过的样子修图”，不太会凭空造出全新的、多样化的样本。真正把“无中生有”做成一门艺术的，是接下来这场对决。

## 12.2　GAN：造假者与鉴定师的对决

GAN的架构里有两个网络，天然是对手（见下图）：

- **生成器（Generator，记作 G）**：造假者。输入一个纯噪声向量 $z$（比如 100 维、从标准正态分布随机采样），输出一张假样本。它从一开始就没人教过“猫长什么样”，全靠造假谋生。
- **判别器（Discriminator，记作 D）**：鉴定师。输入一张图（真的或假的），输出一个 0 到 1 之间的分数——它有多确信这是真货。

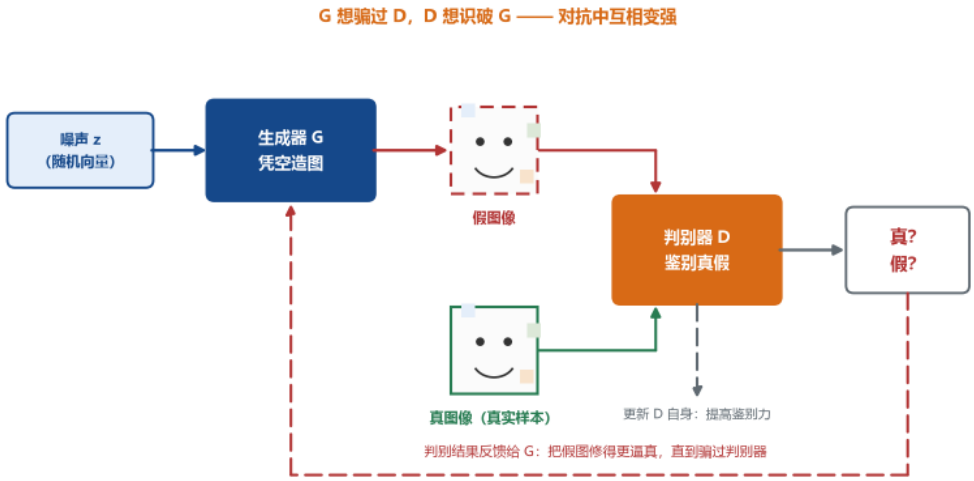



图 12.1　图题：GAN 架构：生成器把噪声变成假图，判别器给真假打分，双方在对抗中共同进化

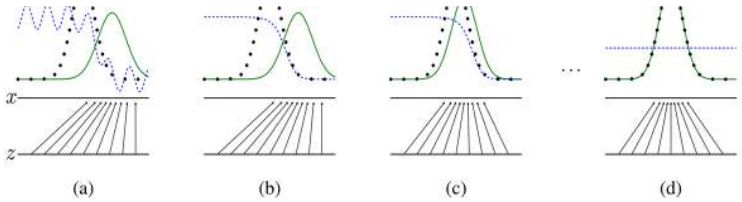



**图 12.2**　图题：GAN 论文原始框图：生成空间到样本空间的映射与判别器的双向夹击

图片来源：来源：Goodfellow et al., Generative Adversarial Nets, NeurIPS 2014

训练时两边同时开工：D 拿真图和 G 的假图混在一起练鉴定；G 拿 D 的鉴定结果当反馈，专门朝“能骗过 D”的方向改进。这盘博弈写成目标函数只有一行：

$$\min_G \max_D V(D,G) = \mathbb{E}_{x\sim p_{\text{data}}(x)}\big[\log D(x)\big] + \mathbb{E}_{z\sim p_z(z)}\big[\log\big(1-D(G(z))\big)\big] \tag{12.1}$$

**人话翻译**：别被符号吓住，两边各说各的话。后半句里，$D$ 想让 max 成立——真图 $x$ 判成 1（$\log D(x)$ 越大越好），假图 $G(z)$ 判成 0（$1-D(G(z))$ 越大越好）；前半句里，$G$ 想让 min 成立——它生产的假图要是能骗得 D 打出高分 1，$1-D(G(z))$ 就趋近 0，D 一无所获。一句话：**D 想识破 G，G 想骗过 D，双方在同一个函数里较劲**。

这场博弈的终点在哪？博弈论早有答案（见下面的盒子）：理想的平衡点是生成分布 $p_G$ 与真实分布 $p_{\text{data}}$ 完全重合——假样本以假乱真，D 再也无从下手，只能对一切图都输出 $\frac{1}{2}$，闭着眼抛硬币。这时候造假这门手艺就“毕业”了。

**博弈论一分钟：纳什均衡**

石头剪刀布的最优策略是什么？以 $\frac{1}{3}$ 的概率随机出拳。一旦你偏科（比如爱出石头），对手就能针对性克制你。这个“谁单方面改变策略都占不到便宜”的状态，就是**纳什均衡（Nash equilibrium）**——博弈论经济学家约翰·纳什提出的概念，他的故事被拍成电影《美丽心灵》。GAN 的训练就是在两个网络的策略空间里找纳什均衡。麻烦在于：第 3 讲训练分类网络是“一个人下山”，沿梯度往下走就有唯一谷底；GAN 是“两个人拔河”，你往东他往西，**没有现成的下山路线**——这正是下一节训练拉锯战的根源。

## 12.3　训练：一场没有裁判的拉锯战

GAN的训练循环是交替进行的：每批数据先更新 D 几步，再更新 G 一步，如此往复。核心代码骨架值得背下来：

```
for epoch in range(50):
    for real_imgs, _ in loader:
        # ---------- 第一步：更新判别器 D ----------
        z = torch.randn(len(real_imgs), 100)     # 采一批噪声
        fake = G(z)                              # 生成器造假
        d_loss = loss(D(real_imgs), ones) \
               + loss(D(fake.detach()), zeros)  # detach: 这笔账不记到G头上
        opt_D.zero_grad(); d_loss.backward(); opt_D.step()

        # ---------- 第二步：更新生成器 G ----------
        g_loss = loss(D(G(z)), ones)             # G的唯一目标：骗过D
        opt_G.zero_grad(); g_loss.backward(); opt_G.step()
```

三处细节：`detach()` 把假图从 G 的计算图里摘出来，让“批评 D 判错假图”这笔账不会顺着梯度算到 G 头上；更新 G 时用的是 `loss(D(G(z)), ones)`——G 唯一的目标函数就是“让 D 把我的假货判成真货”；两边各用各的优化器（`opt_D`、`opt_G`），互不干扰。

那么 GAN 的 loss 曲线长什么样？见下图：两条曲线互相咬合、上下拉锯，不像第 3 讲那条平滑下降的坦途。**这不是 bug，是常态**——D 的 loss 降了说明鉴定变严，G 的 loss 随之升了说明造假受阻、于是奋起直追，反过来又把 D 的 loss 顶上去。GAN 的 loss 只是“战况晴雨表”，你看不出“收敛”没有，只能看出“谁暂时占上风”。所以 GAN 训练好不好，历史上主要靠人眼看生成样本。

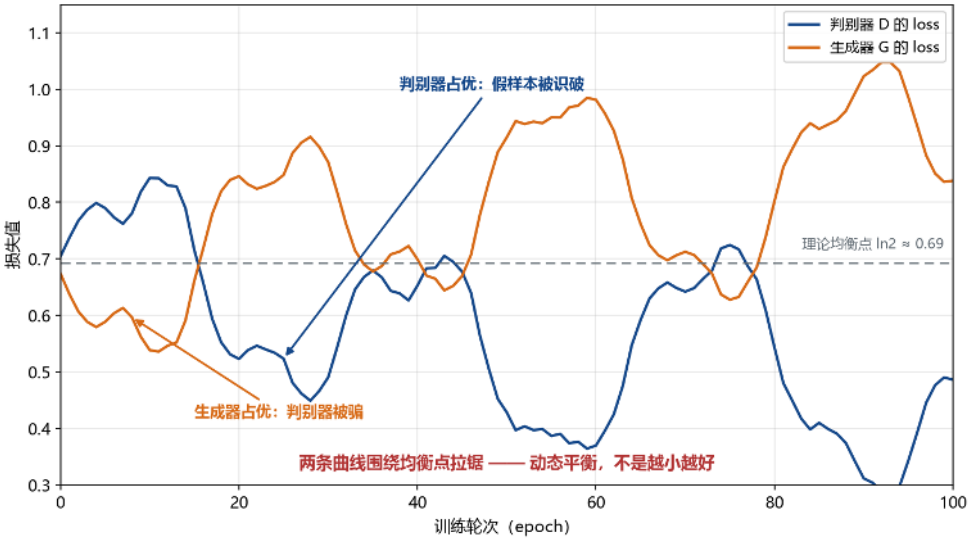



**图 12.3**　图题：GAN 训练拉锯：生成器与判别器 loss 交替波动，谁也压不死谁

拉锯战里最著名的翻车现场叫**模式崩塌（mode collapse）**：G 某天发现某一种猫总能骗过 D，于是它干脆不再冒险，把所有噪声都映射成这一种猫——**多样性死亡**，你想要一千种猫，它只会画一种。工程上还有一整套止血技巧，这里名词级过三句：一是**交替更新**上多让 D 走一两步，别让鉴定师先垮；二是**标签平滑（label smoothing）**，把真标签 1 改成 0.9，别让 D 变得过度自信、一句话判死；三是**谱归一化（spectral normalization）**，给 D 的权重矩阵加“限速带”，防止它一朝练成就能一刀封喉。后来的 WGAN 则干脆换了把更顺滑的“尺子”来度量两个分布的距离，让 loss 终于有了参考价值。记住：调 GAN 曾经是深度学习里公认的手艺活——这也是它后来被扩散模型取代的伏笔。

## 12.4　GAN 演进长廊：从模糊噪点到以假乱真

2014到 2018 年，GAN 家族五年间跑完了一场进化冲刺，几个里程碑如下表。

**表 12.2**　GAN 家族演进速览

| 年份 | 名字 | 一句话贡献 |
|---|---|---|
| 2014 | GAN | 开山之作：博弈式生成，结构还是全连接 |
| 2014 | 条件 GAN（cGAN） | 给 G 加“点菜”条件：要哪个类、要什么文字 |
| 2015 | DCGAN | 卷积版 GAN：训练首次稳定，成为标配基线 |
| 2017 | CycleGAN | 无配对数据域迁移：马变斑马、照片变油画 |
| 2018–2020 | StyleGAN 系列 | 风格分层注入：高分辨率人脸可控生成 |

**DCGAN** 把 G 和 D 全部换成了卷积结构——G 用转置卷积（第 5 讲卷积的“反操作”，把小图逐步上采样放大）从噪声“画”出图像，D 用普通卷积做鉴定，再配上批归一化等稳定训练的手段，GAN 第一次稳定地在图像任务上开花。**条件 GAN** 在输入里拼接一个条件（类别标签、文字描述），生成从“抽盲盒”变成“点菜”。**CycleGAN** 最神奇：不需要“同一匹马的同角度照片”，仅靠两堆毫无配对关系的图片（一堆马、一堆斑马），靠“过去再回来必须还是你”的循环一致性损失，学会了两个世界之间的翻译——马变斑马、夏天变冬天、照片变莫奈。**StyleGAN** 则把生成过程分层控制：先定大格局（脸型），再定中层次（五官），最后补细节（雀斑、发丝），一路生成分辨率与逼真度都惊人的人脸。



**图 12.4**　图题：GAN 生成人脸的五年进化：从 2014 年的模糊噪点到 2018 年后的以假乱真

图片来源：来源：NVIDIA StyleGAN 2 官方样张（NVlabs/stylegan2）

上图是这场进化的直观成绩单：2014 年论文里的手写数字模糊得像隔着毛玻璃，2018 年 StyleGAN 的人脸已经能在视觉上骗过人类——请注意其中几张是真照片、几张是生成的，你未必分得出来。顺便回答一个常见疑问：StyleGAN 生成的人脸之所以一张一个样，正因为 G 学到的是整张“人脸分布地图”而不是某几张脸——这是对模式崩塌的反向印证。

## 12.5　另一条路：变分自编码器（VAE）

GAN 在五年进化冲刺的同时，另一条更“文静”的生成路线也在悄悄长大——**变分自编码器**（Variational Autoencoder，VAE，2013 年提出，比 GAN 还早一年）。回忆第 4 讲的自编码器：编码器把图片压成一个隐向量，解码器再从隐向量还原。它离“生成”只差一步，却是最致命的一步：那个隐空间是“碎”的——只在训练见过的那几个点附近能重建，随手戳一个新点，解码器吐出来的多半是乱码。

VAE 的药方一句话：**把“一个点”换成“一团云”**。编码器不再把一张图压成一个坐标，而是压成两个向量——**均值**与**标准差**，等于给这张图画了一份“模糊画像”：它大概住在哪个片区、活动半径多大；然后从这团云里**随机采一个点**，交给解码器还原。“模糊画像 + 随机采样”两个设定，逼得隐空间处处连续、片片相接——随便戳一个点，解码器都见过长得差不多的邻居，都还原得出一张合理的图。福利随之而来：在两张图的云之间连一条线、沿线戳点，解码器会给你一串“从猫渐变到狗”的过渡帧——隐空间可以做插值了。

训练 VAE 的损失函数是两部分相加。一部分是**重建误差**：还原的图要像原图（第 3 讲那些回归损失的老朋友）；另一部分是 **KL 距离**：把每张图的云往**标准正态分布**上拉，不许某张图的云飘到天南海北、跟别人老死不相往来。一句人话：**还原得要像，云还得守纪律**。代价也藏在纪律里：为了让隐空间处处“合理”，VAE 生成的图天然带一层“平均脸”式的糊——它总在给“什么位置该有什么”打安全牌，锐度让位给了稳健。至于名字里的“变分”，来自统计学的变分推断，你只需记住它管的事：用“均值加标准差的云”去近似真正的数据分布——名词级了解即可。

把两条路线并排一摆，性格差异一目了然：

**表 12.3**　VAE vs GAN：两种生成哲学

| | VAE | GAN |
|---|---|---|
| 生成质量 | 偏糊：“平均脸”式的安全牌，锐度不足 | 犀利：细节逼真，但可能模式崩塌 |
| 训练 | 稳：损失函数一条、普通反向传播，loss 曲线老实下降 | 难伺候：双人拔河、loss 拉锯、要一整套止血技巧 |
| 本质 | 学分布的**参数**：把每张图编成“均值 + 标准差”，再约束它向标准正态看齐 | 学一场**博弈**：在真假对抗中把生成分布逼向真实分布 |
| 2026 年的余脉 | 潜空间基建活在扩散模型里（下一节） | 判别器思想活在鉴伪检测器里（本章末节） |

GAN 与 VAE 之外，生成家族其实还有第三条路，和第 11 讲的接龙一脉相承——**自回归模型**（autoregressive model）：像写字一样一个一个往外“蹦”。PixelRNN、PixelCNN 把图像按行列排好顺序，从左上角第一个像素起，逐像素预测下一个像素；GPT 式的图像模型则把图切成小块当 token，逐 token 往外接。文字接龙接的是字，图像接龙接的是像素块——任务设定完全同构，变的只是接龙的对象。

临收尾再记一笔，供你在 2026 年认亲：Stable Diffusion 全名里的 *Latent*（潜空间），指的正是先由一套 VAE 把图片压进低维潜空间、再在潜空间里做扩散去噪——**2026 年扩散模型的潜空间技术，直系祖先就是 VAE**。这条路线没有死，它转岗当了扩散时代的基建。

## 12.6　后浪：扩散模型

GAN统治图像生成五六年后，2020 年一种老思路新包装的模型悄悄崛起，2022 年全面接管，它就是**扩散模型（diffusion model）**。它的核心想法朴素得惊人，分两步：

1. **前向过程（加噪）**：拿一张真实图片，一小步一小步地往上撒高斯噪声，撒 1000 步，直到原图彻底淹没成一片纯噪声。
2. **反向过程（去噪）**：训练一个网络（通常是 U-Net），给它看“加噪到第 $t$ 步的图”，让它预测这一步撒进去的噪声是什么——预测对了就把噪声减掉。训练好之后，从一张纯噪声出发，网络一步步把噪声“雕刻”掉，图像便从混沌中浮现。

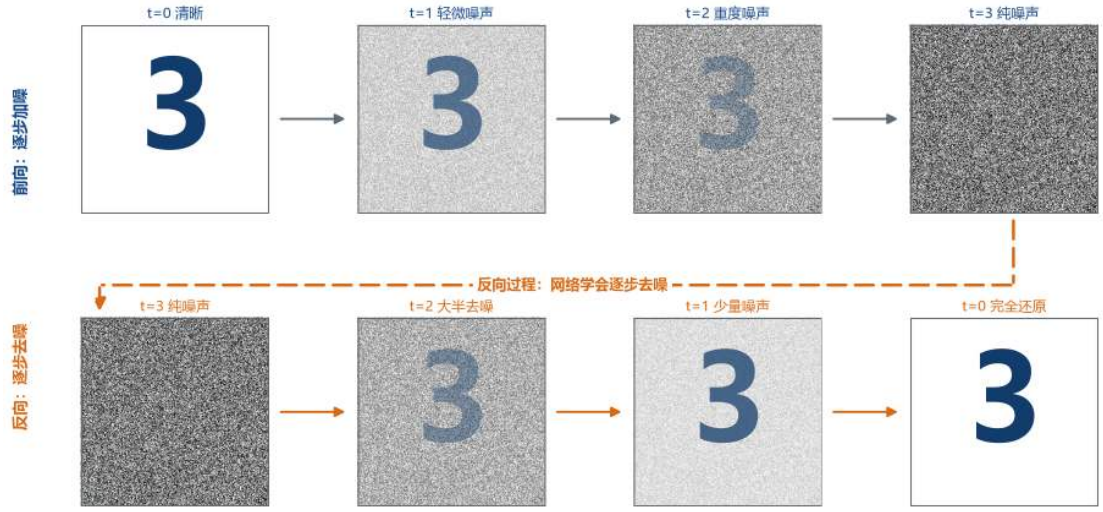



**图 12.5**　图题：扩散模型：前向过程逐步加噪直至纯噪声，反向过程学会一步步去噪还原

训练目标更是朴素到令人发笑：预测的噪声和真实撒的噪声算一个均方误差——就是第 3 讲最古老的回归损失。没有博弈、没有拔河，loss 老老实实往下降。下图把整个流程摊开给你看。

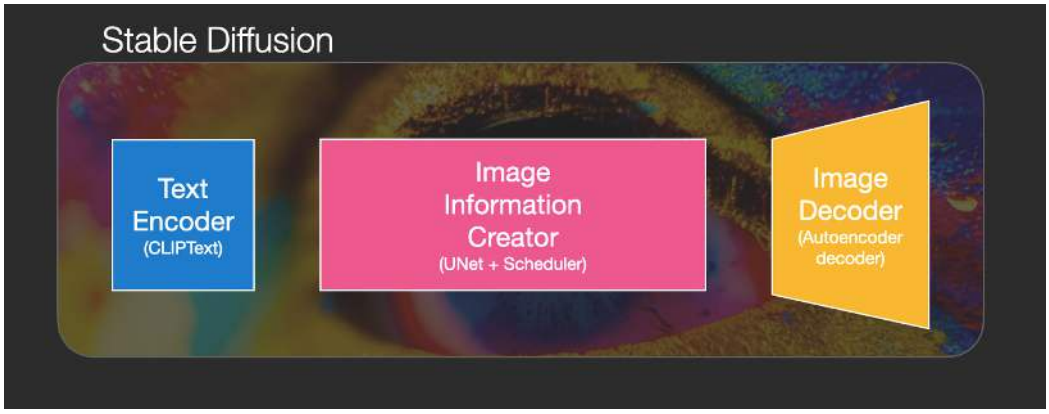



**图 12.6**　图题：Stable Diffusion 流程：文本编码器理解提示词，U-Net 在隐空间里迭代去噪，解码器还原成图

图片来源：来源：Jay Alammar, The Illustrated Stable Diffusion 博客文章, 2022

2022 年，Stable Diffusion 和 Midjourney 相继出圈，“一句话生成一张画”从此进入大众生活；2024–2026 年，同样的技术沿着时间轴扩展，文生视频（Sora、可灵等）成了新战场。扩散为什么能取代 GAN 的主流地位？两个原因：**训练稳**——没有对抗博弈，loss 曲线终于能看懂，调参不再靠玄学；**模式覆盖全**——以拟合数据分布为目标，天然不易模式崩塌，多样性好。代价也明摆着：**采样慢**——GAN 一次前向就出图，扩散要跑几十步去噪迭代。近两年蒸馏与一致性模型正把这几十步压到几步甚至一步（名词级了解），速度差距在快速弥合。还有个彩蛋：翻看 Stable Diffusion 的结构图你会发现，它的去噪网络里塞满了自注意力层，文本理解靠的是 CLIP——**第 11 讲的注意力，在这里一个都不少**。全书到这里合流了：识别靠它，生成也靠它。

## 12.7　深度伪造与鉴伪：攻防共生（工程伦理专场）

技术从实验室走进社会，一定是矛与盾一起长大的。**深度伪造（deepfake）**是生成模型最刺眼的社会副作用：换脸视频把明星的脸安到别人身上，声音克隆用几秒钟素材就能模仿一个人，电信诈骗已经出现“冒充熟人视频通话”的真实案例。与之对抗的**鉴伪检测器**，本质上就是本讲的判别器再就

业：在真图与 AI 图之间画边界，专找生成内容留下的指纹——发丝边缘的涂抹感、眨眼节奏的不自然、眼镜框在镜片两侧不对称、背景文字的扭曲。



**图 12.7**　图题：深度伪造真伪对比：左侧换脸痕迹与右侧真实人脸的细微差异是检测器的切入点

图片来源：来源：网络公开的深度伪造真伪对比样例（FaceForensics++ 等检测研究常用）

更深一层的攻防是**对抗样本（adversarial example）**：在图片上叠加一层人眼几乎看不见的扰动，就能让训练好的识别模型把熊猫认成长臂猿——攻击者用“生成一小段扰动”的思想制造模型盲区；防御方则把这些对抗样本加进训练集重新训练，让模型提前“见过世面”，这叫**对抗训练（adversarial training）**，相当于给模型打疫苗。造假与鉴伪、攻击与加固，是同一枚硬币的两面，谁也消灭不了谁，只能螺旋上升——理解了这一点，你就理解了本讲标题里“对抗”二字的分量。

**工程伦理：造假的能力越大，克制的分量越重**

三条红线，请当作职业操守记住：第一，**不制作、不传播**深度伪造内容去冒充他人、伪造证据，这在我国已明确触碰法律红线——2025 年起施行的 AI 生成合成内容标识办法要求 AI 生成内容必须“亮明身份”；第二，做工程时**不许用生成数据冒充实测数据**交差，实验一至实验四的所有证据都必须是“命令 + 日志 + 产物”三件套，生成的数字再漂亮也是假的；第三，**对“眼见为实”保持怀疑**是新一代工程师的基本素养——你看到的视频、听到的声音，都可能是某台服务器上一次前向传播的输出。能力越强，越要自觉。

**当一回“鉴定师”：给 AI 图找破绽**

打开任一文生图工具生成一张“街景人物”图片（或从网上找一张疑似 AI 图），和同学互相挑破绽，比谁先找到三处。经典破绽清单：手指数量与关节；牙齿的排列；眼镜框在镜片两侧是否连贯；招牌和路牌上的文字是否成字；耳朵左右是否对称；背景里栏杆、砖缝这类直线是否突然扭曲。玩完想一想：你刚才做的事，就是把“人肉判别器”的注意力分配到纹理与几何的异常上——鉴伪检测器做的同一件事，只不过它的“直觉”是几百万个卷积核。你也可以反过来玩：把找到的破绽写成提示词让 AI 重画，看它能否骗过你——恭喜，你刚体验了一轮极小极大博弈。

**前沿速递：2024–2026 生成式 AI 全景**

- **进入工业流水线**：文生图（Midjourney、Stable Diffusion 3、Flux）用于广告与概念设计，文生视频（Sora、可灵、即梦）用于影视预演与短视频量产，文生 3D 开始给游戏和数字孪生喂资产——“生成”从玩具变成了生产力工具。
- **世界模型（world model）**：让模型从海量视频里学“物理规律”，预测世界的下一帧——物体该往哪掉、水花怎么溅。它被视为机器人与自动驾驶的“训练 simulator”，NVIDIA Cosmos 等已开源（名词级了解）。
- **鉴伪与水印在追赶**：C2PA内容凭证标准给图像盖上“出生证明”式的元数据链，Google 的 SynthID 把不可见水印嵌进生成的像素，平台强制标识 AI 内容——生成能力越强，溯源技术越是刚需，而两者至今都在“追赶者”的位置。

一句话总结这个时代：**生成的成本趋近于零，信任成为最贵的资源**。

**项目连线：对抗训练每周都在你的项目里跑**

- **数据基座（三）**：项目的“虚假内容识别与对抗训练”模块，就是 GAN 思想在数据治理上的变体：大模型（DeepSeek）当“造假者”，按它学到的真实数据模式批量生成 26 个数据家族的假数据；规则引擎加小模型当“监考”，逐条判定真假；LLM 当“判官”复核争议样本；被判定“以假乱真”的样本回流当训练集并加 3 倍权重——造假者下周就学会绕开旧监考，监考每周升级。这场军备竞赛真的在你们的实验室里每周运转，demo 界面与曲线见第二篇项目章。
- **六目拼接（一）与水下双目（二）**：坏帧（黑帧、拖影、过曝）和浑浊的水下雾，可视为真实世界的“生成式污染”——把这类样本主动加进训练集做鲁棒性训练，与对抗训练是同一个思想：先见假，后识真。
- **串起三个项目**：数据基座里的“监考/判官”是判别器思想，数据基座里的问数智能体是第 11 讲的 agent，水下项目的 FoundationStereo 是第 11 讲的注意力骨干——你会发现全书没有一章是孤立的。

## 12.8　从这一讲到 2026

生成式模型在 2024–2026 年完成了一次“换代”，三条新闻值得你记住。**第一，训练范式从扩散转向流匹配**：**流匹配**（Flow Matching，工程上常用其特例**整流流** Rectified Flow）把扩散那套“一千步逐步去噪”换成了“直接学一条从噪声到数据的直线路径”——训练更快、更稳，采样步数更少；它

已是 2025–2026 年图像与视频生成的主流训练范式，还扩展到了语音，另有 ProReflow 等工作把多步扩散蒸馏成几步甚至一步生成。**第二，视频生成进入群雄时代**：Sora 2 与可灵（快手 Kling 2.5/2.6）、Veo、Runway、Seedance 同台竞技，可灵 2.6 还开放了**运动控制 API**——把真人演员的动作直接迁移到生成角色身上，“文生视频”正从玩具走向生产线。**第三，“眼见为实”的防线在重建**：美国《Take It Down Act》（2025 年签署）把非自愿深度伪造**入刑**、并要求平台限时删除；欧盟 AI Act 强制 AI 生成内容明确标注；C2PA 内容凭证（给图像盖“加密签名的出生证明”）开始被监管采纳——但《华盛顿邮报》实测发现，8 个主流平台里有 7 个会在上传时**剥离**这些凭证。攻防远未收官，结论朴素：**溯源、检测、立法，三管齐下，缺一不可**。

所以，这一讲的博弈论教你看懂的是 2026 年最重要的一场工程攻防：造假者用生成模型武装到牙齿，而鉴定师、立法者与密码学家正用同一套“对抗”思想筑墙——“深度伪造与鉴伪”一节的鉴伪检测器、前沿盒子里的水印与凭证、上面这些法律条文，都是这场博弈的新玩家。你将来选择站哪一边，都得先懂这一讲。

**第 12 讲课后编程题（用 AI 实现）**

**题目**：用 AI 辅助训练一个 DCGAN，在 MNIST 上生成手写数字（50 个 epoch，CPU 半小时内可跑完，或用 Colab；实在没有条件，可改用 `diffusers` 库调用预训练扩散管线生成数字，交付要求相同）。重点观察**模式崩塌**：随着训练推进，生成的数字种类是越来越丰富，还是越来越窄（比如只剩几个 1 和 7）？

**交付物**：代码文件 + 一张 $5 \times 10$ 生成样本网格图 + 生成器/判别器双 loss 曲线 + 50 字以上观察记录（有无模式崩塌迹象，依据是什么）。

**提示词建议**：

```
请帮我用 PyTorch 写一个最小可跑的 DCGAN，在 MNIST 上生成手写数字：
1. 生成器：100 维噪声经转置卷积上采样到 28x28，末层用 tanh；
2. 判别器：普通卷积下采样，末层用 sigmoid，图像先归一化到 [-1,1]；
3. 训练 50 个 epoch，Adam 学习率 0.0002，每个 batch 先更新 D 再更新 G；
4. 每 10 个 epoch 保存一张 5x10 生成样本网格图，同时记录两个 loss；
5. 结束后用 matplotlib 画双 loss 曲线。
我只有 CPU，请把网络宽度和批量大小控制在 30 分钟内能跑完的规模，
中文注释解释每一层的作用和交替更新的用意。
```

**验收线**：代码能跑完并出图；你能向同学讲清楚这段代码里“谁在学谁”——D在学分辨真假、G在学 D 的鉴定标准；至少一处你自己的修改（噪声维度、加深一层、换学习率）并说出看到的变化。“代码是 AI 写的没关系，讲不清楚就是没做。”

**加分项**：把无条件 GAN 改成条件 GAN，指定“只生成数字 7”；或对比训练 50 与 200 个 epoch 的样本多样性，写 100 字分析过训练与模式崩塌的关系。

**判别模型（Discriminative Model）**

学习类别边界 $p(y \mid x)$ 的模型，看图说话；本课前十一讲的主角。

**生成模型（Generative Model）**

学习数据分布 $p(x)$ 并能从中采样出新样本的模型，无中生有。

**生成器（Generator）**

GAN 中的造假者：把噪声向量映射成假样本，目标是骗过判别器。

**判别器（Discriminator）**

GAN 中的鉴定师：给输入打“真/假”分数，深度伪造检测器的直系祖先。

**极小极大博弈（Minimax Game）**

G 的最小化目标嵌在 D 的最大化目标里，双方在同一函数中较劲的数学表述。

**纳什均衡（Nash Equilibrium）**

任何一方单方面改变策略都占不到便宜的稳定状态；GAN 训练寻找的理想终点。

**模式崩塌（Mode Collapse）**

生成器只产出少数几种“安全”样本、多样性死亡的著名训练事故。

**条件 GAN（cGAN）**

给生成器附加条件（类别、文字）的变体，生成从抽盲盒变成点菜。

**CycleGAN**

无配对数据之间做风格迁移的 GAN，靠循环一致性损失实现马变斑马式“翻译”。

**StyleGAN**

风格分层注入、可高分辨率可控生成人脸的 GAN 系列，2018–2020 年以假乱真的代表作。

**扩散模型（Diffusion Model）**

前向千步加噪、反向学会逐步去噪的生成模型，2022 年起取代 GAN 成为图像生成主流。

**对抗训练（Adversarial Training）**

把对抗样本纳入训练让模型提前“见世面”的加固手段，本讲与数据基座项目的共同关键词。

# 第二部分

# 第二篇　项目背景篇：三个真实项目到底在做什么

# 第二篇导读：三个项目全景

**第二篇地图**

**讲什么**：三个真实项目的来龙去脉——六目拼接、水下双目自动标记、数据基座治理，外加全班 15 个小组的分工地图与人人要守的课程框架。**为什么值得读**：第三篇的任务书只告诉你“做什么、怎么交”；这一篇告诉你“为什么做、船开向哪”。背景清楚了，任务书上的每条要求才不是黑话。**学完你能**：用一句话说清自己项目的甲方痛点与任务，找到自己组在 15 组地图上的位置，把四个实验节点和最终提交日标进自己的日历。

先设想一个场景。开学第三周，组长把任务书拍到你面前：“组 5，双台账加 A/B 对比，回归脚本每周跑一次，第 19 周周五交。”如果你不知道台账为谁而建、A/B 在比什么、回归到底回归什么，这份任务书就是一堵墙；如果你读过对应的项目章，你会发现每一条要求背后都有一个具体的人和具体的难处——台账是给评测员看的，A/B 是在拿新模型和老算法比武，回归是在防止“改好一处、弄坏三处”。**这一篇的任务，就是把那堵墙提前拆掉**：拿到任务书之前，先知道船开向哪。

## 0.9 三个项目：先来一张速览表

这门课的作业不是教科书后面的习题，而是三个**正在运行的真实工程项目**。全班 15 个小组（每组 5 人）分属其下。三个项目的甲方背景、一句话任务、组数与课程支撑讲次见下表，随后每个项目再给一段人话介绍；完整的现场背景、系统架构与难点剖析，请进入第二篇各章细读。

**表 4** 三个真实项目速览（15 个小组，每组 5 人）

| 项目 | 甲方 / 背景 | 一句话任务 | 组数 | 课程支撑 |
|---|---|---|---|---|
| 六目拼接 | 360Stitcher 在研实时全景系统 | 把实时 360° 全景拼接系统里九类人工调参，升级为九个可学习的神经网络“参数头” | 8 | 第 0–3、5–11 讲 |
| 水下双目 | 中海油 HYSY123 FPSO | 给 ROV 锚链检查视频装上双目立体感知：教师打伪标签，训练轻量学生模型 | 4 | 第 0、5–8 讲 |
| 数据基座 | 海洋石油平台 | 治理 126 份 CSV 共 3.1 亿行数据与 1001 页手册：结构化、一句话问数、虚假内容识别与对抗训练 | 3 | 第 3、9–12 讲 |

**项目一：六目拼接。**六台工业相机围成一圈，各管约 60°，系统把六路视频实时拼成一幅 360° 全景。过去拼得好不好，全凭九类手工调的参数；这学期要把它们升级成九个可学习的神经网络“参数头”，而且要配上回退保护——模型万一失手，系统自动退回老算法，现场一分一秒不能停。详见第 一 章。

**项目二：水下双目自动标记。**海上十几万吨的 FPSO 靠几条巨型系泊锚链“锚”在油田上方，锚链要靠水下机器人（ROV）定期下水检查，拍回的视频浩如烟海。项目用双目相机的左右两幅画面算出深度，让机器自动认出锚链每一环；再用大模型当“教师”打伪标签，过滤后训练轻量的“学生”模型，500 对锚链立体图对的数据家底由你们亲手搭建。详见第 二 章。

**项目三：数据基座治理。**某海洋石油平台的机房里躺着 126 份 CSV 文件、共 3.1 亿行运行数据，外加 1001 页工程手册，靠人翻一辈子翻不完。项目干三件事：把数据与手册**结构化**；搭“一句话问数”智能体——工程师说人话，大模型加图检索负责听懂并查数；训练“监考员”模型识别虚假内容，并和造假方反复过招、共同变强。详见第 三 章。

## 0.10 共同的课程框架：节点、证据与底线

三个项目背景各不相同，但游戏规则完全一致，三件事人人要记。

**第一，四个实验节点加一条提交线。**四次现场实验与项目节点并行推进，时间如下表。注意：课程层面四次实验有统一主题，但在你自己的项目里，每个节点各有具体动作——以六目拼接为例，实验一是用大模型 agent 解读 360Stitcher 代码，实验二是在本机完成 VS2022 编译部署并看到九行 [Provider] 日志与探针 12/12 通过，实验三是真训练加交付加 A/B 对比，实验四是复训与优化。各组动作的命令级细节，见第三篇各组任务书的“四次实验安排”一节。

**表 5** 四个实验节点与最终提交（所有项目通用）

| 节点 | 时间 | 课程层面做什么 |
|---|---|---|
| 实验一 | 10月 12 日 | 大模型 agent 模式配置及应用：装好编程助手，并用它解读自己项目的代码 |
| 实验二 | 11 月 2 日 | 匹配合适程序并执行本地部署：项目程序在自己电脑上编译跑通 |
| 实验三 | 11 月 23 日 | 深度网络的模型训练实践：亲手完成所属项目的训练任务 |
| 实验四 | 12 月 21 日 | 模型优化训练实践：把模型跑得又快又稳 |
| 提交线 | 第 19 周周五（2027 年 1 月 22 日） | 全部成果提交验收，完全提交离场 |

**第二，证据包三件套。**凡声称“我做出来了”，必须同时拿出**命令、日志、产物**三件套：你是敲哪条命令跑出来的、终端输出了什么、留下了什么文件。截图可以修图，口说无凭，只有三件套齐了，成绩才认。

**第三，底线只有一句话：AI 可以用，结果自己扛。**这门课不禁止 AI，反而要求你用 AI——但每个交付物都可能被随机抽问，讲不清楚就是没做；实验数字必须是自己代码在自己数据上跑出来的。具体到项目里，还有五条防糊弄纪律：**证据包制、探针防绕过、A/B 数据说话、双台账、如实条款**（不得虚构训练精度，发现一处该节点成绩作废）。它们在项目章和任务书里会反复出现，这里先混个脸熟。

## 0.11 15 个小组地图：找到你自己的位置

全班 15 个组的分工见下表。六目拼接的 8 个组名与任务即第三篇第 1–8 组的章名；水下四组与数据基座三组同理，各组任务的精确边界以第三篇任务书为准。

表 6　15 个小组地图（组名与一句话任务）

| 组号 | 项目 | 组名 | 一句话任务 |
|---|---|---|---|
| 第 1 组 | 六目 | 匹配决策组 | 训练并交付 policy、mc、geometry 三个头：这一对画面怎么拼、信几分、几何参数是多少 |
| 第 2 组 | 六目 | 特征质量组 | 训练并交付 feature_uncertainty、descriptor_softening、temporal_state 三个头 |
| 第 3 组 | 六目 | 缝合融合组 | 训练并交付 seam_uncertainty、gain_field、fusion 三个头 |
| 第 4 组 | 六目 | 数据与标签组 | 录六路视频、抽帧、对齐遥测，制作 match/geometry/policy 三段标签 |
| 第 5 组 | 六目 | 测试集与评测组 | 建指标台账与数据量台账，A/B 对比 `ab_compare.py`，回归 `run_regression.ps1` |
| 第 6 组 | 六目 | 端到端数据组 | 建 50 对以上真实图对数据集与 benchmark |
| 第 7 组 | 六目 | 端到端训练组 | 在 RTX 4080 上以 `--epochs 200` 训练 `ropstitch.onnx`，探针 5/5 |
| 第 8 组 | 六目 | 端到端优化组 | 引擎单次调用 ≤300 ms、总预算 800 ms；fp32/fp16/int8 三档对比，int8 劣化 <10% 才交付 |
| 第 9 组 | 水下 | 数据组 | 采集并构建锚链双目数据家底（含 500 对立体图对） |
| 第 10 组 | 水下 | 部署组 | 把训练好的学生模型部署到现场实时可用，并给出部署证据 |
| 第 11 组 | 水下 | 训练组 | 教师–学生伪标签闭环训练 |
| 第 12 组 | 水下 | 评测组 | 立体感知指标的评测与验收 |
| 第 13 组 | 基座 | 数据结构化组 | 把 126 份 CSV 共 3.1 亿行数据与 1001 页手册结构化入库 |
| 第 14 组 | 基座 | 智能问答与图检索组 | 搭“一句话问数”智能体：大模型 + 数据库 + R-GCN 图检索 |
| 第 15 组 | 基座 | 虚假内容识别与对抗训练组 | 训练“监考员”模型识别虚假内容，与造假方对抗过招 |

看这张表时留意一件事：**这门课为每个组都准备了“不等待”的设计**。实验一至三是离线自研：水下四个组开局就领到资产包（FoundationStereo环境、500 对锚链图对、初步模型），六目八个组用公开数据集（UDIS-D、MegaDepth、HPatches 等）与系统自带仿真模式先把方案跑通；实验四（12 月 21 日）才是合并联调——真实数据接入、跨组对账、统一验收。所以：跨组沟通节点要写进周计划，但任何一组都不需要等另一组开工。

## 0.12　怎么读这一篇

建议的读法分三步。**第一步**，先精读自己所属项目的章节（六目拼接读第 一 章，水下双目读第 二 章，数据基座读第 三 章），目标是能向室友复述“甲方是谁、痛点是什么、我们组要交什么”；**第二步**，翻开第三篇，先读使用说明（第 三 章），再精读自己组的任务书，把提交清单和四次实验抄进日历；**第三步**，把另外两个项目的章当“课外读物”略读——知道隔壁组在干什么，联调时才不会鸡同鸭讲。

最后提醒一句：第二篇不是读完就束之高阁的小说。第三篇任务书里大量术语——参数头、伪标签、对抗训练、双台账——的完整解释都在第二篇对应章节里，读任务书卡壳时，回到第二篇查，比问 AI 更可靠，因为这里的每个字都对着你们项目的真实系统。

# 项目一　六目拼接：给六只眼睛一个全景世界

想象你站在一座海洋石油平台的直升机坪中央。海风、管线、火炬塔、远处的补给船——四面八方都是需要盯住的东西。可无论把一台相机架在哪里，它一次也只能看住一个方向：普通监控相机的水平视野不过 90° 上下，人眼双眼合并的视野也只有约 120°，而平台四周是完整的 360°。装四台相机各看一面？值班员就得在四块屏幕间来回切换，脑子当拼图器。**更好的办法是让机器自己把画面拼成一幅完整的全景**——这正是全景拼接（panorama stitching）技术要做的事，也是你们项目一的全部出发点。

**本章地图**

**讲什么**：六目拼接项目的前世今生——需求从哪来、全景拼接五步曲的原理直觉、360Stitcher系统的真实架构与性能数字、为什么要给九类人工参数装上神经网络“参数头”、两条技术路线怎么对照推进、八个组如何分工。**为什么值得读**：第三篇任务书里的每个词（缝合线、增益、A/B、探针、回退）都长在这一章的枝干上。**学完你能**：向任何人讲清“六路画面如何变成一幅全景”“参数头是什么、为什么要学、学坏了怎么办”，并找到自己组的上下游位置。

下图是一幅典型的全景拼接成果：镜头原地转一圈，画面被摊平成一张长卷。项目一要做的，就是让这个过程**由六台相机实时自动完成**——不是拍完回家慢慢拼，而是每秒几十遍地拼下去。



**图 1.1**　全景拼接成果示例：相机原地旋转一圈，视野被展开成一幅无缝长卷——这就是“全景拼接”四个字的样子

图片来源：维基共享资源（Wikimedia Commons）360° 全景示例图，仅用于课堂教学

## 1.1　需求从哪来：一台相机看不住一座平台

先说甲方视角的痛点。油气平台、炼化厂区、大坝、广场周界这类场所，安全监控有个共同的尴尬：**要盯的地方是一整圈，而相机只看得见一个扇面**。多装几台相机当然能补，但画面各管一段，值班员要靠人脑把它们“缝”起来——疲劳的时候，缝就漏了。有人会想到超广角或鱼眼相机，但它在后文 1.2 节会看到死穴：边缘畸变严重，且有限的像素被摊到 360° 上，远处目标小到没法辨认。下面这座海洋石油平台就是典型场景：管线密布、人员往来，四周海面与航道都值得关注，任何一块视野盲区都是隐患。



**图 1.2**　应用场景示例：海洋石油平台——四周海面、航道与设施构成完整的 360° 监视需求，单台相机无论如何架设都看不全

图片来源：维基共享资源（Wikimedia Commons）海洋石油平台照片，仅用于课堂教学

项目的硬件答案是**六相机环形阵列**：六只“眼睛”头朝外围成一圈，每只负责约 60°，相邻两只的画面留出一条重叠带——正是这条重叠带，让后续算法能把六幅画面严丝合缝地接起来。市面上的消费级全景相机（下图这类多镜头阵列）用的也是同一思路：与其造一只万能的超级镜头，不如让一群普通镜头协同作战。

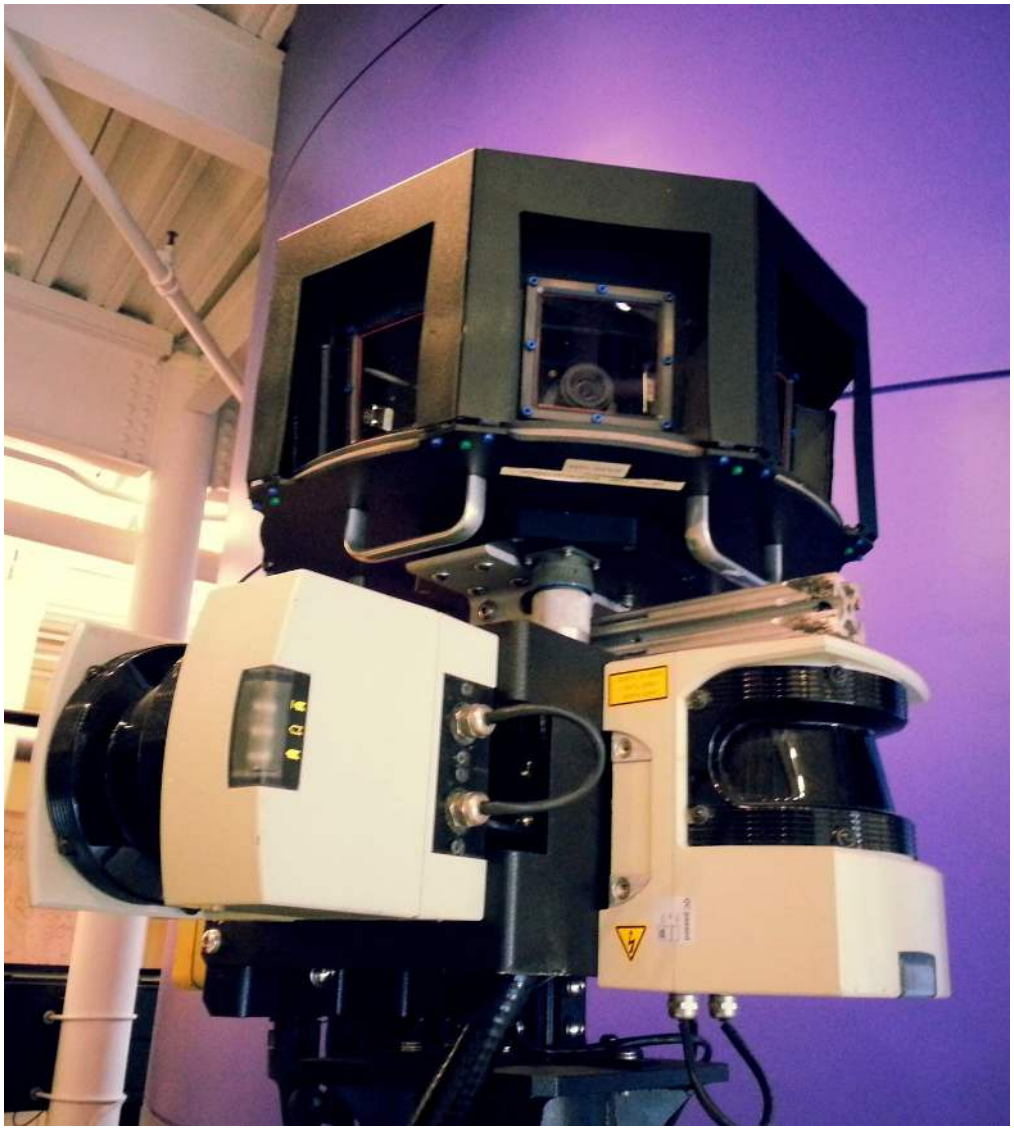


**图 1.3**　多相机阵列示例：若干镜头朝向不同方向、共享同一台处理单元——消费级全景相机与本项目六相机环阵是同一个思路

图片来源：维基共享资源（Wikimedia Commons）多相机全景阵列照片，仅用于课堂教学

我们的真实系统名叫 **360Stitcher**，是实验室在研的实时多相机全景拼接系统。看两张它的实际运行截图，你就能把“六目”两个字看明白：下图中，上排小窗标着“原始画面”，是六台相机各自看到的景象——一座红砖楼环绕、中央铺着草坪的园区庭院；下排那条黑底长卷，是系统把六路画面实时拼出来的全景。右下角的 FPS 读数 26.6，表示此刻系统每秒能拼出约 26 帧全景——注意是“拼出来”，不是“播放出来”，每帧都是六幅图现场算出来的结果。

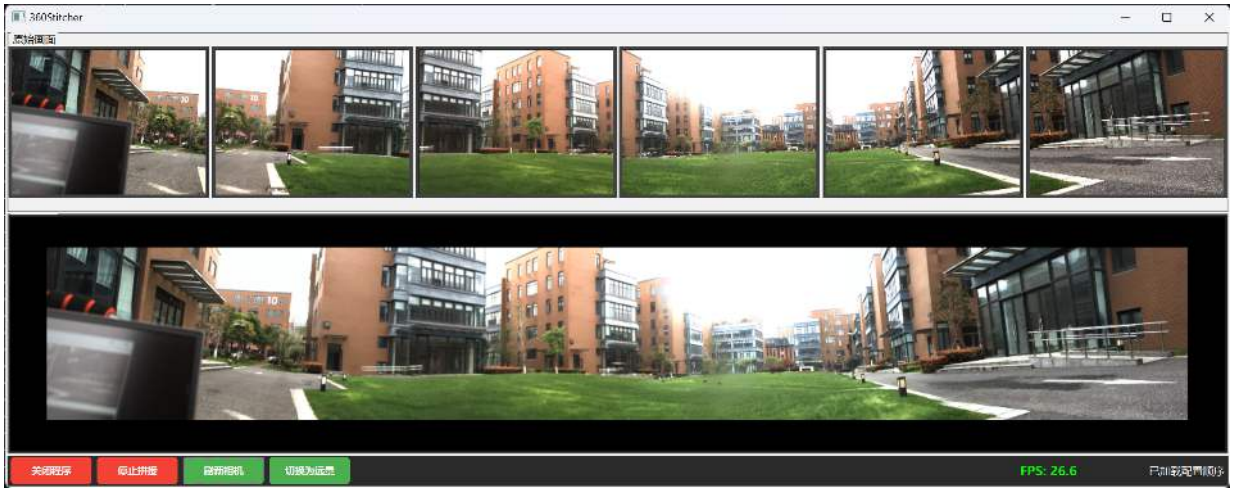



**图 1.4**　360Stitcher 实际运行截图（园区场景）：上排为六路相机的原始画面（红砖楼与草坪的庭院），下排为系统实时拼出的全景长卷；右下角 FPS 26.6 表示每秒约拼出 26 帧

图片来源：360Stitcher 项目组真实运行截图（六目拼接项目素材）

换一个场景它照样干活。下图的取景地是园区外侧的街道：斑马线、行道树、路边的共享单车与电动车，被六路画面分头拍下、无缝接成长卷。你仔细看长卷里那些横向连续的白线与绿化带——它们每个瞬间都在接受“六只眼睛看的是同一个世界”的检验，接歪一点都会立刻露馅。




**图 1.5**　360Stitcher 实际运行截图（街景场景）：斑马线、行道树与共享单车被六路画面接成一条连续长卷，横向的斑马线与绿化带是检验“接得正不正”的天然标尺；右下角 FPS 24.7

图片来源：360Stitcher 项目组真实运行截图（六目拼接项目素材）

最后回答“为什么不用一个超广角”。用数字算账最清楚：本项目相机单张原始图为 4608×3456 像素（约 1600 万像素）。按水平视野约 90° 估算，六目方案每度视野能摊到约 51 个像素；而假如用一只 360° 鱼眼把同样 4608 个像素摊满一整圈，每度只剩约 13 个像素——差了 4 倍，远处的人脸直接糊成马赛克。再看畸变：视野越广，镜头把直线掰弯得越厉害，画面边缘的人与物会被拉伸变形。六相机环阵用“普通镜头 + 算法拼接”换来了**高分辨率、低畸变的 360°**，代价是把难题从 optics（光学）转移给了 algorithm（算法）——也就是转交给了你们。

**零基础扫盲：人眼 120°，全景 360°，长卷是怎么“卷”出来的**

人眼双眼视野约 120°，是因为两只眼球朝前、各管约 60° 再拼起来——你会发现，**人眼本身就是一台“双目拼接”的机器**，本项目不过是让硅基眼睛重走了一遍进化路。那全景长卷为什么能摊平？设想把一圈相机看到的视野画在一只易拉罐的侧面，再把罐身剪开摊平——这就是**圆柱投影**（cylindrical projection）：把 360° 的环形视野展开成一张矩形长图。它的副作用你也见过：全景照片里，本来笔直的楼边在画面两端会略显弯——那是罐面被强行摊平留下的“折痕”。本章截图里的黑色长卷，正是这种圆柱投影长卷。

## 1.2　全景拼接五步曲：从六张歪图到一幅长卷

把六路画面变成一幅全景，系统每帧都要走完五步。先用下图建立全局印象，然后逐步拆解——这五步也是本章的骨架，后面 1.4 节的九个参数头，一个不落地长在这五步上。




**图 1.6**　全景拼接五步曲总览：畸变校正把“哈哈镜”掰直，特征匹配找到两幅画面的“同名点”，单应对齐把画面搬正，动态规划在重叠区挑出最不显眼的缝合线，融合把接缝羽化到看不见

### 1.2.1　第一步：畸变校正——先把“哈哈镜”掰直

理想相机是针孔模型：光线走直线，方块拍出来还是方块。现实中的镜头是玻璃磨出来的，光线在边缘弯得更狠，于是画面里的直线会向内弯——整幅图像得像照过哈哈镜，这叫**桶形畸变**（barrel distortion）。畸变不除，后面算“两幅图差多少”时全都会失准：你以为两幅画面对不齐，其实是镜头在撒谎。

对策是**畸变校正**（distortion correction）：离线给每台相机做一次**标定**（calibration）——拍十几张棋盘格照片，解出这台镜头的“度数”（焦距、光心等内参和畸变系数），此后运行时每帧图像先按这组系数反解、掰直，再进入后续流程。你已经在 1.3 节的截图里提前见过它的道具：室内场景那块贴在白板上的黑白棋盘格，就是标定用的靶子。

### 1.2.2　第二步：特征提取与匹配——在两幅画面里找到“同一个点”

校正之后，算法要回答一个关键问题：左眼画面里的这栋楼角，对应右眼画面里的哪个像素？逐像素两两比对？两幅 1440×1080 的图有十几亿种配对，算到下个世纪。工程的办法是只挑**特征点**（feature point / keypoint）说话：选那些“挪一点就大不一样”的位置——角点（corner）。想象拼拼图时你会先对天空？不，你会先拼窗户角、旗杆尖，因为那里信息量最大。本项目用的 **ORB** 特征就是这么一对组合拳：用 **FAST** 算法飞快地找出角点，再用 **BRIEF** 给每个角点生成一段 256 位的二进制“指纹”，合称 ORB（Oriented FAST and Rotated BRIEF）。

匹配（matching）就是拿指纹互相打分：两段指纹越像，越可能是“同一个点”。打分的思想你其实学过——第 0 讲的点积就是最早的“合拍程度打分器”：对应位置相乘再相加，分高即相似。二

进制指纹的比对用“数不同位的个数”（汉明距离），思路同源：**都是把“像不像”变成一次可计算的算术**。每幅图取多少个特征点？本项目目前定为每图 1500 点——这是个写死的人工经验值，记住它，1.4 节它会成为被改造的对象之一。

### 1.2.3　第三步：几何对齐——一张 3×3 表格把画面“扶正”

找到了几十上百对“同名点”，下一步是求出两幅画面之间的**坐标换算表**：右图的点该挪到左图的哪里？这张换算表就是大名鼎鼎的**单应矩阵**（homography）。它的直觉见下图：同一块平面（一面墙、一片草坪）在两幅画面里各留了投影，单应矩阵能把一块平面上的四角精确搬到另一块平面的四角上——相当于把一张拍歪的照片“扶正”再“平移”到位。

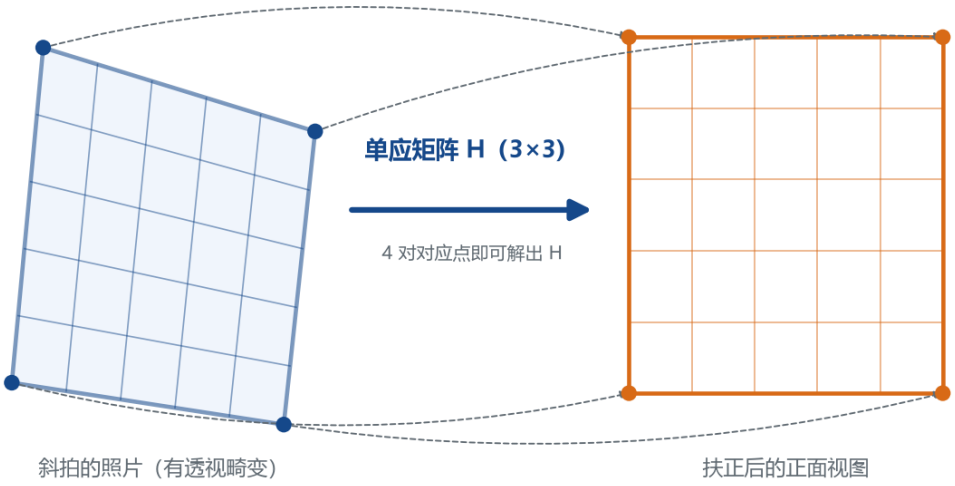



图 1.7　单应变换示意：同一平面在两幅画面中的投影之间存在一张 3×3 的“换算表”，能把平面 A 的四角映射到平面 B 的四角——斜拍的照片由此被“扶正”

**知识补齐：3×3 矩阵、8 个自由度与 RANSAC 一句话**

**单应矩阵是 9 个数排成的 3×3 方阵，却只有 8 个自由度：**因为整体缩放不改变映射效果（所有数翻倍，同一个点算出的位置不变），习惯上约定最后一个数为 1，于是真正待定的数只有 8 个。每对同名点能提供 2 个方程，所以**最少 4 对点就能解出它**——这就是为什么各种拼接软件都要求相邻照片留出重叠、并在重叠处找到至少几个可靠同名点。推导不必急，先记住直觉：**单应 = 两块平面之间的一张坐标换算表**。**RANSAC 一句话：**4 对点就能解一个单应，可万一其中一对是误匹配的“卧底”呢？**随机抽样一致**（RANSAC, Random Sample Consensus）的对策是赌概率：随机抽 4 对算一个解，拿它到全体点里验票，看多少对点“同意”这个换算表；换几批重抽，最后票数最多的解胜出——少数卧底翻不了盘。

360Stitcher 在这一步还有自己的工程取舍：六台相机刚性固定成一圈，相邻相机之间**只有水平方向的相对偏移，没有旋转和俯仰**，于是系统采用比完整单应更简单的**纯平移几何模型**——参数更少，解得更快，抗噪声也更稳。此外，为了让相邻帧的跟踪不每帧从零算起，系统用**相位相关**（phase correlation）在频率域里直接测出两帧间的平移量，作为帧间跟踪的“惯性基准”，ORB 匹配只在这个小范围里精修。

### 1.2.4　第四步：缝合线——让“刀口”藏在最不显眼的地方

对齐之后，重叠带里两幅画面“说的是同一件事”，但取哪边？最笨的办法是沿中线一刀切——可刀口要是穿过一辆正在移动的自行车或一盏忽明忽暗的灯，接缝处立刻穿帮。**缝合线**（seam）的使命，是在重叠带里找出一条**差异最小**的路径当刀口：让它沿着草地、墙面这些两幅画面最一致的地方走，绕开行人、车灯与反光。

找这条最优路径用的正是**动态规划**（Dynamic Programming, DP）：把重叠带想象成一张网格，每个格子的“代价”是两幅画面在该处的差异，缝合线是从上端走到下端的一条路径，DP 保证逐列累积代价最小时全局最优——像在山谷里找一条翻越起伏最小的山脊线。这一步的输出是一条逐像素的路径，告诉融合步骤“线的左边信左眼，右边信右眼”。

### 1.2.5　第五步：融合——把接缝“羽化”到看不见

即便对齐和走线都完美，两台相机的曝光、增益也总有细微差别，硬接仍会留下一道可觉察的色阶。**融合**（blending）负责最后一层柔化，本项目备着两种手段。**feather 融合**（羽化融合）：在缝合线两侧各划一个过渡带，两幅画面按“离线有多远”加权混合——越靠近左眼的地盘左眼权重越高，反之亦然，颜色渐变，肉眼无痕。**多频段融合**（multi-band blending）：把画面按空间频率分解成若干层（可以想成“大色块层”到“细纹理层”），低频层用很宽的过渡慢慢混，高频层用很窄的过渡快速换——大尺度亮度平稳过渡、小尺度细节不失真，是全景软件的看家绝活。而在融合之前，通常还有一步**增益补偿**（gain compensation）：按各相机重叠区的亮度统计，给每路画面乘一个系数，先把整体明暗拉齐，再让羽化收尾。

**动手时间：用手机预演一遍五步曲**

不用写一行代码，手机就能把本章五步全走一遍。**步骤**：(1)打开相机 App 的**全景模式**，原地缓摇，拍一张 360° 或 180° 全景——这就是“实时拼接”的单帧版；(2) 再切普通模式，对着同一场景拍两张水平错开、重叠约三分之一宽度的照片；(3) 放大重叠区，找出至少 4 对“同一个点”（窗角、标志牌、砖缝交点都行），在纸上记下它们在两张照片里的位置——你刚刚手工完成了“特征匹配”;(4) 数一数：4 对点、每对 2 个方程、单应矩阵 8 个自由度，恰好够解——这是第三步的最小配置；(5) 回到那张全景照片，找出缝合线走过的位置和一处你觉得“接得不太完美”的地方。**验收**：能向同学指出你那张全景照里“第五步融合”最好与最差的地方各一处，并说清为什么。**加分项**：故意在拍摄两张照片时把其中一张靠近镜头的手指拍进重叠区，看看全景模式会不会把它切掉或留下“半个你”——想想这是第四步缝合线在保护谁。

## 1.3　360Stitcher 长什么样：六个模块与一串硬数字

五步曲是“怎么拼”，现在看“怎么跑”。360Stitcher的设计哲学是**离线建模 + 在线执行**：标定、内参、相机次序这类一锤子买卖全部离线做好、存成配置；运行时只做必须每帧做的事。整条数据通路是一条流水线，六个工位见下表（职责按模块名与项目组公开结构整理）。

**表 1.1**　360Stitcher 的模块链：六路画面从相机到长卷的旅程

| 顺序 | 模块 | 职责（一句话） |
|---|---|---|
| 1 | MainWindow | Qt6界面门面：预览、按钮、编辑模式，人只跟它说话 |
| 2 | HikCamera | 对接海康相机 SDK，把六路大图从相机取进来 |
| 3 | CameraWorker | 每路相机一个工作线程，专职取流、不掉帧 |
| 4 | FrameSubmitter | 把六路几乎同一时刻的帧配成一组，排队提交 |
| 5 | AsyncStitcher | 异步调度：收到一组帧就交给拼接核心，绝不堵塞取流与界面 |
| 6 | FastStitcher | 拼接核心：ORB 匹配 + 纯平移模型 + 相位相关跟踪 + DP 缝合线 + 多频段/feather 融合 |

相机本体是海康工业相机（下图）。它与家用摄像头的区别不在“清楚”而在“守时”：工业相机的曝光与出帧由时钟或外部信号统一触发，六路画面的时间戳对得齐齐整整——要是六只眼睛各看各的时刻，运动物体在接缝处会凭空瞬移，拼什么都白拼。



**图 1.8**　工业相机示例（海康威视等品牌）：与家用摄像头相比，工业相机的价值在于分辨率、帧率稳定与同步触发——六路画面“同一时刻”才拼得起来

图片来源：海康威视（Hikvision）机器视觉官方产品图，仅用于课堂教学

性能上，系统挂着一串值得背下来的硬数字，见下表。注意“帧周期约 33 ms”与“FPS 天花板约 16”的关系：相机每 33 ms 交来一组新帧（约 30 fps），而 8 图静态基准测出的拼接吞吐天花板只有约 16 FPS——来料速度接近出料速度的两倍。**流水线两头速度不匹配，是所有实时系统的日常**，也是 1.4 节“参数头必须又小又快”的物理根源。

**表 1.2**　360Stitcher 性能数字速查

| 指标 | 数值 | 说明 |
|---|---|---|
| 相机原始分辨率 | 4608×3456（约 16 MP） | 六路海康相机每路的大图 |
| 拼接工作分辨率 | 1440×1080 | 缩小后参与拼接运算，拿分辨率换实时性 |
| 相机帧周期 | 约 33 ms | 相机每约 33 ms 交来一组新帧（约 30 fps） |
| 8 图静态拼接吞吐 | 天花板约 16 FPS | 基准测试下拼接核心的吞吐上限 |
| 界面实测帧率 | 约 18–27 FPS | 本章各截图右下角读数的波动范围 |
| ORB 特征点数 | 1500 点/图 | 人工经验值，1.4 节起待参数头接管 |

系统跑起来什么样？看下面这张遥测（telemetry）截图：终端里每帧滚出一行 `[Latency]` 日志，四个字段分别是 total（单帧总延迟）、stitch（拼接耗时）、frameAge（帧龄：这帧从相机出发到现在过了多久）、uiLatency（界面延迟）。截图里 total 约 32–63 ms，其中拼接核心稳定在 29–45 ms——系统正贴着 33 ms 的帧周期“低空飞行”。这套逐帧自报家门的数据流，就是你们第三篇任务书里“遥测”两个字的本体：**参数头好不好，最终都由这行日志和它背后的画面质量说了算**。

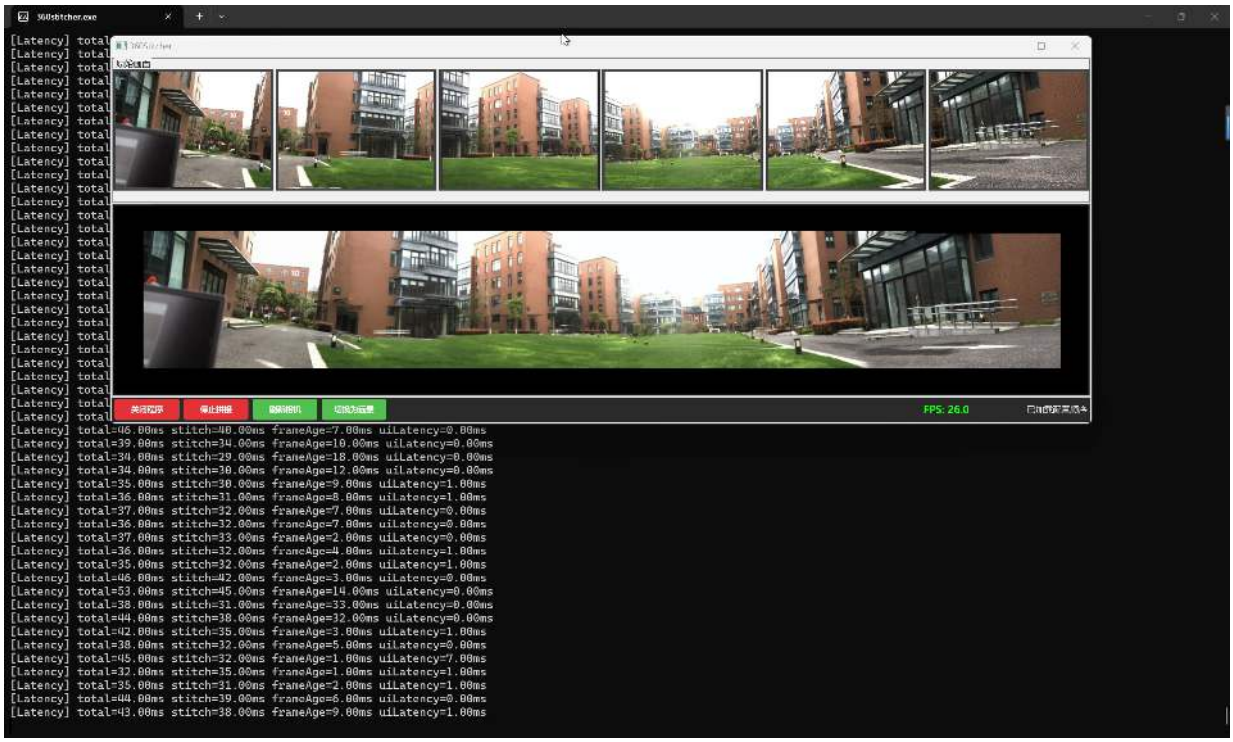



**图 1.9**　系统运行遥测实录：终端每帧打印一行 [Latency] 日志（total 总延迟 / stitch 拼接耗时 / frameAge 帧龄 / uiLatency 界面延迟），前景 360Stitcher 正以 FPS 26.0 运行

图片来源：360Stitcher 项目组真实运行截图（遥测终端实录）

界面本身也有讲究。下图是室内场景：六路画面拍的是同一间办公室，左侧白板上那块黑白棋盘格正是第一步标定的道具。更值得注意的是底部工具栏——编辑模式提供“调整顺序、上移、下移、保存顺序”等按钮：六台相机谁挨着谁、长卷从哪路开始，都可以在界面里调好后存成配置。这再次呼应“离线建模 + 在线执行”：**凡是能提前想清楚的，都不占用每帧 33 ms 的宝贵预算**。

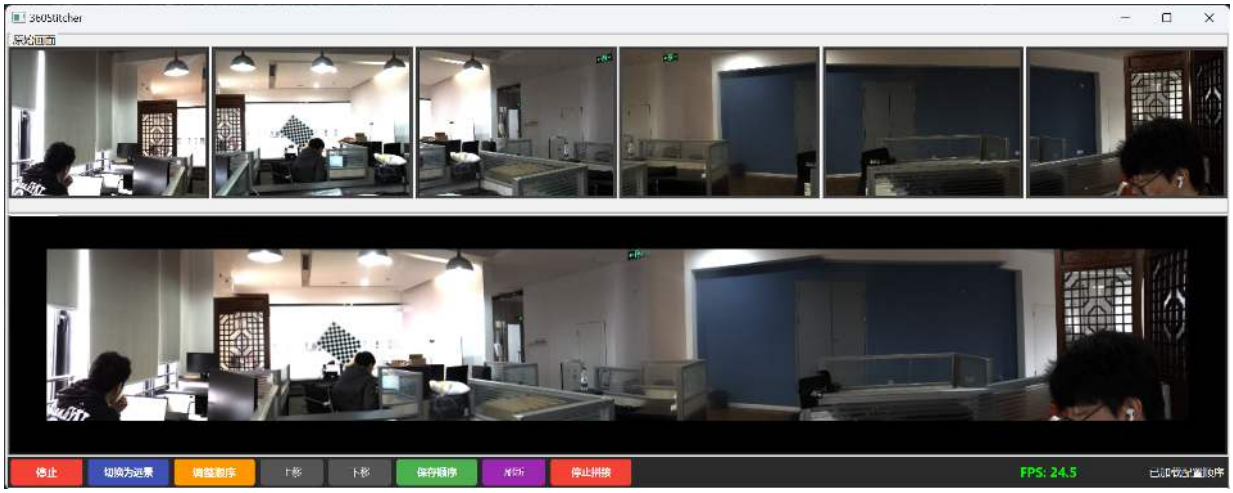



**图 1.10**　360Stitcher 室内场景与编辑模式：六路画面拍摄同一间办公室，白板上放置棋盘格标定板；底部“调整顺序/上移/下移/保存顺序”等按钮用于配置六路相机在全景中的排列次序

图片来源：360Stitcher 项目组真实运行截图（六目拼接项目素材）

最后一条，也是最容易被忽视的一条需求：**长期稳定运行**。监控类系统一开就是几周几个月，任何一次崩溃都是安全事故的种子。这个约束现在看似平淡，等你在 1.4 节见到“回退保护”时就会明白它有多深远——它甚至决定了整个项目敢不敢让神经网络上场。

## 1.4　为什么要“可学习”的参数：九个参数头

现在的 360Stitcher 是一位“经验丰富的老师傅”：ORB 取 1500 点、匹配的比例阈值、RANSAC 的距离阈值、融合的过渡带宽……九类关键参数全是人工经验值，写死在代码里或靠工程师现场手调。老师傅在熟悉的场子（本章截图那种光照均匀的园区）手艺精湛；可场景一换——傍晚逆光、雨后反光、相机被碰歪半度——质量立刻退化。下面这张截图就是一个“吃力现场”：从高楼俯瞰、天色将暗，画面大面积逆光且天空占比极高，长卷上部天空的接缝处已经泛起淡淡的拼接痕迹，FPS 也从二十几掉到 18.8。**这不是 bug，是固定参数的哲学极限**：为场景 A 调的参数，天生不可能同时是场景 B 的最优。

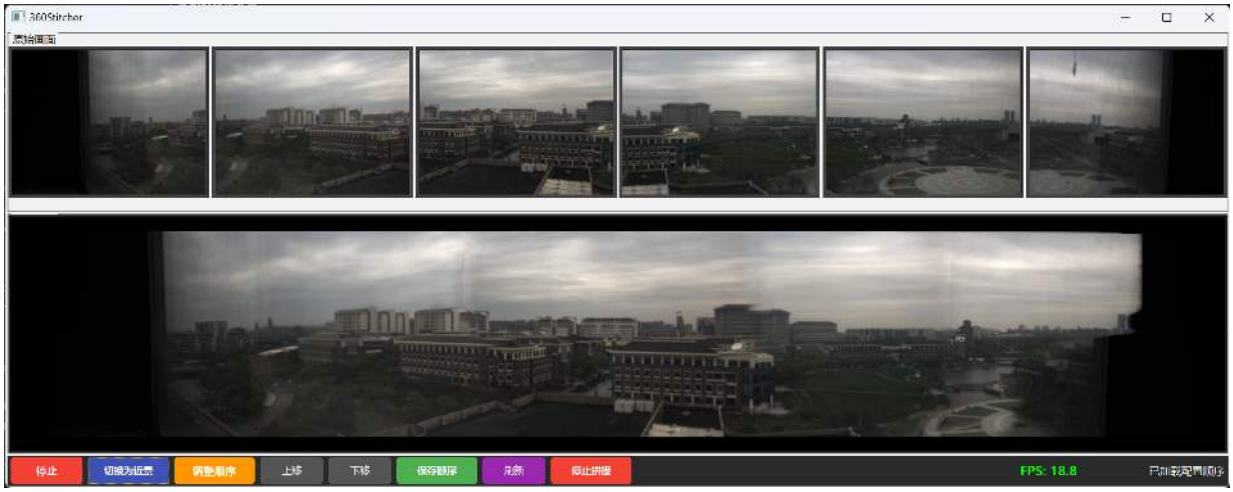



**图 1.11**　一个“吃力现场”（傍晚俯瞰）：逆光、低照度加大面积天空，长卷上部天空接缝处出现淡淡的拼接痕迹，FPS 掉到 18.8——固定参数换场景后开始露怯，这正是本项目要解决的问题

图片来源：360Stitcher 项目组真实运行截图（六目拼接项目素材）

工程上不是没试过改良。常见思路是**统计自适应**：让参数跟着统计量自动走，比如按重叠区亮度差自动调节融合强度、按匹配点对的分布自动收紧阈值。这类办法能救一部分场，但天花板明显：**规则终究是人写的**——你只能覆盖想得到的情况，而现场的场景组合是写不完的。光照、纹理、运动物体、相机微动，任何两两组合都可能让某条手写规则失灵，维护它们比重调参数还累。

项目的答案，是把九类参数从“写死的数”升级为**九个可学习的“参数头”**（parameter head）。参数头是挂在网络末端的一个小型神经网络分支：吃进系统现成的统计特征或特征图，吐出就该由“老师傅”凭经验给的那几个参数。九个头的名字、输入输出形状与职责见下表（职责按头的命名与形状整理，逐字定义以第三篇任务书与代码为准）。方括号是**张量形状**记号：B 表示一次一起送进网络的一批样本数（batch，批次），T 表示时间步数；[B,8] 就是“一批若干行、每行 8 个数”。

**表 1.3**　九个参数头一览：输入 → 输出与一句话职责

| 参数头 | 形状（输入 → 输出） | 一句话职责 |
|---|---|---|
| policy 匹配决策 | [B,8] → [B,3] | 看 8 项匹配统计量，三选一决定这一对画面走哪条拼接策略 |
| mc 匹配质量 | [B,10] → [B,1] | 给本轮特征匹配打个置信分：分低就别硬拼 |
| geometry 几何参数 | [B,16] → [B,5] | 从 16 项统计特征回归 5 个几何参数，接管手调的几何阈值 |
| feature_uncertainty 特征不确定性 | [B,1,32,32] → [B,3] | 扫一遍特征图，报出 3 个“这帧特征有多不可靠”的指标 |
| descriptor_softening 描述子软化 | [B,1,32,32] → [B,256] | 为 256 维描述子的比对生成软权重，让“生硬的位比对”学会看场合 |
| temporal_state 时序状态 | [B,T,24] → [B,4]×2 | 读最近 T 帧的 24 维时序特征，输出两个 4 维状态：跟踪该信谁、要不要重置 |
| seam_uncertainty 缝合不确定性 | [B,2,128,256] → [B,1,128,256] | 逐像素标注重叠区里“哪里走缝容易穿帮” |
| gain_field 增益补偿 | [B,12] → [B,6] | 从 12 个亮度统计量算 6 个增益系数，接管手动增益补偿 |
| fusion 融合参数 | [B,12] → [B,3] | 从 12 项统计量算 3 个融合权重，决定多频段/feather 怎么混 |

九个头的组织方式见下图：从拼接系统引出九条分支，三个颜色分组恰好对应三个小组（组 1 管匹配与几何三头，组 2 管特征与时序三头，组 3 管缝合与融合三头）；每个头**独立训练、独立交付、逐头验收**，交付格式统一为 **ONNX**（Open Neural Network Exchange，开放神经网络交换格式——一种跨框架通用的模型文件格式，训练用 PyTorch，部署用 C++ 运行时加载同一个文件）。训练好一个头，就在系统里替换掉一类人工参数；没训练好，老参数原样顶岗，互不拖累。

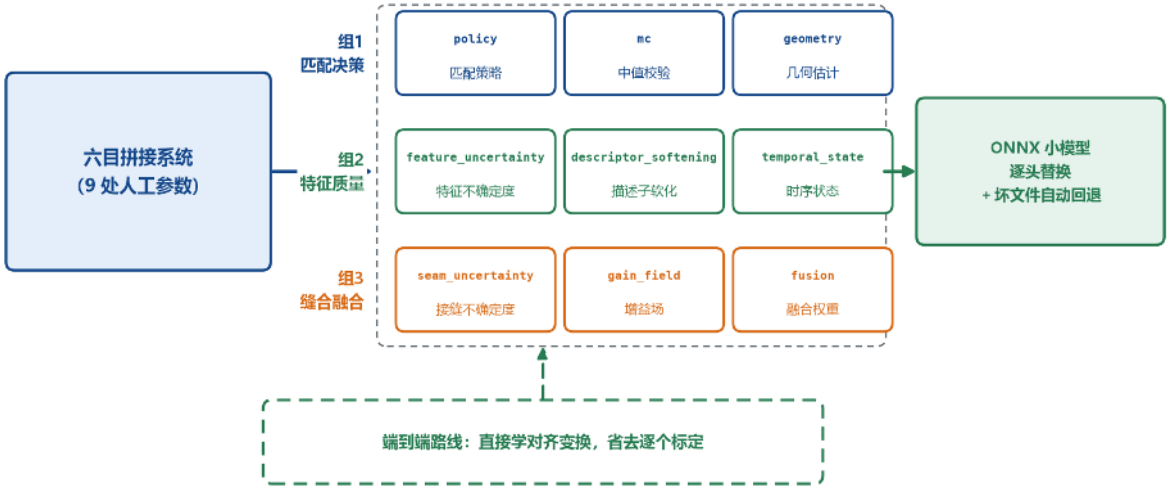



**图 1.12**　九头学习路线总览：拼接系统把统计特征交给 9 个参数头（三组颜色对应三个组），每个头独立训练、转成 ONNX 交付，身后站着回退保护

而让神经网络敢上岗的最后一道闸，是**回退保护**（fallback）。每个参数头加载时走 `[ProviderFallback]` 机制：模型文件损坏、推理出错、输出越界，系统不崩溃、不报异常退出，而是**自动退回原来的人工参数**，日志里记一笔，继续拼接。想通这件事你会拍案叫绝：参数头的产出只是“参数”，而参数错了的后果不过是“这一帧拼得没原来好”——**交付质量不合格，只会没用，不会坏事**。这条哲学给整个项目松了绑：大胆训练、大胆交付、快速迭代，最坏结局也只是回到起点，而每一次成功替换都实打实减一分人工调参的负担。

**项目连线：九个参数头，对应第一篇的哪些讲**

九个头不是天外来物，每个都能在第一篇找到血缘：policy、mc、geometry 这些“吃几个统计量、吐几个数”的小头，骨子里是第 1 讲的感知器加第 2 讲的反向传播——小型全连接网络；feature_uncertainty 与 seam_uncertainty 吃 32×32、128×256 的特征图，非第 5 讲的卷积网络莫属；temporal_state 读最近 T 帧，是第 9–10 讲时序模型的领地；交付阶段的 ONNX 与 int8 压缩，用的正是第 8 讲轻量化与量化的手艺；而“逐头训练、逐头验收”的方法论，全盘来自第 3 讲的训练与评估。第三篇任务书下达每个头的任务时，都会回指这些讲次——**第一篇学得不虚，第二篇才接得住**。

## 1.5　双路线对照：逐个学头，还是整网端到端

站在 2026 年的技术路口，改造一个拼接系统其实有两条路，本项目**两条同时修**。**路线一**就是上文的主角：九个参数头逐个学习——保留传统流水线的骨架，只把九处“老师傅经验”换成小网络。它的优点是渐进、可控、可回退：每个头形状明确、单独验收，坏了退回老参数；缺点也明摆着：九个头各自为政，头与头之间的联动（比如匹配质量影响缝合线选择）没人统筹。**路线二**则釜底抽薪：训练一个**端到端**（end-to-end）网络——代号 RopStitch——直接吃进相邻相机的画面，一次性输出对齐变换，跳过“手工五步曲”里最难的几何估计环节。路线二在实验室的 RTX 4080 显卡上训练，标准命令 `--epochs 200`（把全部训练数据完整过 200 遍），产物是 `ropstitch.onnx`，用探针验收要达到 5/5 全过。

为什么要花双倍力气走两条路？两个理由，一实一虚。**实的那条是工程保险**：单一技术路线一旦中途受挫（数据不够、精度不达、速度超标），项目就没有退路；两条路线共享组 6 的数据、互相校验，一条受阻，另一条照常交付。**虚的那条其实是科学方法**：只有 A/B 对照才能回答“学习式方法到底比人工参数好多少”这个严肃问题——同一段六路视频，老算法拼一遍、九头版拼一遍、端到端版拼一遍，三个成绩摆上桌，结论才有分量。这批对照结果还有更远的去处：项目组正把成果整理成论文投稿 *Image and Vision Computing* 并支撑基金申报——你在组 5 评测时跑出的那张 A/B 对比表，可能就是论文里的一行关键数据。

## 1.6　八个组怎么分工：一条流水线上的八个工位

先看清队伍全貌。项目由导师统筹，研究生袁博总体主责（九头方案与端到端路线的技术负责人），8个本科小组每组 5 人，另有 2 位毕业设计学生深度参与。八个组的名字、任务与数据来源见下表——把它和 1.2 节的五步曲、1.4 节的九个头对着看，你会发现**分工图就是系统图的镜像**：组 1–3 各领三个参数头，组 4 供给训练口粮，组 5 手握标尺，组 6–8 在另一条路线上复刻“数据、训练、优化”三段论。

**表 1.4**　八个小组分工与数据来源（公开数据先行，实验四接真机）

| 组号 | 组名 | 一句话任务 | 数据来源 |
|---|---|---|---|
| 组 1 | 匹配决策组 | 训练并交付 policy/mc/geometry 三个头 | 组 4 公开仿真标签（COCO/MegaDepth → simulation）开局即训；真机标签实验四接入 |
| 组 2 | 特征质量组 | 训练并交付 feature_uncertainty/descriptor_softening/temporal_state 三个头 | 仿真时序标签 + HPatches/MegaDepth patch 开局即训；真 patch 第 11 周迁移转真 |
| 组 3 | 缝合融合组 | 训练并交付 seam_uncertainty/gain_field/fusion 三个头 | 仿真标签开局即训；fusion 彩排先行，正式版第 13 周放行 |
| 组 4 | 数据与标签组 | 公开图对喂 simulation 管线产仿真标签（开局）；录六路视频制真机标签（实验三） | 组 1–3 第一周即吃你的仿真标签；真机版实验四接入 |
| 组 5 | 测试集与评测组 | 双台账（指标台账 + 数据量台账）、A/B 对比 `ab_compare.py`、回归脚本 `run_regression.ps1` | 上游组 1–4；产出全项目成绩单 |
| 组 6 | 端到端数据组 | 建 50 对以上数据集（UDIS-D 公开对→真机对）与 benchmark（基准测试集） | 组 7 实验三开训公开版；组 8 同步开工 |
| 组 7 | 端到端训练组 | RTX 4080 上 `--epochs 200` 训练 `ropstitch.onnx`，探针 5/5（实验三出公开版 v1） | 组 6 UDIS-D 数据；真机数据实验四续训 v2 |
| 组 8 | 端到端优化组 | 引擎单次调用 ≤300 ms、总预算 800 ms；fp32/fp16/int8 三档对比，int8 劣化 <10% 才交付 | 复现公开版 v1 即可开工；真机版 v2 联调日补做 |

八张工位图里藏着三条硬规矩。其一，**组 5 的双台账是全项目的账本**：每个头训练用了多少数据（数据量台账）、验收指标是多少（指标台账），笔笔入库，杜绝“差不多就行”。其二，**组 8 的预算是物理红线**：参数头再聪明，若把整帧拖过预算，实时性破产即全盘皆输——fp32/fp16/int8 三档精度逐一测速测精度，int8（8 位整数量化，第 8 讲的老朋友）劣化超过 10% 就不许交付。其三，下

面这个红盒子里的事，比以上所有都重要。

**防糊弄五条：本项目的成绩护城河**

**(1) 证据包制**：任何“我做出来了”必须附命令、日志、产物三件套，缺一不算。**(2) 探针防绕过**：系统内置探针脚本（如部署验收的 `nine_head_probe.py` 要求 12/12 通过、端到端探针 5/5），探针不过，人工演示再漂亮也不算数——它专防“跳过步骤装样子”。**(3) A/B 数据说话**：新方法好不好，由 `ab_compare.py` 在同一批数据上的对比结果说了算，不由 PPT 说了算。**(4) 双台账**：指标台账与数据量台账笔笔可查，训练用的每一张图都有出处。**(5) 如实条款**：不得虚构训练精度与实验结果，**发现一处，该节点成绩作废**——工程世界里，造假不是扣分，是出局。

**前沿速递：拼接领域的“学习式改造”进行到哪了（2024–2026）**

全景拼接正在被深度学习逐个工序地“换零件”。特征环节，SuperPoint 用自监督学习学出比手工 ORB 更稳的特征点；匹配环节，SuperGlue（2020）用图神经网络给特征点“拉郎配”，其 2024 年的蒸馏版 GIM 把这种匹配能力推广到了新场景；LoFTR（2021 起持续演进）更进一步，干脆放弃“先找点再匹配”，在特征图上直接做全局注意力匹配。对齐与融合环节，UDIS++（2023）等端到端拼接网络把“估计变换 + 网格重排 + 融合”一并学掉。2024–2026 年的趋势是拼接与基础模型、扩散模型进一步合流，视频流式拼接与开放场景鲁棒性成为热点。但要紧的判断是：**工业现场落地的主流仍是“传统流水线 + 学习式参数头”的混合路线**——纯端到端在可控性、实时性与可解释性上尚未过关。你们的 360Stitcher，恰好一屁股坐在当前工业界的主流路线上。

**项目连线：四个实验节点，在六目拼接里各干什么**

把 1.4–1.6 节的地图对到日历上：**实验一**（10 月 12 日）用大模型 agent 解读 360Stitcher 代码，画出模块链与数据流；**实验二**（11 月 2 日）本机部署——VS2022 编译通过、日志里出现九行 [Provider] 加载记录、`nine_head_probe.py` 探针 12/12 全绿；**实验三**（11 月 23 日）真训练、真交付、真 A/B——各头 ONNX 上系统，与老参数同场比武；**实验四**（12 月 21 日）复训与优化——精度、速度双复盘，int8 三档对比收官。之后是第 19 周周五（2027 年 1 月 22 日）完全提交离场。每个节点的命令级步骤、验收证据与常见坑，见第三篇第 1–8 组任务书的“四次实验安排”一节。

## 1.7　你将用到第一篇的哪些讲

第二篇读到这里，最后一件事是把项目与课堂焊在一起。下表给出 12 讲内容在六目拼接里的落点——别把它当复习提纲背诵，它是你以后“学了一讲就回来对号入座”的索引。

表 1.5　第一篇 12 讲在六目拼接项目中的落点

| 讲次 | 这讲给你什么 | 用在项目一的哪里 |
|---|---|---|
| 第 0 讲 | 点积：合拍程度打分器 | 描述子匹配与 policy/mc 头打分的直觉原点 |
| 第 1 讲 | 感知器与学习规则 | 一切参数头的祖宗：加权求和出参数 |
| 第 2 讲 | 反向传播 | 九个头逐头训练的引擎 |
| 第 3 讲 | 训练、验证与评估 | 逐头验收、A/B 对比、防过拟合的方法论 |
| 第 4 讲 | 自编码器：压缩与重建 | “特征图进、小向量出”的头设计思路 |
| 第 5 讲 | 卷积与特征图 | 读懂 feature_uncertainty 等头的输入 |
| 第 6 讲 | 数据增强与训练技巧 | 组 4 数据制作与各头训练的调参手艺 |
| 第 7 讲 | 残差网络与演化 | 路线二端到端网络的骨架 |
| 第 8 讲 | 轻量化与量化 | 组 8 的 fp32/fp16/int8 三档与 ONNX 部署 |
| 第 9 讲 | 循环神经网络 | temporal_state 头的时序建模 |
| 第 10 讲 | LSTM/GRU 门控记忆 | 时序状态“该记的记、该忘的忘” |
| 第 11 讲 | 注意力机制 | 学习式匹配（SuperGlue/LoFTR 一脉）的钥匙 |
| 第 12 讲 | 生成对抗思想 | 融合质量评估可借鉴“判别器”的挑剔眼光 |

对照之下你会发现一个规律：**课程进度与项目进度是咬合的**。10月中你在课堂上学反向传播，11月底项目里正好要交第一批训练好的参数头；12 月讲轻量化与量化，组 8 的 int8 优化恰好进入收官。这不是巧合，而是这门课的设计初衷：**学的每一样东西，几周内就在项目里用一次**。所以当第一篇某讲让你觉得抽象时，翻回本章对应行，看看它在真实系统里的模样——抽象概念一旦有了月薪 33 ms 的工作岗位，就好懂多了。

**畸变校正（distortion correction）**

用标定得到的镜头参数，把镜头造成的“直线变弯”反向掰直的预处理；本项目离线标定、在线逐帧执行。

**ORB**

特征提取组合拳：FAST 找角点 + BRIEF 生成 256 位二进制指纹，快而够用，是 360Stitcher 的特征引擎。

**描述子（descriptor）**

特征点的“指纹”：一串能代表该点外观的数值，匹配即指纹比对打分。

**单应矩阵（homography）**

两幅画面平面之间的 3×3 坐标换算表，8 个自由度，最少 4 对同名点可解。

**RANSAC（Random Sample Consensus，随机抽样一致）**

随机抽最小点集算解、全场投票、票多者胜的稳健估计法，专治误匹配。

**缝合线（seam）**

重叠区里“左图说了算”与“右图说了算”的分界路径，由动态规划找差异最小的一条。

**多频段融合（multi-band blending）**

把图像按空间频率分层、低频宽过渡高频窄过渡的融合法，消除接缝色阶的主力。

**feather 融合（羽化融合）**

在缝合线两侧按距离加权混色的简单融合法，多频段的轻量替身。

**增益补偿（gain compensation）**

按重叠区亮度统计给各路画面乘系数，先把整体明暗拉齐再融合。

**参数头（parameter head）**

挂在网络末端的小型分支：吃统计特征、吐系统参数；本项目九个头逐个替换九类人工参数。

**ONNX（Open Neural Network Exchange）**

跨框架通用的模型文件格式：训练用 PyTorch，交付用 ONNX，C++ 运行时直接加载。

**回退（fallback）**

参数头加载失败或输出异常时自动退回人工参数的保命机制（[ProviderFallback]），保证“只会没用、不会坏事”。

**遥测（telemetry）**

系统逐帧自报的家门：[Latency] 日志里的 total/stitch/frameAge/uiLatency，评测与调试的第一手证据。

**A/B 对比**

新旧方法在同一批数据上的对照评测（工具 `ab_compare.py`），项目里“谁更好”的唯一裁判。

# 项目二　水下双目视觉自动标记：让 AI 看懂海底世界

**本章地图**

**讲什么**：项目二的“总剧本”——从一段真实的 ROV 锚链检查视频出发：场景与需求；水下成像的四道坎；双目立体视觉原理（视差、基线、校正）；没有深度真值时的自动打标签流水线（本章高潮）；我们亲手构建的 500 对数据集；四个小组的分工与红线。**为什么值得学**：这是三个项目里“最像科幻”的一个——输入一对水下相机拍到的左右画面，输出每个像素的深度，并知道每个像素的结果有多可信。它把第一篇的卷积、注意力、蒸馏、部署全部串成一串，还要在真实 GPU 上真刀真枪地跑。**学完你能**：看懂 $z = bf/d$ 并手算深度；说清“没有真值怎么造标签”的三步流水线；明白你们组第 5–19 周的四个实验、人工必做与验收线。

先看一张“海上炼油厂”的照片。FPSO（Floating Production Storage and Offloading，浮式生产储卸油装置）是一种比足球场还长的大船：海底油井采出的原油在船上加工、储存，再卖给运油船——相当于把炼油厂从陆地搬到了海上。它没有腿，不能像固定平台那样站在海床上，那靠什么不被风浪吹跑？答案朴素得吓人：**用巨大的锚链把自己拴在海床上**。锚链一旦断裂，船就会在风浪里“跑锚”，输油管被撕扯、被迫停产甚至泄漏——对油田来说是大事故。所以行业规范要求：系泊系统的每根锚链，都必须定期下水体检。



**图 2.1**　FPSO（浮式生产储卸油装置）示例：海底油田的“海上炼油厂”，靠系泊锚链固定在作业海域

图片来源：Wikimedia Commons，FPSO 作业照片（CC 授权转载）

体检靠谁下水？靠 ROV（Remotely Operated Vehicle，遥控潜水器）：母船通过一根“脐带缆”给它供电、传视频，操作员在船上遥控它下潜一两百米，沿着锚链一寸一寸地拍。它是本项目的“摄影师”——没有它，就没有后面的一切数据。

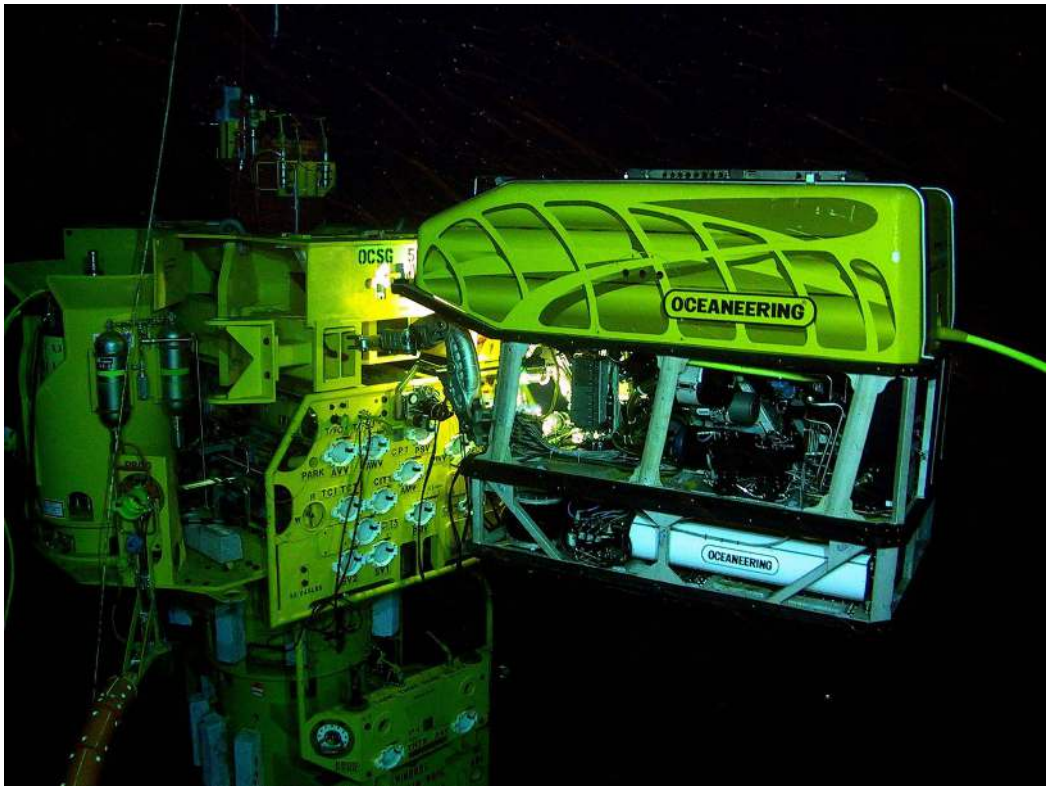



**图 2.2**　ROV 水下作业示例：由母船缆绳供电与通信，携带相机与传感器下水检查水下结构物

图片来源：NOAA Ocean Exploration / Wikimedia Commons，ROV 作业照片（公有领域授权）

这不是教科书上的摆拍。下面这一帧，就是我们拿到的**真实项目素材**——请先仔细“读”一遍这张图，练一练看图说话的功夫：画面中央斜穿而过的那根亮色“柱子”，就是长满海生物的锚链本体，周围漂浮的白色小点是水中的悬浮颗粒（俗称“海雪”）；左上角叠着一整块字符，叫 OSD（On-Screen Display，屏上显示），是作业设备“盖章”在视频画面上的遥测信息——任务名“HYSY123 FPSO Mooring Line #1 Inspection”（“海洋石油 123”号 FPSO 1 号系泊缆检查）、UTM 坐标、深度 186.0 米、离底高度 ALT 11.8 米、航向 HDG 158.0；右上角是作业方 COSL FUGRO 的标志；右下角是拍摄时间戳。凭这张图你已经可以“报出”拍摄位置和深度——**但画面里每个像素离相机多远，图上什么都没说**。这正是甲方要我们解决的问题。




**图 2.3**　本项目真实素材：HYSY123 FPSO 1 号锚腿 ROV 检查视频帧——亮色斜条为长满海生物的锚链，左上角 OSD 遥测给出任务名、UTM 坐标、深度 186.0 m、离底高度 ALT 11.8 m 与航向，右上角为 COSL FUGRO 标志，右下角为时间戳

图片来源：本项目真实数据（E:\锚链视频，2024 年 8 月 31 日 1 号锚腿检查视频）

## 2.1　场景与需求：甲方要“AI 看一眼就知道有多远”

先把镜头拉远。海上油气的开采与运输，是围绕海洋平台与 FPSO 展开的：平台采油、FPSO 储卸、穿梭油轮外运，而把它们与海床“拴”在一起的，正是系泊系统与锚链。这些钢铁巨物常年泡在海水里，腐蚀、磨损、海生物附着，任何一处疲劳断裂都可能酿成大事故，所以每年都要派 ROV 下水巡检。



**图 2.4**　海洋石油平台示例：海上油气开发的前线，其系泊与水下结构物需要 ROV 定期巡检

图片来源：Wikimedia Commons，海洋石油平台照片（CC 授权转载）

需求就从这里来。ROV 每年巡检拍下的视频是**海量**的：一次检查动辄几个小时、几个硬盘，而看片的只能是少数有经验的老工程师——盯着屏幕逐段回放，判断“这段链环有没有损伤、离相机大概多远”。又贵、又慢、又主观，看视频的人眼连“这段链条离镜头 8 米还是 12 米”都说不准。甲方（中海油，CNOOC）的诉求可以概括成三句话：

1. **构建数据集**：从海量检查视频里，构建出能训练、能测试的立体视觉数据集——视频不是数据，能进训练集的才是数据。
2. **解决标注难题**：水下没有深度真值（第 2.2 节细讲为什么），怎么给每对图像打出“每个像素多远”的标签？这是全项目最硬的一颗钉子。
3. **训练并交报告**：在数据集上完成训练实验，把结果与结论整理成一份能交付的训练报告。

整个项目的大图景，值得单独抄一遍，后面所有工作都为它服务：**输入一对水下双目相机拍到的左右画面，输出每个像素的深度，并知道每个像素的结果有多可信。**“每个像素有多远”叫稠密深度，“有多可信”叫置信度——第 2.4 节你会看到，后者恰恰是自动打标签能不能“自证清白”的关键。

团队配置：项目负责人张洪进，研究助理 1 名，大三本科生 4 个组（第 5–19 周进场，每周投入 6–8 小时），毕设生 1 名负责算法主线。四个组各守一条并行赛道、实验四合并联调，见第 2.6 节；每个人的具体任务书在第三篇（第 9 章至第 12 章）。

## 2.2　水下为什么难：光、匹配、泛化、真值四道坎

有人会问：立体测距在空气里早就是成熟技术了，自动驾驶车顶的双目相机都能量产，把它直接搬下水不就行了？恰恰不行。水把这件事的全部前提都改了。我们数出四道坎，每一道都对应后文的

一招“化劲”。



**图 2.5**　浑浊水下成像示例：悬浮颗粒与散射让画面发灰、对比度下降，这是水下视觉的常态

图片来源：Wikimedia Commons，浑浊水下照片（CC 授权转载）

### 2.2.1　第一道坎：光被水“吃掉”

太阳光扎进海水，会同时遭到**吸收**与**散射**的双重消耗，而且消耗是“挑食”的：不同波长的光被吸收的程度不同。看下图：红光波长最长，最先被水吃掉，几米之内就所剩无几；接着是橙、黄；能传得最远的只剩蓝、绿。这就是为什么你在纪录片里看到的深海一律偏蓝绿——不是导演调的色，是物理。

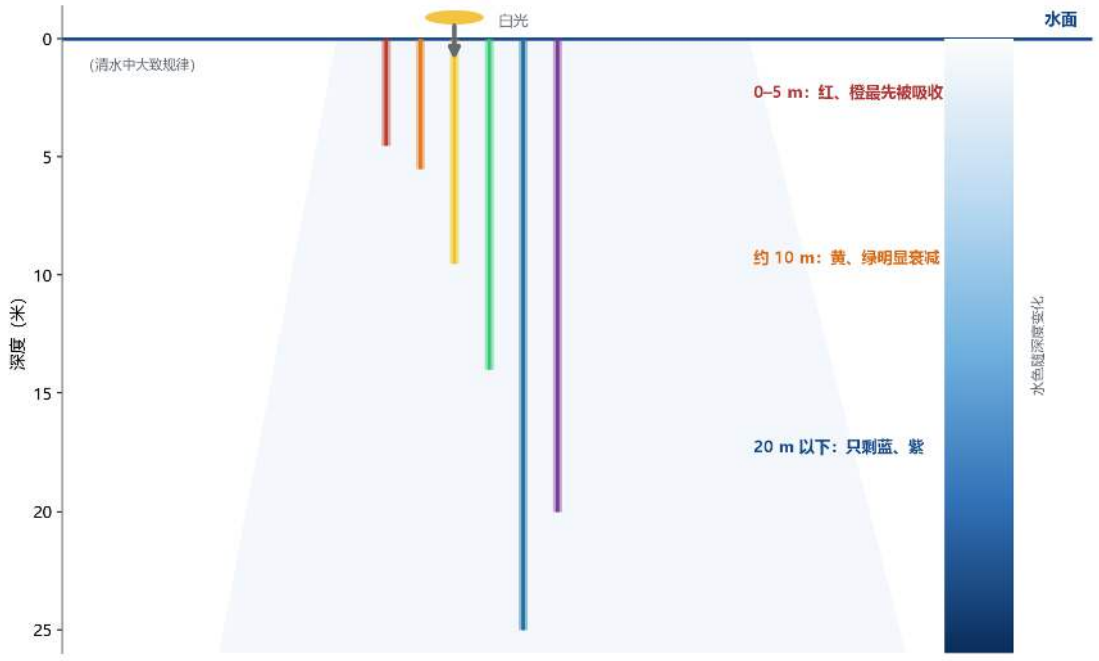



**图 2.6**　水下光衰减：白光入水后各色光按波长先后被吸收，红光“阵亡”得最快，只剩蓝绿能走远——水下照片偏蓝绿的物理根源

对工程来说这带来两个恶果。其一，**对比度崩塌**：ROV上的补光灯只能照亮眼前一小段，稍远的物体淹没在一盆“蓝绿色的汤”里，就像大雾天开车——水面下的“雾”（backscatter，后向散射）还会把灯光原路弹回镜头，形成一层灰蒙蒙的“水下雾”。其二，**颜色不可信**：空气里我们还能靠颜色、纹理辅助判断“这两块是不是同一个东西”，水下大家全是蓝绿色，颜色线索基本作废。

### 2.2.2　第二道坎：立体匹配的前提被破坏

第 2.3 节会讲到，双目测距的核心一步叫**立体匹配**：在左右两幅图里找到“同一个点”。它有一个默认前提——**同一个物体在左右图像里“长得一样”**（灰度、纹理一致）。空气里这个前提大体成立；水下，光照经过两段不同的水路，散射与吸收是随机的：左相机眼前的悬浮颗粒与右相机眼前的不是同一批，两束补光的角度也不同——同一个链环，左图里亮、右图里暗，是常态。“长得一样”的假设被水搅了，匹配就会找错对象，深度就会错。

### 2.2.3　第三道坎：深度网络泛化差

那不用双目，用第 12 讲那种单目深度大模型行不行？也有麻烦：深度网络是在**空气数据**上训练的，水的颜色分布、散射模式它从没见过——把水下照片喂给一个在陆地上百试百灵的深度模型，输出可能完全离谱。这叫**域偏移**（domain shift，训练数据的“世界”和真实使用场景不一样）。解药只有两条：要么用水下数据重新微调（所以我们先要造数据集），要么用对域偏移更鲁棒的基础模型（第 2.4 节的主角）。

### 2.2.4　第四道坎：没有深度真值

最致命的一道。在空气里，想给一张图配“每个像素多远”的真值，可以用激光雷达（毫米级精度）或结构光设备对着标；实验室里还能用机械臂移动相机精确控制基线。水下这些通通失效：光被水吃掉，激光雷达没了距离；浑水里的声呐分辨率又太粗，对不上像素。结果就是：**甲方提供的海量**

**视频，一帧标注都没有**——一个标签都给不了你。空手套标签，就是第 2.4 节要讲的“自动打标签”要创造的奇迹。

**生活化类比：为什么水下照片都偏蓝绿——顺便理解“挑食”的光**

拿一根吸管插进一杯清水，从侧面看，吸管还是原色；换一杯掺了牛奶的水，吸管立刻发灰发蓝——牛奶颗粒把光散射掉了，波长越长的红光“死”得越快。海水介于两者之间：纯吸收（像深水）先吃红光，浑浊（像近岸）再叠加散射。记住这幅图景：**水是光的“红先蓝后”的收费站**，走不远的光就是丢掉的信息。你们处理的真实视频之所以是那种蓝黑色，不是相机坏了。

## 2.3　双目立体视觉：两只眼睛如何“看出”距离

讲完了难，该讲原理了。好消息是：核心原理简单到可以闭眼复现——因为它就长在你脸上。

**知识补齐（重点）：视差——大脑的“测距仪”**

人有两只眼睛，间距约 6.5 厘米，这个间距叫**基线**（baseline）。同一物体发出的光，分别进入左眼和右眼，成像位置有一个小小的错位——这个错位叫**视差**（disparity）。你可以现在就验证：盯着眼前的手指，轮流闭左右眼，手指相对背景会“左右跳”；把手指放远，“跳动”就变小。**近的东西视差大，远的东西视差小**——大脑拿这个错位当测距仪，时刻在算你与世界的距离。双目相机就是把这套生理结构换成两台并排的工业相机：基线已知、焦距已知，测出视差，深度就是一个除法（见下文公式）。

把两台相机水平并排、同时拍一个物体，几何关系如下图：两相机光心相距基线 $b$，物体在深度 $z$ 处，它在左右图像里成像的位置差了 $d$ 个像素——这个 $d$ 就是（像素单位的）视差。

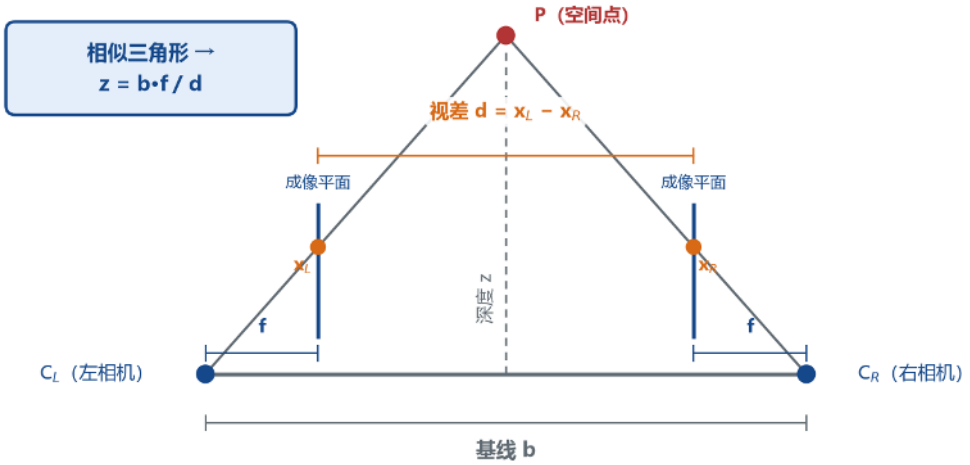



图 2.7　双目成像几何：两台相机光心相距基线 $b$，同一物体在左右图像中的成像位置错开 $d$ 个像素（视差）；由相似三角形即可反解深度 $z$

由相似三角形（下图光线的交叉关系），四个量被一条公式锁死：

$$z = \frac{b\,f}{d}, \tag{2.1}$$

人话：$b$ 是基线（米），$f$ 是焦距（像素），$d$ 是视差（像素），$z$ 是深度（米）。深度等于“基线乘焦距、再除以视差”。分子是相机的“体检参数”（装好就不变），分母是每张图要算出来的东西——**测距问题由此变成找对应点的问题**。

这条公式还藏着一条规律：深度翻倍，视差减半。看下图的反比曲线——**近处视差变化剧烈、远处视差“挤成一团”**。这解释了两件事：双目系统擅长看近处、看不准很远；以及为什么视差差半个像素，近距离的深度能差出几十厘米——**视差必须算得准**。

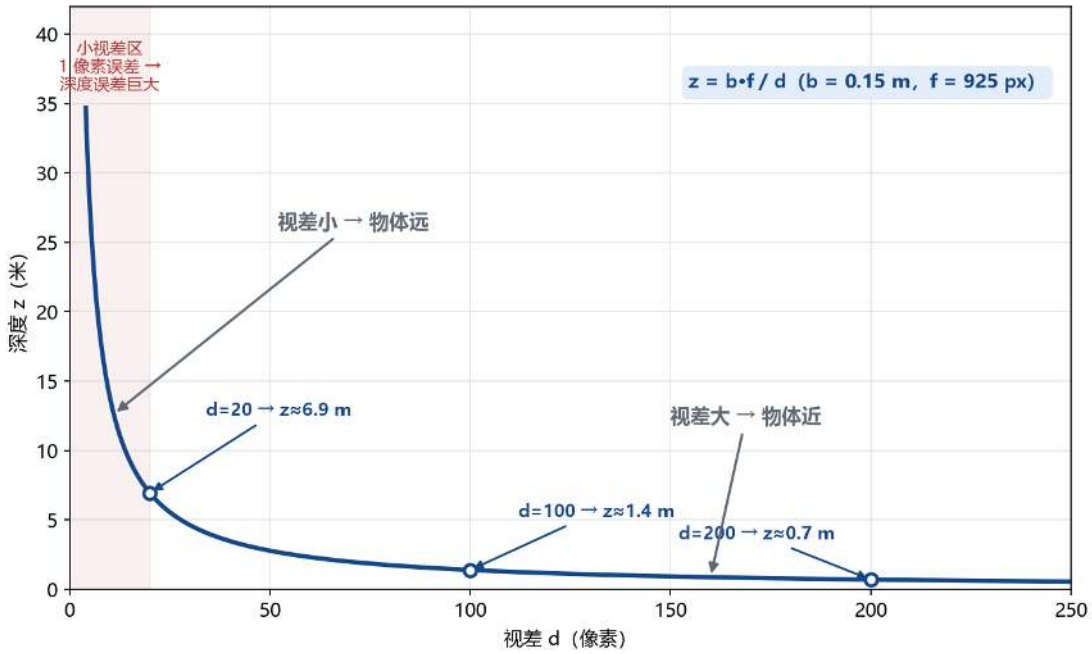



**图 2.8**　视差–深度反比关系：$z = bf/d$ 是一条双曲线，视差越大物体越近；远处视差挤在一起，微小的视差误差被深度放大

代个数找找感觉。本项目后来从视频里自标定出焦距 $f \approx 925.9$ 像素（第 2.5 节讲怎么来的），数据集里视差中位数约 19.2 像素；假设基线 $b = 0.2$ 米，代入式 (2.1)：$z = 0.2 \times 925.9/19.2 \approx 9.6$ 米——链条在镜头前约十米。你看，只要视差算对了，深度几乎是白送的。

**动手时间：一根手指的“跳”——亲手测出自己的视差测距仪**

(1) 伸出一根手指，放在眼前约 30 厘米处，背景对着窗户或书架；轮流闭左眼、闭右眼，交替几次——手指相对背景“跳”了一大步。(2) 把手指伸到一臂远，重复——跳动明显变小。(3) 把手指再放远到几乎看不清，跳动几乎消失。**结论：跳动的幅度就是视差的体感**；近跳大、远跳小，正是 $z = bf/d$ 的反比关系。第 (2) 步里你的大脑正在用“变小的视差”告诉你“手指变远了”——你刚刚人工执行了一遍立体匹配。ROV 拖着的双目相机，就是这台“眼睛机器”的钢铁版。

还差最后一块拼图：**立体校正**（stereo rectification）。真实装到 ROV 上的两台相机，不可能绝对水平对齐——总有一点歪斜。而式 (2.1) 成立的前提是：**同一个点在左右两幅图里位于同一行**（这样才能逐行搜索匹配）。校正做的事，就是把左右两幅图各自旋转、重新投影到“理想水平摆放”的坐标系里，让所有对应点的连线变成水平线（这条线叫**极线**，epipolar line）。一句话记牢：**校正 = 把“找同一个点”从全图搜索简化成“同一行里左右找”**。校正好不好，有个硬指标：真实的对应点偏离同一行的垂直像素数，叫**垂直极线误差**——本项目对它的要求是中位数小于 1 像素，实际做到了 0.254 像素（第 2.5 节的验收表）。




**图 2.9**　双目相机硬件示例：两只镜头并排、基线固定——一台“钢铁版双眼”，水下 ROV 用的正是这类结构

图片来源：Stereolabs ZED 立体相机官方产品图（官网公开资料）

**项目连线：一条公式牵出整个项目**

**本项目：**式 (2.1) 是全项目的“宪法”——组 1 数据组做的一切（去畸变、校正、自标定）都是为了把 $b$、$f$ 弄准、把 $d$ 好算；组 3 训练组训练的网络，本质就是一台“视差计算器 $d$”；组 4 评测组量的是 $d$ 差了几个像素（EPE）。**六目拼接**（第 一 章）：那边六只眼睛围成一圈拼接全景，这边两只眼睛并排测距——相机阵列的两个方向，靠的都是“多看一个视角”。**数据基座**（第 三 章）：形态不同，方法论相通——都是把非结构化素材（视频/文档）加工成结构化、可检索、可训练的数据资产。

## 2.4　没有深度真值怎么办：自动打标签（本章高潮）

现在是全项目最硬核的问题。第 2.2 节说了：水下没有深度真值，甲方一帧标签都给不了。可训练一个深度网络，恰恰需要成千上万张“每个像素多远”的标签。先弄清标签长什么样，再讲没有真值怎么凭空“造”出可信的标签——这一节的方法论，是近几年机器人与三维视觉界最重要的范式转移之一，你学会了可以带走，用在任何“没有标注的领域”。

### 2.4.1　标签本体：一张逐像素视差图

先回答“标签是什么”。本项目里，一张标签就是**一张与原图等大的“灰度图”，每个像素填一个数——该像素的视差 $d$**（单位：像素）。有了它，代入式 (2.1)，深度逐像素白送。所以“给数据打标”＝“为每对左右图像生成一张视差图”。空气里这张图可以用激光雷达扫出来；水下没有，只能**算**出来——算出来的标签不是真值，是“伪”的，所以得名**伪标签**（pseudo-label）。

### 2.4.2　三步流水线：教师打标、多重过滤、学生迭代

自动打标签不是玄学，用一句话说清全部：**大模型教师逐帧打标 ＋ 多源一致性过滤 ＋ 学生自训练迭代**。看下图的闭环：

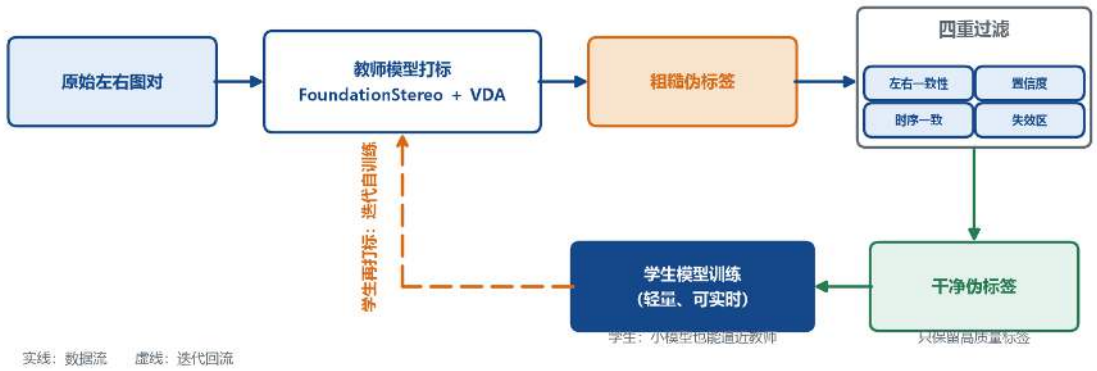



**图 2.10**　教师–学生伪标签闭环：大模型教师逐帧打标，多重过滤筛掉可疑标签，学生模型在幸存标签上训练；学生变强后回头重新打标，数据质量螺旋上升

第一步，**教师打标**。我们请两位“老师”给每一帧打分：**立体教师** FoundationStereo（见下面的前沿模型一览表，给逐像素视差和置信度）+ **视频深度教师** VDA（Video Depth Anything，单目深度大模型的视频版，深度序列在时间上更稳定）。两位老师各批一份作业，正好互相校对。

第二步，**四重过滤**。老师的作业不能照单全收——恰恰相反，**过滤是整条流水线的灵魂**。一条伪标签想活下来，得过四关：

1. **左右一致性**：用左目视差图把左图“搬”到右目，再算一次反向视差，两者应当互相吻合——对不上，说明匹配找错了对象。
2. **置信度**：教师自己会输出“这像素我有多确定”，低于门槛的直接淘汰。
3. **时序一致性**：视频是连续的，相邻帧的深度不该跳变——前后帧对不上号的，可疑。
4. **失效区剔除**：水体背景、无纹理区、OSD字符这些“本来就算不出深度”的区域，整块划掉，不许混进训练集。

第三步，**学生迭代**。幸存的伪标签拿来训练学生模型（学生正是我们要交付的、比教师小的网络）；学生训练好了，回头**重新给数据打标**——学生比原教师更懂本场景，打出的标签更干净，再过滤、再训练……循环几轮，**数据质量与学生水平螺旋上升**。这就是**自训练**（self-training），也称自我蒸馏。

### 2.4.3　最大的先例：NVIDIA 用 140 万对伪标签蒸出快 10 倍的学生

这条路线绝不是我们的发明，而是 2024–2026 年间业界公认的“阳谋”。最有分量的先例来自 NVIDIA：Fast-FoundationStereo（CVPR 2026）用“立体教师 + 单目深度教师 + 开放词表分割（划出天空等失效区）”给海量真实视频打伪标签，一致性校验后拿 **140 万对**伪标签蒸馏学生，得到比原模型**快 10 倍**、精度几乎不掉的实时版。看它的流水线图，与我们的设计思想如出一辙：

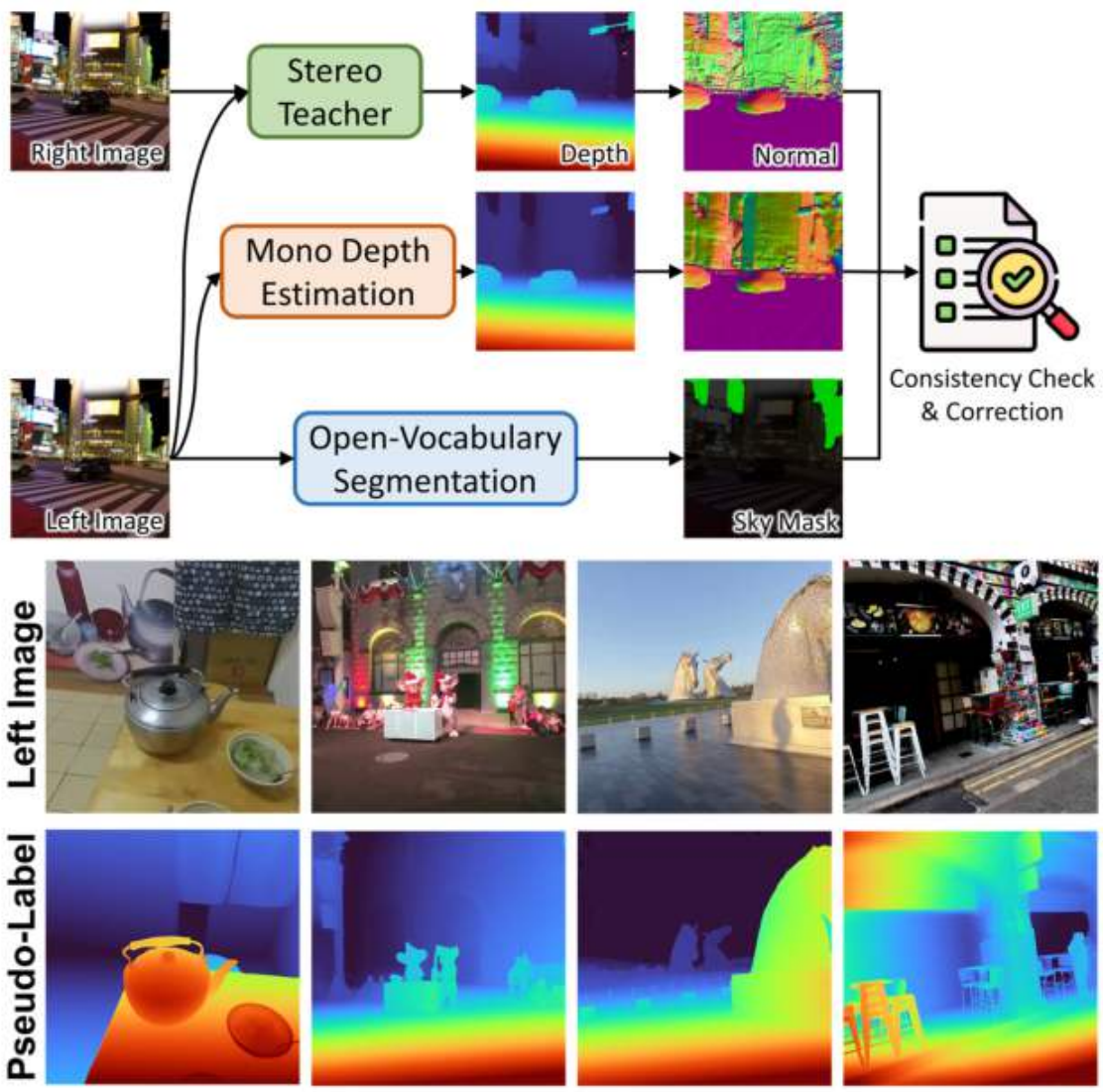



**图 2.11**　Fast-FoundationStereo 的伪标签流水线：左右图像分别经立体教师与单目深度教师打标，开放词表分割划出天空等失效区，一致性校验与修正后成为伪标签（下：多样真实图像与打出的伪标签）

图片来源：NVIDIA Fast-FoundationStereo 论文配图（CVPR 2026）

再往前追，单目深度大模型 Depth Anything V2 已经把“伪标签规模化”玩到极致：先用合成图训练出最大的教师，再用教师给 6,200 万张无标注真实图像打伪深度，最后用这些伪标签训练出各尺寸学生模型——见下图的三段式。**没有真值也能造数据**，它是最有力的证词。

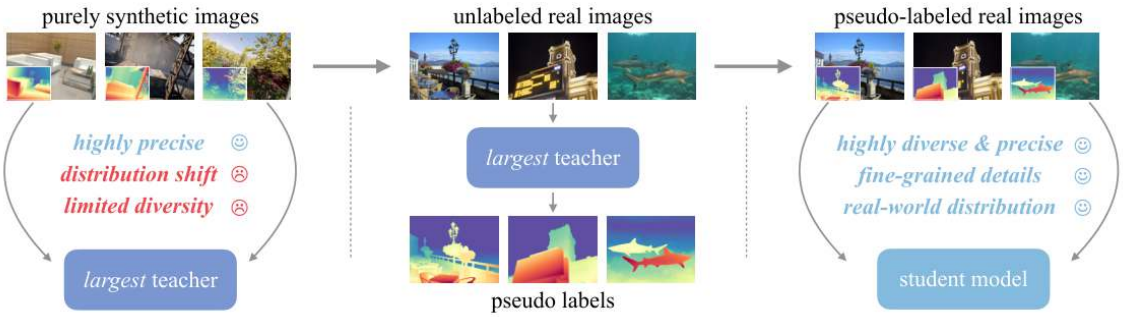



**图 2.12**　Depth Anything V2 的三段式蒸馏：合成图（精确但域窄）训练最大教师；教师给海量无标注真实图像打伪标签；伪标签（多样且精确）训练各尺寸学生模型

图片来源：Depth Anything V2 论文配图（Yang et al., 2024）

### 2.4.4　五条路线与前沿模型

伪标签不是唯一的路。下表把当前五条主流路线摆在一张表里，本项目走的是最后一条：

**表 2.1**　“没有真值”的五条主流路线对比

| 路线 | 代表 | 核心思路 | 短板 |
|---|---|---|---|
| 合成渲染 | UE5 + EasySynth、UW-Stereo | 游戏引擎渲染“自带真值”的水下世界 | 量大管饱，但与真实水下有域差距 |
| 单目伪标签 | Depth Anything V2 | 单目深度大模型给真实帧打伪深度 | 尺度不确定，缺左右一致性约束 |
| 视频伪标签引擎 | Fast-FoundationStereo | 立体教师 + 单目教师 + 分割过滤，规模打标 | 算力门槛高 |
| 单目先验 × 立体 | Stereo Anywhere、MonSter、DEFOM-Stereo | 把单目大模型先验“嫁接”进立体匹配 | 依赖先验的质量与鲁棒性 |
| 自训练迭代（本项目） | AutoLabel-UW 流水线 | 双教师打标 + 四重过滤 + 学生重打标迭代 | 迭代要防“学偏”，过滤阈值需调优 |

这一领域 2024–2026 年的代表作，值得你每个都混个脸熟——它们不只是名词，后续实验里会真的出现在你的命令行里：

**表 2.2**　前沿模型一览（2024–2026）

| 模型 | 年份 | 一句话认识 |
|---|---|---|
| FoundationStereo | 2025 | NVIDIA出品、CVPR 2025 最佳论文提名；当前最强零样本立体匹配基础模型——本项目的主线底座与“立体教师” |
| Depth Anything V2 | 2024 | 用 6,200 万张伪标签单目图像训练的单目深度大模型；单目伪标签路线的代表作 |
| Video Depth Anything | 2025 | 单目深度大模型的“视频版”，深度序列时序稳定——本项目第二位教师（VDA） |
| Fast-FoundationStereo | 2026 | NVIDIA 用 140 万对伪标签蒸馏出快 10 倍的学生；伪标签范式最大先例（CVPR 2026） |
| Stereo Anywhere | 2025 | 单目先验与立体匹配“双分支”融合的代表 |
| MonSter | 2025 | 让单目与立体“互相救赎”：单目补纹理缺失，立体稳尺度 |
| DEFOM-Stereo | 2025 | 把单目基础模型（DINOv2）塞进立体迭代优化的代表 |

最后交代我们自己的主线算法与战绩。毕设生与研究助理在张洪进带领下，把三项传统数学方法——匹配代价泛函分析（DCFA）、模糊图关系（FGR）、置信度后处理（CFP）——改写成**可微模块**，嵌入 FoundationStereo，让经典数学与基础模型各补对方短板（其中 CFP 正对应大图景里的“每个像素有多可信”）。训练与验证在 UWStereo——目前最大的公开合成水下立体数据集（训练 26,612 对 + 测试 2,956 对，约 71 GB）——上微调后，视差误差 EPE 约 1.39 像素。这套工作支撑了一篇在投论文（Image and Vision Computing）与 2026 年青年基金申请。你们四个组的公开数据干跑与对照实验，就是在这套主线的延长线上干活。

**知识补齐：EPE 与 bad-N——视差怎么算“错了几个像素”**

评价视差/深度预测，通篇就两个指标。**EPE**（End-Point Error，端点误差）：预测视差与真值视差逐像素求差的绝对值，再对全图取平均，单位是像素——

$$\mathrm{EPE} = \frac{1}{N}\sum_{i=1}^{N}|\hat{d}_i - d_i|,$$

人话：全图平均“错几个像素”，越小越好（我们微调后约 1.39）。注意第 2.3 节的反比曲线：**视差错 1 个像素，近距离的深度误差会被放大**——所以 1.39 像素不等于“差 1.39 米”，深度误差还要看远近。**bad-N**：视差误差超过 $N$ 像素的像素占全图的比例（常看 bad-1 / bad-2 / bad-4），衡量“错得离谱的部分”有多少。第 12 章评测组的指标白皮书，就围绕这两个数展开。

**前沿速递：水下立体感知 2024–2026 与“无真值域”方法论**

三件事值得记住。其一，**数据集**：UWStereo（最大公开合成水下立体数据集，约 71 GB）已收录为标准测试场；真实侧有 FLSea（海上真实双目）与 SQUID（水下真实立体对）——本项目四个组用这三个公开数据集做扩量与对照，真实数据（500 对）已在资产包里，原始视频依旧按保密红线管理。其二，**零样本能力**：FoundationStereo 这类基础模型在从没见过的场景上直接推理就有可用精度，“零样本”（zero-shot）正把三维视觉从“一个场景训一个模型”推向“一个模型吃遍天下”；但第 2.2 节的四道坎决定了水下仍需微调——这正是你们的活儿。其三，**范式**：“没有真值的领域，用基础模型教师打伪标签 + 一致性过滤 + 学生自训练”已经是机器人界的通用方法论——水下、矿山、仓储、深空，逻辑通用。学会它，你带走的不是一个项目，是一把万能钥匙。

**项目连线：自动打标签，四个组各守一环**

**本项目**：这条流水线把四个组焊在一条链上——组 1 造出能打标的数据，组 2 让教师在真实 GPU 上跑起来，组 3 训练学生并守住“零样本 → 微调”对照，组 4 用 $\tau$（过滤阈值）扫描和误差分桶回答“标签能不能信”；毕设生的 DCFA/FGR/CFP 可微化是学生的“内功”。**六目拼接**（第 一 章）：它的“九个参数头”是“老经验 + 新网络”的混合体，与“传统数学方法可微化嵌入基础模型”是同一个思想的两副面孔。**数据基座**（第 三 章）：那边用大模型给文档打结构化标签、用门控过滤幻觉，这边用大模型给视频打深度标签、用一致性过滤伪标签——“教师打标 + 过滤”跨域通用。

## 2.5　我们的 500 对数据集：一次真实的翻车与自救

讲完方法论，讲我们自己的数据——这一节是真实工程叙事，有翻车、有自救，比成功更值得读。甲方给的是几个硬盘的检查视频（就是本章开头那种带 OSD 遥测的 ROV 画面）。第一步干什么？不是训练，是**给视频做体检**：搞清每段视频“动了没有、拍了什么、能不能用”。

### 2.5.1　翻车：八个“典型场景”全军覆没

最初我们挑了 8 段看起来最典型的视频段，信心满满开始构建。一算指标，傻眼：这些段几乎全是 **ROV 定点悬停段**——ROV 悬停在原地看链条，相机几乎纹丝不动，画面里像素的平均位移（净光流）不到 1 像素。人话翻译：**双目测距靠“从两个位置看同一物体”，相机不动，两个位置就重合成一个**——等于你睁眼闭眼看到的画面毫无差别（回顾动手盒子），基线近似为零，深度在数学上不可解。八个“典型场景”，一个能用立体数据的都没有。教训：**挑数据不能看“画面好看”，要看“几何有用”**——这个反直觉，就是数据组实验一要你亲手体会的。

### 2.5.2　自救：两条硬指标把数据“挖”出来

改进后的筛选是双重的：第一，**活跃度 $\geq$ 2 像素/秒**——ROV 得沿链移动，画面有足够的净运动，立体基线才“长得出来”；第二，**剔除带底部遥测 OSD 的画面**——OSD 是后“盖章”上去的字符，不是真实场景，会污染匹配与训练。两条过一遍，终于从视频里挖出 8 个可用的源窗口，清一色锚链场景：链环特写、海生物、藤壶、卸扣（连接链环的“大销子”）。

**零基础扫盲：OSD——画面上的“盖章”字符**

OSD（On-Screen Display，屏上显示）是设备把遥测数字（深度、坐标、时间戳）直接“印”在视频画面上的功能，常见于行车记录仪与监控。对人是信息，对算法是污染：它随屏幕而动、不随世界而动，网络若把它当真实场景学进去，学到的是“数字水印的排布规律”。所以本项目两条铁律：**筛选时剔除带 OSD 的帧**；**训练时把 OSD 区域划为失效区**。

### 2.5.3　没有标定板的自标定

有了画面，还差相机的“体检参数”：焦距 $f$、畸变系数。水下 ROV 不可能给你架标定板，怎么办？**自标定**：用 `pycolmap` 对视频帧做全局 SfM（Structure from Motion，运动恢复结构——从画面自身的运动反推相机参数与场景结构），解出焦距 $f_x = f_y = 925.9$ 像素，重投影误差 0.62–0.71 像素（人话：反推的参数把所有特征点“放回”画面时，平均偏不到一个像素——参数可信）。

### 2.5.4　一个价值 7 倍的工程决策

更隐蔽的坑在畸变。ROV相机多为广角，画面边缘有明显的镜头畸变（直线被拍弯）。实测发现：**如果在带畸变的原始像素坐标上直接做几何计算，画面旋转会被“揉”进视差里，把视差虚高约 7 倍**——深度直接错 7 倍，全盘皆输。团队因此定下一条决策：**全部几何在去畸变坐标上进行**——先把每帧图像“掰直”到无畸变的理想坐标，再谈校正、再谈视差。这类决策教科书不会写，只有亲手翻过车才会懂。

### 2.5.5　验收：数字说话

最终交付的数据集长这样：目录 `stereo_dataset` 下，每对样本含 `left.png`、`right.png` 与一份 `meta.json` 元数据，另有总账 `manifest.csv`。产出经手写 Bouguet 校正（含 Q 矩阵）共 507 对，组装交付 **500 对校正立体图对**。质量验收按下表执行——每组一个硬指标、一个门槛，全部达标才算交付：

**表 2.3**　stereo_dataset 质量验收单（全部通过）

| 指标 | 数值 | 门槛与说明 |
|---|---|---|
| 可用源窗口 | 8个 | 双重筛选后全部为锚链场景（链环特写/海生物/藤壶/卸扣） |
| 校正立体图对 | 507 对 → 交付 500 对 | 手写 Bouguet 校正 + Q 矩阵 |
| 垂直极线误差（中位数） | 0.254 px | 门槛 $< 1.0$ px，对应第 2.3 节的校正质量 |
| 视差（中位数） | 19.2 px | 场景目标可测深，供 $z = bf/d$ 换算 |
| 匹配内点（中位数） | 492 | 门槛 $\geq 100$，衡量左右图可配对的稳健程度 |
| 自标定焦距 $f_x = f_y$ | 925.9 px | `pycolmap` 全局 SfM，重投影误差 0.62–0.71 px |

看看辛苦换来的数据长什么样。下图是 seg0 窗口的校正左目图像：一根长满海生物（白色斑点多为藤壶壳体）的锚链沿对角线横穿画面，背景是近乎纯色的深蓝海水，右缘还残留少量 OSD 字符——低纹理、弱对比、杂字符，水下立体匹配的“地狱难度”一眼可见：

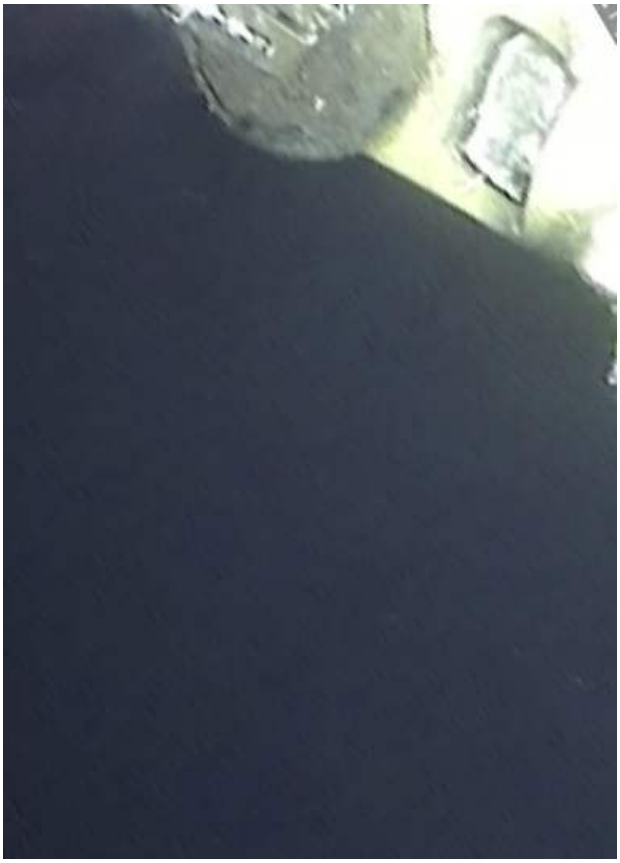


**图 2.13**　校正后的左目图像（seg0 首帧）：长满海生物的锚链沿对角线横穿画面，背景近乎纯色，右缘残留少量 OSD 遥测字符——低纹理、弱对比的典型水下场景

图片来源：本项目 stereo_dataset（E:\锚链视频\stereo_dataset，seg0 样本）

下图是 seg2 窗口：镜头贴近锚链部件拍摄，画面上部可见黄褐色的金属链环/卸扣，大部分区域

沉入阴影——光照不均也是水下常态。两幅图的“校正”效果，就看同一根链条在左右图中是否处于同一行：




**图 2.14**　校正后的左目图像（seg2 首帧）：镜头贴近锚链金属部件（链环/卸扣）拍摄，上部可见黄褐色金属与海生物附着，大面积阴影展示水下光照不均

图片来源：本项目 stereo_dataset（E:\锚链视频\stereo_dataset，seg2 样本）

最后一张，是从原始视频里做“体检”时的抽帧联系表：每 10 秒取一帧、共 36 帧。亮色的锚链在画面中缓缓摆动，OSD 里的深度读数从约 185.5 m 缓慢变到约 187.5 m、离底高度从约 11.8 m 降到约 10 m——运动有了、场景清楚了，“活跃度筛选”挑的就是这样的段：

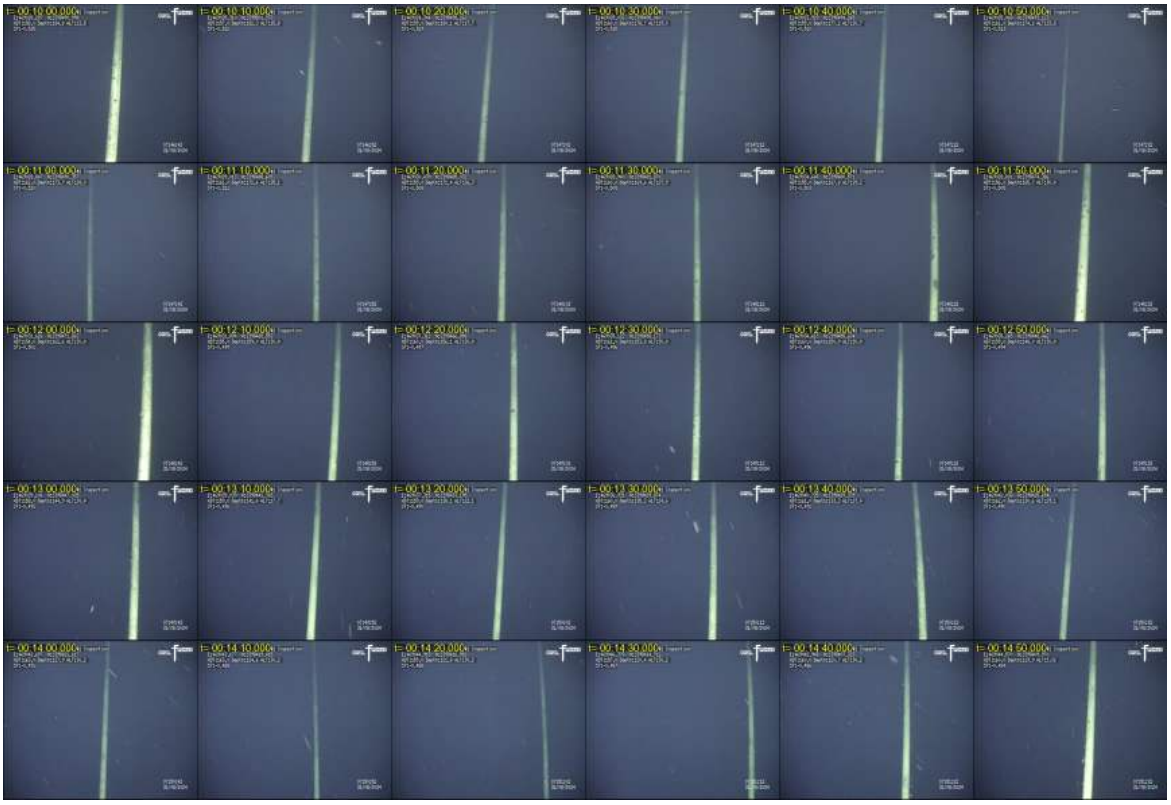


**图 2.15**　检查视频抽帧联系表（每 10 秒一帧，共 36 帧）：锚链亮条在画面中缓缓移动，OSD 遥测中深度约 185.5–187.5 m、离底高度约 10–12 m 缓慢变化——“视频体检”量化的正是这些运动与场景信息

图片来源：本项目分析产物（E:\锚链视频\analysis）

**项目连线：500 对数据从哪来、到哪去**

**本项目**：本节就是组 1 数据组的“前传”——视频体检、抽帧去冗、双目预处理、数据集组装四步，第 9 章给了命令级的实验安排；前期最小实现已把这套流程跑通：500 对真实图对已在资产包里，四个组从第一周就在真实资产上工作；UWStereo/FLSea/SQUID 用于扩量与对照实验。**六目拼接**（第 一 章）：同样吃过“画面好看但几何没用”的亏——全景拼接也要求相机有足够视差，两个项目对“运动”的理解殊途同归。**数据基座**（第 三 章）：manifest.csv 总账、meta.json 元数据、验收门槛表——“先建台账、再谈智能”的数据治理纪律，两边一模一样。

## 2.6　四个组怎么分工：四条并行赛道，实验四会师

方法有了、数据有了，剩下的是组织问题：4个本科生组 + 1 名毕设生，怎么把上面这些活儿分掉？先交个底：前期的最小实现已经完成——FoundationStereo 权重在 4080 上部署可跑、500 对锚链校正立体图对就绪、用 UWStereo 初步微调的基线模型 v0 已产出。所以组织方式不是接力，而是**四条并行赛道：组 1 管粮草、组 2 管炉子、组 3 管炒菜、组 4 管试菜**，共享同一份资产包，实验一至三各自离线自研、谁也不等谁，实验四（12 月 21 日）合并联调。一句话各说清：

**表 2.4**　四个小组的分工与四次实验（第 5–19 周，每周 6–8 小时）

| 组别 | 一句话使命 | 四次实验（一至三离线自研，四联调） | 人工必做 |
|---|---|---|---|
| 组 1 数据组 | 把海量视频变成能训练的数据集 | ① agent 配置 + 视频体检脚本；② ffmpeg / PySceneDetect / OpenCV 抽帧去冗工具链；③小规模微调亲眼看 loss 下降；④数据配比对比 | 抽 30 帧亲眼核对体检指标 |
| 组 2 部署组 | 让大模型在真实 GPU 上跑起来、跑得快 | ① 4080 环境清单；② FoundationStereo 权重部署 + `run_demo.py` + UWStereo 50 对零样本冒烟 + 降级链 vitl → vits → IGEV++；③显存占用曲线与吞吐；④批量/分辨率/fp16 对比 + ONNX 导出初探 | 亲手装机、实录环境坑 |
| 组 3 训练组 | 从零样本基线到 UW-Stereo 微调复现、超参优化 | ①配置说明书；②数据加载冒烟；③复现“零样本 → 微调”对照，画 loss/EPE 曲线；④学习率/增强对比 2–3 组 | epoch 闸门必须经小助手批准 |
| 组 4 评测组 | 给所有模型打分：EPE/bad-N → 误差分桶 → 可视化 → 阈值扫描 | ①指标白皮书 + 手工验证 1 对；②评测全链路；③每模型 $\geq 50$ 对建基线数字表；④伪标签过滤阈值 $\tau$ 扫描 + 分桶归因 | 看误差可视化 $\geq$ 20 张，写“错在哪” |

看懂这张表的关键是**并行不乱套**：四个组开局就拿到同一份资产包（FoundationStereo环境、500 对图对、UWStereo 子集、零样本基线与初步模型 v0），实验一至三各干各的离线自研，产出仍然互为嵌入：组 1 的数据集是组 3 的粮、组 2 的环境是全队的地基、组 4 的指标给组 3/组 1 判卷——但谁也不必等谁先交付。特别点两处“人工闸门”：组 3 的 epoch 推进必须经小助手批准，组 4 必须亲眼看至少 20 张误差可视化并写归因（AI 不可代看）。到了实验四，四条赛道在同一台机器、同一套考卷上会师：新配比接入重训、新权重补测、统一口径总评测干活。特别点两处“人工闸门”：组 3 的 epoch 推进必须经小助手批准（防止过拟合了还傻训）；组 4 必须亲眼看至少 20 张误差可视化并写下“错在哪”（AI 可以算指标，不能替你长眼睛）。

纪律方面，本项目有四条红线（下方红盒），外加一个固定节点：**第 19 周周五（2027 年 1 月 22 日）全员离场，四组合并提交《四组交接总表》**——数据、代码、模型、指标数字表、遗留问题，一项都不能少。每组详细的任务书、命令级步骤与评分表，见第三篇第 9 章至第 12 章。

## 2.7　回头看：你将用到第一篇的哪些本事

这门课把项目编排在第一篇之后不是偶然——本项目的每一步，几乎都能在第一篇找到“那块积木”。下表是对照清单，学习第一篇时请带着右栏的问题去学：

**表 2.5**　第一篇知识在本项目中的落点

| 第一篇 | 讲什么 | 在本项目哪里用上 |
|---|---|---|
| 第 5–6 讲 | 卷积与 CNN 前向/训练 | 立体匹配网络的“眼睛”：特征提取靠卷积；微调就是第 6 讲的训练循环 |
| 第 7 讲 | CNN 演化与基础模型骨干 | FoundationStereo 的骨干源自这条演化线：懂骨干才看得懂结构图 |
| 第 8 讲 | 轻量化、部署与 FCN | 组 2 的日常：fp16、ONNX、显存/吞吐，全是“部署量化”的现场版 |
| 第 9 讲 | 序列与时序建模 | 四重过滤里的“时序一致性”：视频是序列，逐帧独立必丢分 |
| 第 11 讲 | 注意力与 Transformer | 基础模型的骨干与代价聚合都靠注意力：看懂 Q/K/V 才读得懂论文图 |
| 第 12 讲 | 生成对抗与蒸馏 | 教师–学生伪标签闭环就是蒸馏思想的工程化；“教师打标”与 GAN 的博弈思想同理 |

还有一条贯穿始终的“软本事”：**与 AI 协作**. 四个组的实验都在 AI 小助手辅助下进行——但本项目把话说在前面：AI 产出的每一行代码、每一个数字，都必须实跑验证；看图、看曲线、写“错在哪”，必须你亲自来。这不是不信任 AI，而是分清“谁能替你算、谁必须替你看”。

**数据保密红线：四条，一条都不能碰**

本项目使用甲方（中海油）真实生产数据，以下红线写入每个组的任务书，违者按不及格处理：(1) **原始数据不上云**——原始视频、抽出的帧、构建的数据集，**一律不上传任何网盘、不投喂任何在线 AI 服务**，全部处理在本地离线环境完成；(2) **主线代码只读**——毕设生维护的算法主线仓库，本科生组只许读取调用，不许修改，改动一律走自己组的分支并留下说明；(3) **不直接联系甲方**——一切需求与问题经项目负责人张洪进对外沟通，学生不得私自对接；(4) **AI 产出必须实跑验证**——AI 给的代码/结论，跑不通、对不上就不算数，禁止把 AI 的话直接写进报告。红线之外，大胆试错。

本章结束前，把开篇那句话还给你：输入一对左右画面，输出每个像素的深度，并知道每个像素的结果有多可信。现在你知道这句话背后的全部代价——四道物理的坎、一条 $z = bf/d$、一套伪标签流水线、500 对翻车又自救换来的数据、四个组的接力。接下来，去第三篇领你的任务书：数据在第 9 章，算力在第 10 章，训练在第 11 章，评分在第 12 章。海底世界等着你们的 AI 去看懂。

**ROV（Remotely Operated Vehicle，遥控潜水器）**

由母船通过缆绳供电与通信、人在船上遥控的水下机器人；本项目的“摄影师”，负责下水检查锚链。

**FPSO（Floating Production Storage and Offloading，浮式生产储卸油装置）**

海上“炼油厂 + 仓库”：接收海底油井的原油就地加工储存；本项目的甲方场景是“海洋石油 123”号 FPSO。

**锚链（mooring chain）**

把 FPSO/平台拴在海床上的巨型铁链；断裂即大事故，故需 ROV 每年下水巡检——本项目全部数据的拍摄对象。

**OSD（On-Screen Display，屏上显示）**

设备“盖章”在画面上的遥测字符（深度、坐标、时间戳等）；对人是信息，对算法是污染，须剔除或划为失效区。

**视差（disparity）**

同一空间点在左右两幅校正图像中的水平像素差 $d$；近大远小，是深度计算的原料，也是标签本体。

**基线（baseline）**

双目两相机光心之间的距离 $b$；基线为零（相机悬停不动）则深度在数学上不可解。

**立体校正（stereo rectification）**

把左右图像重新投影到“理想水平摆放”的坐标系，使对应点落在同一行，把匹配从全图搜索简化为逐行搜索。

**垂直极线误差（vertical epipolar error）**

校正后真实对应点偏离同一行的垂直像素数；衡量校正质量，本项目验收门槛为中位数 $< 1.0$ px，实测 0.254 px。

**EPE（End-Point Error，端点误差）**

预测视差与真值视差逐像素求差取绝对值再平均，单位像素；本项目 UWStereo 微调后约 1.39。

**bad-N**

视差误差超过 $N$ 像素的像素占全图比例（常看 bad-1/bad-2/bad-4）；衡量“错得离谱”的部分有多少。

**零样本（zero-shot）**

模型不在目标场景数据上做任何训练、直接推理；FoundationStereo 的看家本领，但水下仍需微调。

**伪标签（pseudo-label）**

用模型（教师）的预测当训练标签；不是真值，须经一致性过滤，是“无真值域”的核心数据来源。

**知识蒸馏（knowledge distillation，教师–学生范式）**

让小“学生”模型模仿大“教师”模型的输出；伪标签自训练正是其工程化——FFS 用 140 万对伪标签蒸出快 10 倍的学生。

**UWStereo**

目前最大的公开合成水下立体数据集（训练 26,612 对 + 测试 2,956 对，约 71 GB）；本项目训练对照与干跑的“标准测试场”。

# 项目三　数据基座治理：让海量工程数据开口说话

本章地图

**讲什么**：项目三的“总剧本”——甲方是“海洋石油 123”号 FPSO 相关业务方，手上有 126 份 CSV 共 3.1 亿行运行数据，外加 6 份共 1,001 个物理页的成套工程资料。本章讲我们如何建成“工程资料结构化与本地数据查询系统”（ERAG-ST-20260928 Rev.3）：让机器按结构读懂文档；让你一句话问数、每个数字能点回原件；让 AI 抽取的知识先过门控再入库、再用图谱查关系；以及一场识别虚假内容的攻防战——包括一次把 0.88 改判成 0.74 的诚实审计。**为什么值得学**：前两个项目“造”数据，这个项目“治”数据。它把第一篇的评估、向量表示、训练循环、部署、Agent、对抗思想全部串成一条流水线，还示范了工程里最稀缺的品质：如实。**学完你能**：说清“文档变数据、问题变 SQL、知识过门控、假货现原形”四件事；看懂一张验收缺口表；第 5–19 周带着三个学生组的任务书（第三篇第 13 章至第 15 章）进场。

先设想一个场景。你是“海洋石油 123”号 FPSO（Floating Production Storage and Offloading，浮式生产储卸油装置，第 二 章介绍过的“海上炼油厂”）相关业务团队的一名普通工程师。某天快下班，领导路过你的工位，丢下一句话：“把 2023 年 11 月 18 日 20 点 27 分前后，平台的所有信息整理给我。”就这一句话，你面前立刻立起两座山。

第一座山叫**运行数据**：126 份 CSV 文件、共 3.1 亿行——航行、装载、机械、摇荡，各种传感器一分一秒地记。想找“20 点 27 分”，你得先弄清哪份文件管哪个系统、时间戳是哪个时区、列名是什么意思，然后在 3.1 亿行里大海捞针。第二座山叫**工程资料**：成套的计算书与设备手册，阶段材料就有 6 份、1,001 个 PDF 物理页。领导要的“所有信息”有一半写在纸面里——某台泵的额定排量、某个舱室的设计参数，你得一页一页翻，翻到了再用手抄。

过去这道题的标准答案是三个动作：**人工翻页誊录**（取一张表要照着 PDF 敲进 Excel）、**逐行比对**（核对一个数字要两份资料来回对照）、**人工把关**（就算让 AI 抽取，抽出来的东西入库前还得靠老师傅一条条过目）。一个“20 点 27 分”，足够一个熟练团队加班一整晚。本项目要做的，就是让这道题变成一句话的事。系统有个朴素的名字：**工程资料结构化与本地数据查询系统**（ERAG-ST-20260928 Rev.3）。它由几条能力流水线组成，见下图——工程资料与运行数据从左边进，经过结构化、问数、门控与图检索，最后变成“随口问、秒级答、可追溯”的服务。

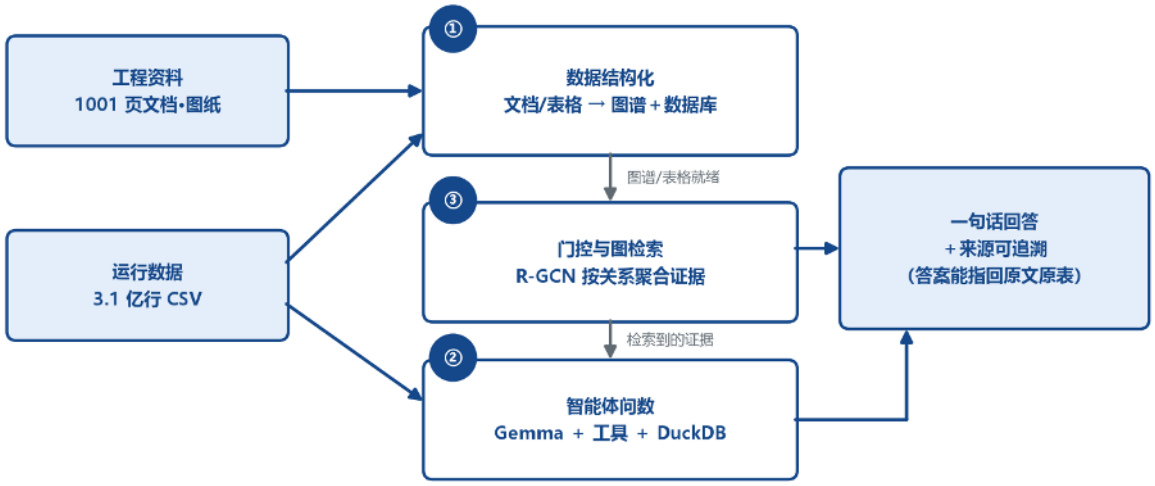



**图 3.1**　数据基座三大能力流水线总览：工程资料与运行数据先经“结构化”变成带锚点的表格、公式与片段，再由“智能体问数”变成一句话可查、每个数字可回溯的查询服务；“知识门控与图检索”为入库知识把一道关、把关系连成网；第四条独立的攻防流水线专门识别输入中的虚假内容——四条链各司其职，层层设卡

系统在技术上有六层：交互层（网页）→ 智能层（本地大模型 Gemma 加语义模型 Granite）→ 工具层（8 个固定工具）→ 计算层（SQLGlot / Wren / DuckDB 三道关卡）→ 数据层（CSV、Parquet 与只读原件）→ 知识层（门控与图检索）。正式环境跑在 8765 端口，测试环境跑在 8766——连端口都分开，演示数据永远混不进生产库。本章按流水线顺序逐层拆解：第 3.1 节先看清问题；第 3.2 节讲结构化；第 3.3 节讲一句话问数；第 3.4 节与第 3.5 节讲知识门控与图检索；第 3.6 节是本章高潮——假内容攻防战；第 3.7 节讲那次著名的“0.88 改判 0.74”审计；第 3.8 节讲三层身份与验收文化；第 3.9 节回头盘点你要用到的第一篇本领。

## 3.1　数据的洪水与资料的迷宫

先把两座山的“地质构造”看清楚，因为它们难的方式完全不同，解法也完全不同。

**运行数据是“洪水”**：量大、规整、没人看。3.1 亿行数据并不是乱码——CSV 嘛，行列分明——问题在于**量**：一个工程师一天最多核对几千行，3.1 亿行意味着几百个人年的工作量。而且这些数据过去“躺着”的姿态很不友好：126 份文件各自为政，列名不统一、时间精度不一，想跨文件问一个问题（“11 月 18 日晚上主机转速和纵摇角有没有联动?”），你得先自己把几份文件拼起来。

**工程资料是“迷宫”**：量不算大（1,001 页），但**结构**极其复杂。计算书里的表格动辄跨页，一列数据跨了页还要继续合并；公式不是图片就是扫描件，想抄进报告只能重新敲；图纸和“图 3-2”这样的标签散落在不同页；更坑的是，PDF 的页码还分“目录里的逻辑页码”和“文件里的物理页”——封面、扉页、目录加起来，逻辑第 12 页可能在物理第 20 页。人在这座迷宫里认路，靠的是记忆和经验；机器过去连“门牌号”都找不到。

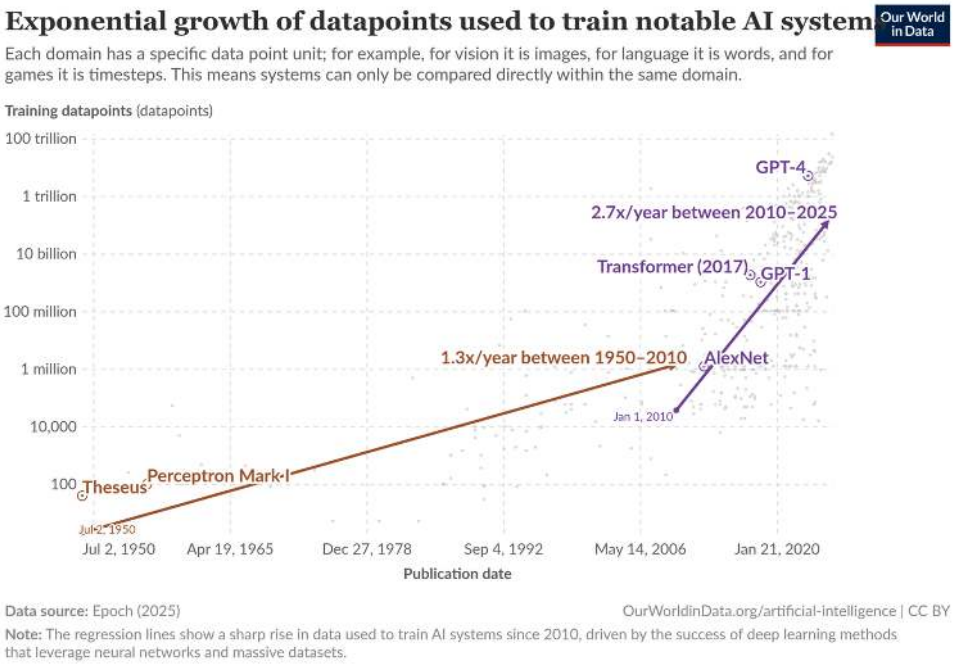



**图 3.2**　数据量增长示意：全球每年产生的数据量呈指数级攀升，工程行业“资料多到没人读得完”是普遍现状——本项目的 3.1 亿行只是这股洪水中的一滴水

图片来源：IDC Global DataSphere 全球数据量统计与预测（公开资料转载）

数据洪水不是海洋石油行业独有的。看上图，全球数据量近乎指数增长，而人类的阅读速度一万年没变过——**“数据多到没人读得完”会是你这一代人在所有行业里的常态**。出路只有一条：让机器替你读。但机器要能读，先得解决三个问题：第一，机器怎么把 PDF 里歪七扭八的表格**按结构**读出来（第 3.2 节）；第二，读出来的数据怎么让人**用一句话**查、查到的数字怎么保证不是模型编的（第 3.3 节）；第三，机器自己“读后感”式的知识产出，怎么防止它夹带私货（第 3.4 节）。

这三个问题对应系统的三类需求编号：**A 类**（查询与追溯）、**B 类**（文档结构恢复）、**C/D 类**（知识门控与图检索）。字母看着枯燥，其实是验收时的“军令状”——每类需求都挂着一串可判定、可复核的验收条款，这与前两个项目“先定指标再动手”的纪律一脉相承。

**项目连线：一个甲方的三张考卷**

三个项目共用同一个甲方背景（中海油海上油气业务），但考卷各不相同：**六目拼接**（第 一 章）处理**图像**，把六路相机画面拼成全景；**水下双目**（第 二 章）处理**视频**，把像素变成深度；**数据基座**（本章）处理**文档与表格**，把页面变成数据库。素材不同，方法论却惊人一致：都先把原始素材加工成“有身份、可追溯”的数据资产（全景图有拼接参数、立体图对有 meta.json、表格有锚点），再在上面谈智能。可以说，本章是你将来在任何一个行业做“AI 落地”时最先碰到的必修课——因为客户手里最不缺的就是文档和表格。

## 3.2　让机器“按结构读懂”：数据结构化

数据结构化对应 **B 类需求：文档结构恢复**。一句话说清目标：**把 1,001 页 PDF/Word 还原成“表格是表格、公式是公式、图是图、每一段都能指回原文”的结构化数据**。注意“按结构”三个字——不是拍张截图让大模型“看图说话”，而是像拆房子一样，把梁、柱、管线一根根登记造册。

先看两类输入的两条路。**PDF 走“真实物理页”**：系统记录的不是目录里的逻辑页码，而是内

容在文件里的物理页面与坐标位置——封面目录造成的页码错位，在这里不起作用。**DOCX 走“原生 XML”**：Word 文件内部本来就是一套结构化标记（XML，一种用标签组织数据的文本格式），系统直接读原生结构，不走“打印成 PDF 再识别”的弯路，表格、标题、脚注自带的信息一点不丢。

读出来之后，四样东西要“保得住”：

- **表格跨行合并保得住**：工程表格最怕跨页——一张表从第 37 页排到第 38 页，列头还得沿用上面的。系统把跨页的行重新拼回一张完整的表，合并单元格的关系也保留下来。
- **公式保存 TeX 源码**：数学公式不存截图，而是抽取成 LaTeX 源码（第 0 章里本讲义自己排版用的那套记号），查询时用 MathJax 在本地渲染——公式可复制、可检索、可再编辑。
- **图片与标签对应**：图纸/照片与它的图号、图题（“图 3-2 系泊布置图”）绑定登记，不会张冠李戴。
- **断句保留双锚点**：长句被切开成片段时，每个片段都带着指回原文位置的锚点，任何一段随时能拼回原文。

**知识补齐：SHA-256——给每份内容发“内容身份证”**

SHA-256（Secure Hash Algorithm 256，安全散列算法 256 位）是一种**指纹函数**：无论输入是一句话还是一部电影，它都算出一个 256 位的“指纹”，内容变一个字节，指纹就面目全非；反过来从指纹推不出原文。本项目用它给每份资料、每张表、每个片段登记**内容身份**。它解决一个细腻的问题：**同名不同内容**。两份文件都叫《主机操作手册.pdf》，一份是 Rev.2 一份是 Rev.3，按文件名登记必然撞车；按 SHA-256 登记，两份各自有各自的身份证，互不冒充。你在第 3.8 节会看到，这枚指纹是“三层身份”的第一层。

“锚点”（anchor）值得单独说，因为它是全系统可追溯性的地基。**锚点就是“出处坐标”**：一串形如“哪份资料、第几个物理页、哪一行、哪一格”的登记信息。19,893 个来源锚点，意味着结构化产物里的每个片段都带着自己的“户口”——后来智能体答的每个数字能“点回原件”，靠的就是它。下图的总体方案里，左列“文档处理”、右列“表格数据”两条链，最终都汇到底部同一句话：“核查结果后返回答案和出处”。

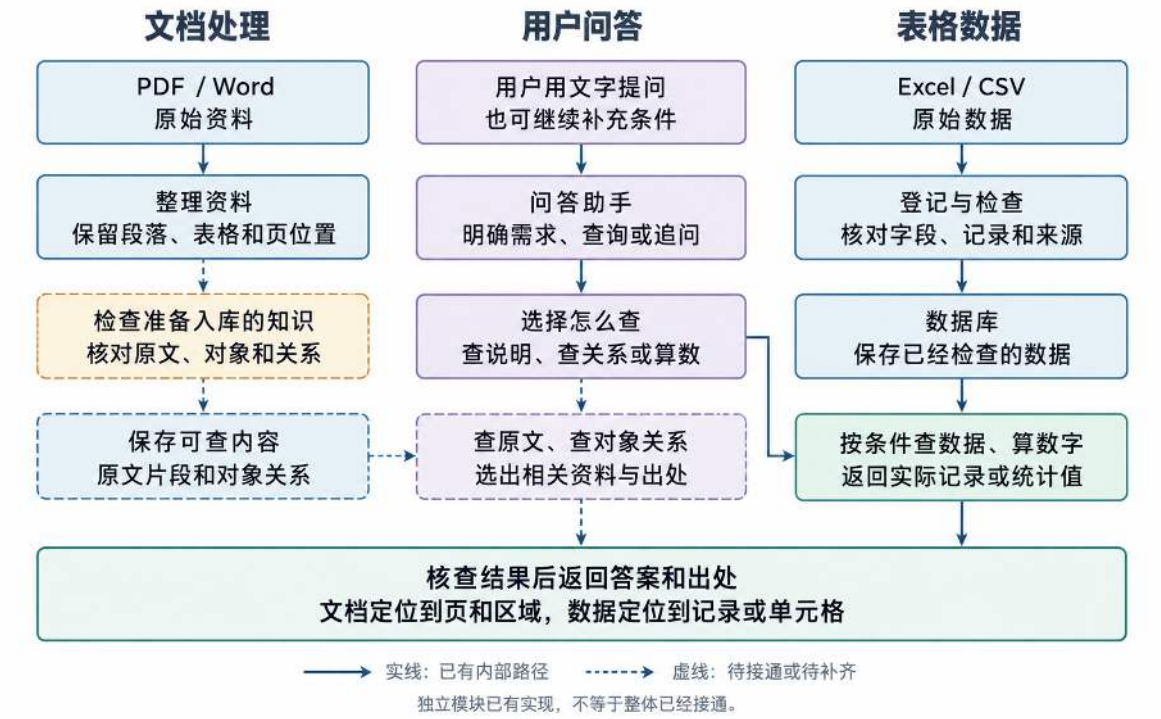



**图 3.3**　总体方案架构图：左列“文档处理”（PDF/Word 原始资料 → 保留段落、表格与页位置的整理 → 入库前的知识检查 → 保存可查内容），右列“表格数据”（Excel/CSV 原始数据 → 登记与检查 → 数据库 → 按条件查数据、算数字），中列“用户问答”把两条链汇到一起；底部是共同要求——核查后返回答案和出处，文档定位到页和区域、数据定位到记录或单元格。注意图例：实线是已有内部路径，虚线是待接通或待补齐——“独立模块已有实现，不等于整体已经接通”

图片来源：本项目《总体方案与验收计划》文档配图

结构化干了这么多活，成果与缺口都必须**拿数字说话**。下表是某一阶段的真实产物清单——注意最后三行，它们和成果一样重要：

**表 3.1**　数据结构化阶段产物与如实登记的缺口（1,001 个源物理页）

| 项目 | 数量 | 说明 |
|---|---|---|
| 恢复表格 | 484张 | 跨页跨行合并，保留合并单元格关系 |
| 数学公式源码片段 | 272 段 | TeX 源码入库，本地 MathJax 渲染 |
| 来源锚点 | 19,893 个 | 每个片段可指回“哪页哪行哪格” |
| 局部回退区域 | 39 处 | 结构过于复杂，自动恢复降级为保守模式，**如实登记待改进** |
| 列宽证据不足的表 | 138 张 | 原始 PDF 无法证实原表列宽，按保守策略恢复 |
| 人工确认计数 | 0 | 全部产物为机器自动生成，**未冒充人工复核成果** |

最后三行就是本项目反复出现的“**如实条款**”：39 个回退区域、138 张存疑的表，不藏、不抹、不改口；更难得的是“人工确认计数 0”——系统大大方方承认这些产物**没有**经过人工逐项确认。听起来像自曝家丑？恰恰相反：把“哪些是机器猜的、哪些是人核过的”分得清清楚楚，下游才敢放心用、也才知道用之前该查什么。这份诚实，第 3.7 节还会回来收利息。

## 3.3　一句话问数：智能体与可追溯查询

数据进了库，怎么查？过去你得会写 SQL（下一节的青盒里 30 秒教会你）——但平台工程师不该被迫学数据库语言。系统给出的答案是**智能体**（agent，能理解意图、调用工具、组织回答的程序）：你用人话问，“2023 年 11 月 18 日 20:27 的尾吃水是多少”，它替你查库、替你算数、替你组织答案。听上去像手机语音助手？往下滑你会发现，这套系统处处是“镣铐”——而**镣铐正是它能被信任的原因**。

先认识两位主角。**大脑**是 Gemma 12B——一个有 120 亿参数的本地大语言模型，用 llama.cpp 在本机 GPU 上量化部署（第 8 讲的部署套路），上下文窗口 8,192 个词元，温度设为 0（不许即兴发挥），并发固定为 1（一次只办一件事）。它负责**理解问题、挑工具、提交 SQL、组织回答**。**检索**由 Granite 语义模型负责：它把文字变成 768 维向量（第 4 讲讲过“向量是语义的坐标”），问题与表名、字段说明、历史案例就在这个向量空间里比相似度。一个管想，一个管找，谁也不越界。

架构的精髓在下图：用户问题交给本地大模型，模型只能**提出**一条 SQL；这条 SQL 先过 **SQLGlot** 检查（是不是只读？字段存在吗？有没有超过资源限制？），再由 **Wren** 做数据模型规划，最后才交给 **DuckDB**——一个单机就能跑的分析型数据库——去执行真正的筛选与聚合。校验不过，错误信息退回模型修改重试；条件不明，系统直接反问你，而不是替你瞎猜一个条件。

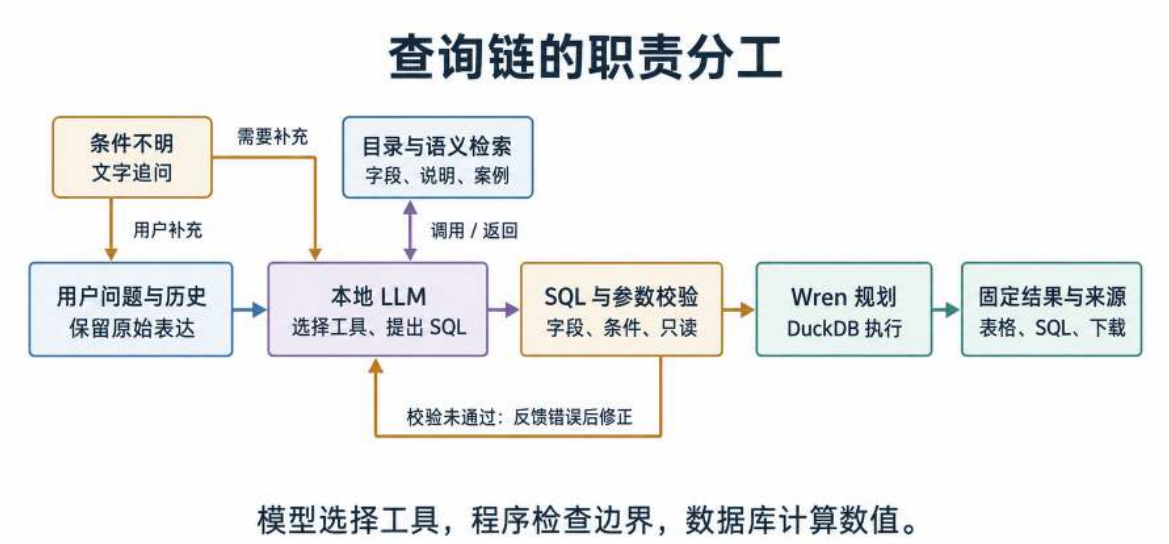



**图 3.4**　查询链的职责分工：本地 LLM 选择工具、提出 SQL，须经“SQL 与参数校验”（字段、条件、只读）通过后，才由 Wren 规划、DuckDB 执行，结果连同表格与 SQL 一起固定下来；条件不明走“文字追问”请用户补充，校验未通过则把错误反馈给模型修正——底部一句口号概括全部哲学：模型选择工具，程序检查边界，数据库计算数值

图片来源：本项目《总体方案与验收计划》文档配图

请把底部那句话读三遍：**模型选择工具，程序检查边界，数据库计算数值**。大模型在这条链里“只提出操作，不直接接触数据”——它碰不到原始 CSV，更碰不到原件；真正碰数据的只有被检查过的 SQL 和数据库引擎。这个设计叫**工具白名单**：智能体手里只有 8 个固定的工具，**没有**任意文件读写，**没有**命令行——模型再“聪明”，也翻不出这八件套的手掌心。这种“思考–行动–观察”的循环正是大模型 Agent 的标准范式（下图），区别在于：多数教学 demo 给 Agent 全权，本项目给 Agent 全副镣铐。

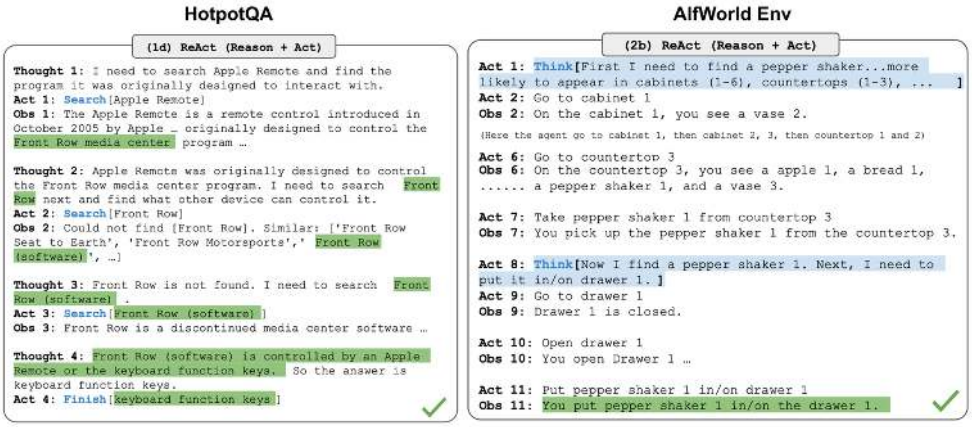



**图 3.5**　LLM Agent 循环示例：模型“思考”决定下一步行动，调用工具获得“观察”结果，再基于观察继续思考——循环直到给出最终回答；本项目的智能体就是这个循环，但每一步“行动”都被限制在 8 个白名单工具之内

图片来源：公开技术资料中的 ReAct 模式 Agent 循环示意（Yao et al., 2022）

**零基础扫盲：SQL 是什么——数据库界的“填空题”**

SQL（Structured Query Language，结构化查询语言）是跟数据库对话的官方语言。它的可读性好到惊人，比如“查 11 月 18 日 20 点 27 分的吃水”翻译成 SQL 大概是：`SELECT 尾吃水 FROM 运行数据 WHERE 时间 = '2023-11-18 20:27:00'`——SELECT 是“挑哪几列”，FROM 是“从哪张表”，WHERE 是“满足什么条件”。数据库收到后逐行筛出答案。它最大的好处是**没有歧义**：语句写定了，结果就定死了——这正是本章让大模型“只写 SQL、不算数字”的原因：人话有歧义，SQL 没有。

智能体对外承诺的能力，系统文档里编号 A1–A7。逐条翻成人话：

**表 3.2**　A 类查询与追溯的七项能力承诺

| 编号 | 名目 | 人话版 |
|---|---|---|
| A1 | 一句话问数 | 直接说人话就能查，不用学 SQL |
| A2 | 时间精度到秒 | “20:27”就是 20:27，不会给你“20 点档平均值”这种和稀泥的答案 |
| A3 | 缺条件会追问 | 条件不足时反问你，而不是替你瞎猜一个答案 |
| A4 | 数量与边界由程序保证 | 说“前五条”就只给 5 条；默认最多返回 100 行——截断是程序干的，不是模型嘴上说的 |
| A5 | 数字由数据库算 | 所有求和、平均、计数都是 SQL 聚合的结果，模型只负责转述 |
| A6 | 每个数字能点回原件 | 答案带锚点，可以一路点回 PDF 的哪页哪行哪格 |
| A7 | 结果固定不串题 | 温度 0、并发 1：同一个问题今天问明天问，答案一致 |

A4 值得多说一句，它体现了一种**边界控制哲学**。“给我看前五条”这种话，若交给大模型自由发挥，它可能给你 6 条、4 条，甚至编 5 条——语言是模糊的，模型顺着模糊走。本项目的做法是：模型把“前五条”翻译成 SQL 里的 `LIMIT 5`，**由数据库引擎机械地执行截断**——说前五条就只给五条，一条不多，一条不少。把“模糊的语言承诺”变成“确定的程序行为”，是让用户敢信一个 AI 系统的第一块基石。

支撑问数的数据层也值得一窥。126 份 CSV 并不是简单倒进数据库了事：每份先**登记核验**（每份文件都要被完整地读两轮，通过核验才准入库），然后**按月分区**转成 Parquet 列式存储（按列组织

文件的格式，只查几列数据时不必读全表，快得多），最后在 DuckDB 里建成四个**查询模型**——把杂乱的原始表整理成四个“官方视图”：

**表 3.3**　四个查询模型：智能体能问什么，先由数据层划定

| 查询模型 | 列数 | 覆盖内容 |
|---|---|---|
| `fpso_navigation` | 29 | 航行相关：位置、航速、航向等 |
| `fpso_load_status` | 30 | 装载相关：各舱液位、吃水、卸载等 |
| `fpso_machinery` | 20 | 机械相关：主机、泵、发电机状态等 |
| `fpso_motion` | 27 | 运动姿态：纵摇、横摇、升沉等 |

能力不是吹出来的，是考出来的。项目组独立编写了 500 道题的回归考卷（“回归”指每改一版系统就重考一遍，防止改好一处坏三处），六种题型成绩如下：

**表 3.4**　智能体问数 500 题历史回归成绩（独立标准答案判卷）

| 题型 | 得分 | 说明 |
|---|---|---|
| 明细查询 | 219/220 | 查具体某时某刻某值；唯一失分题如实留档 |
| 普通聚合 | 100/100 | 求和、平均、计数、最值 |
| 连续对话 | 70/70 | 多轮追问，指代与上下文不丢 |
| 模糊歧义 | 60/60 | “最近”“那次”等模糊说法，按 A3 追问或按约定消解 |
| 执行边界 | 28/30 | LIMIT 截断、100 行上限等边界行为 |
| 复杂挑战 | 20/20 | 多表关联、嵌套条件 |

合计 497/500。速度上，60 道实测题的中位数耗时 4.93 秒——从“一个团队加班一晚”到“五秒钟一句话”，这就是开篇那道题的答案。还要强调一句验收纪律：这 500 道题的**标准答案是独立编写、先行复审**的，系统自己生成的 SQL 不许反过来当标准答案——否则等于让学生自己批自己的卷子。这句话在验收文化里还有更响亮的版本，第 3.8 节见。

## 3.4　知识门控：AI 抽的知识入库前先过检查

结构化和问数解决的是“资料里的东西怎么查”。但系统还有更进一步的野心：把资料里的**知识**（“P-101 泵的额定排量是 120 立方米每小时”这类事实条目）抽取出来、存进知识库，供检索与推理复用。这就引入了新的风险：**抽取靠大模型，而大模型会幻觉**——它会无中生有、会张冠李戴、会把“某工况下”悄悄放大成“任何工况下”。于是系统设了第三道关卡：**知识门控**（gating，入库前的闸门）。一句话概括其哲学：**机器抽的知识只是“候选”，过得了检查才配叫“知识”**。

门控流水线五步：**读取 → Chunk → 提取候选知识 → 执行门控 → 发布**。Chunk（分块）是把长文档切成带锚点的片段；提取步骤让模型从片段里抽出“主体–关系–客体”式的候选条目；然后是主角登场——每条候选都要过**三类检查**：

1. **是否支持**：候选条目说的内容，原文证据真的支持吗？还是模型脑补的？
2. **是否同一对象**：条目里的设备、系统，和证据里的是同一个吗？——三个平台都可能有一台叫“P-101”的泵。

3. **关系是否成立**：抽出的关系（“额定排量是”）与证据里的关系（“设计工况下排量可达”）是同一回事吗？

检查之后三个出口：**通过**（发布入库）、**转复核**（证据不够干净，交人工或专家裁定）、**拒绝**（打回并留下记录）。看下图的三岔口：

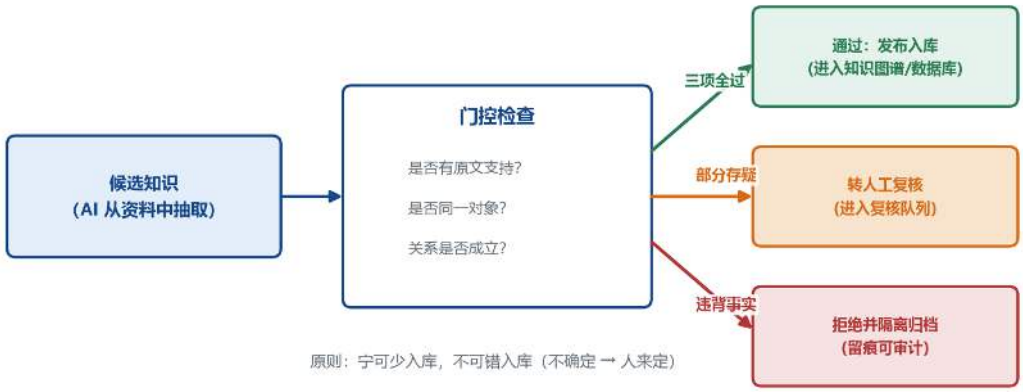



**图 3.6**　知识门控三出口：候选知识过三道检查（原文是否支持、是否同一对象、关系是否成立），按结论分流——通过（发布入库，带完整证据链）、转复核（证据存疑，交人工裁定）、拒绝（打回并留痕）；三个出口都有登记，没有一条候选能“悄悄消失”

光看流程图不刺激，看一个真实的教学例子。系统文档里有一张“P-101 表”，把泵 P-101 的五种典型情形和门控的五种处置一一对应——这张表是给新工程师的第一课，我们原样搬给你：

**表 3.5**　P-101 教学表：五种情形，五种处置（门控逻辑的人话版）

| 情形 | 候选知识 vs 原文证据 | 门控处置 |
|---|---|---|
| 内容一致 | 两条来源都写“P-101 额定排量 120 $m^3/h$” | **通过**：发布入库 |
| 数值冲突 | 手册写 120，计算书写 150 | **拒绝**：证据打架，不许带病入库 |
| 身份不清 | 证据讲的是 3 号平台的 P-101，条目没说是哪个平台 | **转复核**：交人确认对象 |
| 缺少证据 | 条目言之凿凿，却找不到原文出处 | **拒绝**：孤证不立 |
| 扩大含义 | 原文说“设计工况下可达 120”，条目抽成“任何工况下 120” | **拒绝**：偷换条件，性质是歪曲 |

这套逻辑跑在真实数据上是什么样？某一阶段的运行数字：候选知识共 **9,505** 条，门控判下来——**接纳 2,988** 条、**待复核 582** 条、**拒绝 5,935** 条。看清楚：**六成以上的候选被拒之门外**。这不是模型太笨，恰恰是门控在工作——抽取模型天生“手松”，门控负责“手紧”。2,988 条通过者也不是直接上生产：先在**沙箱**（sandbox，与正式环境隔离的试运行区）分 20 批发布，每批观察无异常才转正。事后审计环节，**9,533 条事件链被完整重算一遍，结论一致**——每一条“为什么通过、为什么拒绝”都有账可查。对比一下老办法“AI 抽完人工把关”：人把关的瓶颈是人只有一双眼睛；门控把关的瓶颈是规则是否严谨——而规则，是可以审计、可以改进的。

## 3.5　图谱查关系：知识图谱与图检索

门控入库的知识是一条条孤立的事实。但工程师的真问题往往是**关系题**：“3号平台 P-101 的备泵是哪台？它的润滑系统和哪台主机共用？”——答案散在几十份资料里，靠关键词检索永远拼不齐。这就轮到**知识图谱**上场了。

**知识补齐：实体、关系、三元组——知识图谱的三块积木**

**实体**（entity）是世界里的“名词”：设备、系统、平台、文档；**关系**（relation）是名词之间的“动词”：“3 号平台 –拥有– P-101”、“P-101 –备泵是– P-102”。一条“主体–关系–客体”叫一个**三元组**（triple），把成千上万条三元组连起来，就是一张**知识图谱**（knowledge graph）：节点是实体，边是关系。它和向量检索的分工是：向量擅长“找相似”（这段话和那段话意思近），图谱擅长“查关系”（A 和 B 之间隔了几跳、走的什么关系）——你问“意义相近的段落”，用向量；你问“三亲等以内的亲戚”，用图谱。

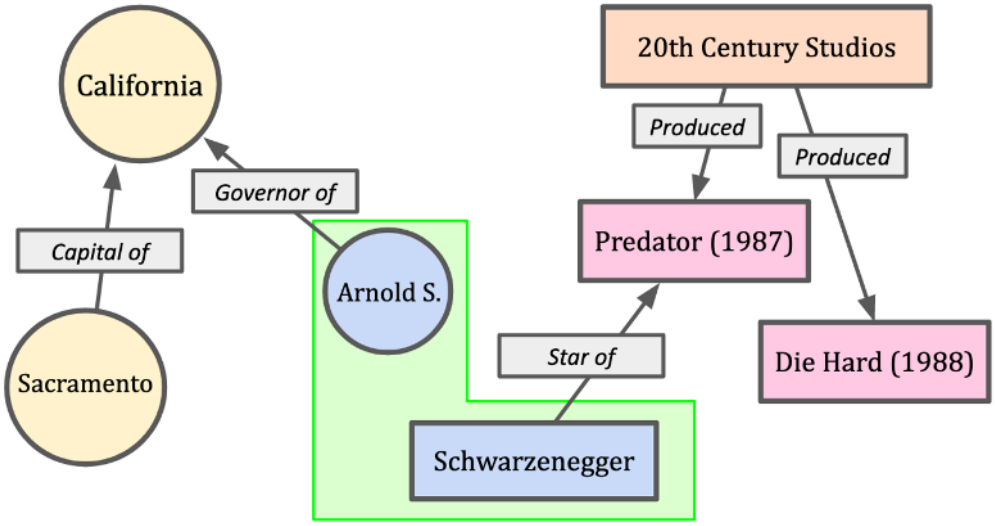



**图 3.7**　知识图谱可视化示例：节点是实体（设备、系统、文档），连线是关系；顺着边走就能回答“A 和 B 之间有什么联系”这类关键词检索答不了的题

图片来源：Wikimedia Commons，知识图谱网络可视化示例（CC 授权转载）

检索的具体打法叫 **GraphRAG**（Graph-based Retrieval-Augmented Generation，基于图的检索增强生成）：回答问题前，先沿着图谱把**相关的实体与关系当作证据**找出来，再把证据交给生成模型组织答案——比普通 RAG（只检索文本片段）多了“关系”这味药。而给证据**排序**的模型叫 **R-GCN**（Relational Graph Convolutional Network，关系图卷积网络），它的直觉看下图：一个节点该“信谁”，取决于它沿**不同类型的关系**连着谁——设备类关系连来的证据，和文档类关系连来的证据，权重应当不同。R-GCN 给每类关系各配一套聚合参数，本系统叠了三层，并把 BGE 模型输出的 1,024 维实体向量投影到 256 维排序空间——人话：把“语义相似”的向量，微调成“关系上也站得住”的排序。

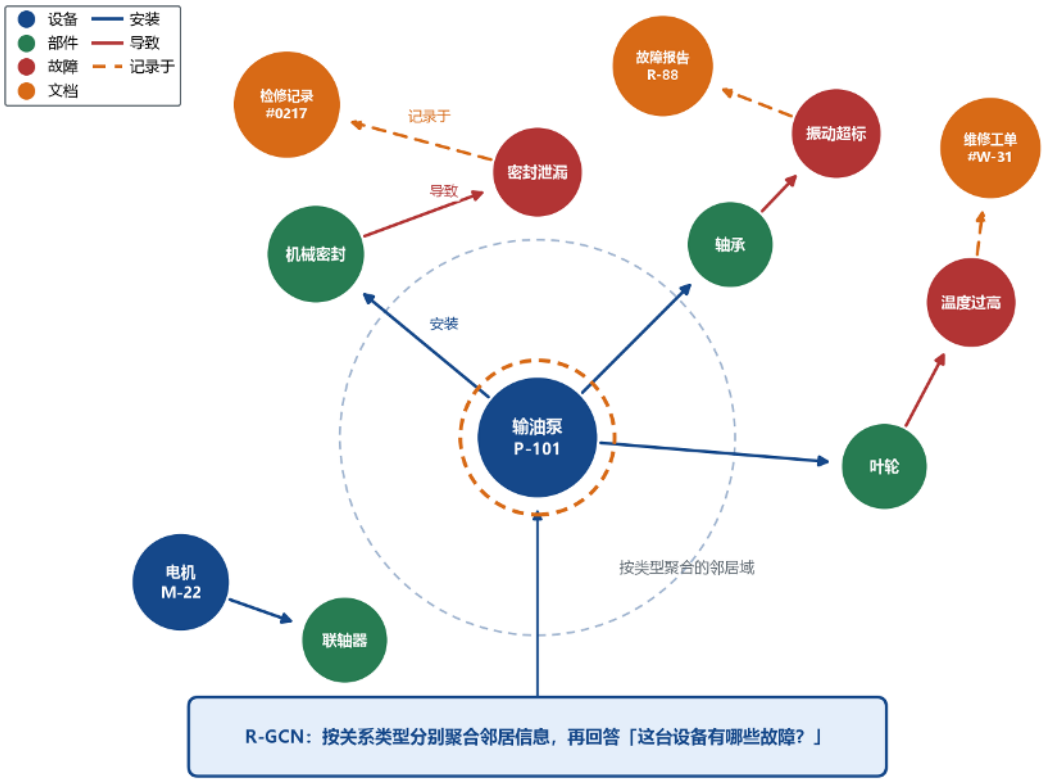



图 3.8　R-GCN 图检索示意：以查询抽中的实体为种子，沿知识图谱中不同类型的关系（不同颜色的边）逐层聚合邻居信息；三层之后每个节点都“见过”三跳以内的邻居，再投影到排序空间为候选证据打分——不同关系类型各有各的聚合权重，“谁说的话更可信”由结构说了算

在线检索的流程是一套严格的“节流阀”：一次提问最多取 **8 个种子实体** → 沿真实关系向外扩 **3 跳** → 裁剪到 **128 个候选** → R-GCN 聚合排序，选出最相关的证据交给生成端。为什么处处设上限？和 A4 的边界哲学同源：**检索范围必须是确定的、可审计的**——不设上限的图检索，一遇密集节点就会把无关证据卷进来，既慢又会“污染”答案。8/3/128 三个数字写在系统文档里，每次检索都照此执行，复盘时能精确说清“当时看了哪些证据”。

## 3.6　假内容攻防战（本章高潮）

门控很能干，但它有盲区。回看 P-101 表：门控的五连处置，管的全是**证据层面**的问题——没出处、证据打架、对象不清。可如果一条假知识**带着貌似齐全的证据、写着貌似合理的数值、语言毫无破绽**呢？随着 AI 生成内容渗入工程文档，这不再是抬杠：一份被 AI“润色”过的报告完全可能把 120 写成 135、把“试验值”写成“额定值”，语言通顺、格式规范、还带着参考文献式的装饰。**语义词造假**——无数值格式错误、无逻辑硬伤、纯粹在事实层面撒谎——是门控规则抓不住的。系统的回应，是一个独立的对抗训练 demo（接入 DeepSeek 云端接口，模型 `deepseek-flash`，即 DeepSeek-V4.1-Flash，使用时须关闭思考模式）：**让大模型扮演造假者，日夜不停地生产假条目，把防御系统练出“火眼金睛”**。

先立铁律，再谈攻防。本项目两条铁律，值得抄在扉页上：

- **“大模型只能改说法，永远改不了事实——目标假数值由程序给定。”**造假者拿到的假数值是程序写死的，大模型的任务只是把假数值“包装”得像真的；它无权自己发明数字。没有这条，造假者自己幻觉出来的“假货”会污染整个训练场。

- **“LLM 只做增强，离线逻辑永远是兜底。”**云端模型断网、涨价、下线，离线流水线照样出题、照样判卷——第 3.7 节的四跑对照表会给这条铁律一个漂亮的注脚。

整个攻防闭环由四个角色组成，看下图与角色表：

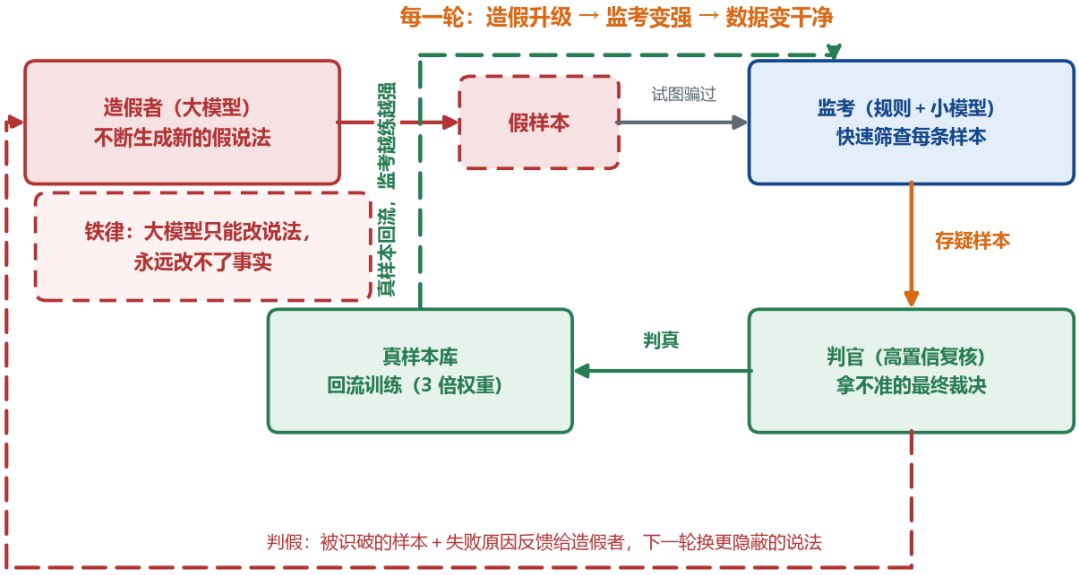



**图 3.9**　真假内容攻防闭环：造假者（大模型 + 程序给定假数值）批量生产以假乱真的条目；监考对不确定样本追问、对放行样本抽检；判官只认数值与单位等硬证据定生死；库存管家看管家底、扫描“中毒”样本；复核结论沉淀为蒸馏样本与先例库回流——四方在同一张考卷上螺旋进化

**表 3.6**　攻防四角色：人设与关键数字

| 角色 | 人设 | 关键规则与数字 |
|---|---|---|
| 造假者 | 不写出题范围的老千选手 | 每轮 30 条直接造假 + 10 条语言造假 + 10 条纯文字造假，最多 5 条润色；骗过的题目以 few-shot 注入下一轮“进化”，每 4 轮强制策略变异 |
| 监考 | 不确定必问的场务 | 不确定必追问，每轮不超过 25 条；纯文字样本走三分类复核；放行样本抽检 10% |
| 判官 | 只认硬证据的主考 | 存疑样本复核每轮不超过 25 条；判“真”须置信度 $\geq 0.75$ 且有数值/单位等硬证据，光“读着像真的”不算数 |
| 库存管家 | 看家底的仓库主任 | 管蒸馏库（2,654/5,000）、监考先例库（6,891/8,000）、中毒隔离（72 条）；支持缓存报告、中毒扫描、一键回滚快照 |

进化机制是这场军备竞赛的灵魂。造假者每轮产出的题目，凡骗过防御的，会被作为 few-shot 示例“喂”回它自己下一轮的提示词——“照这个水平继续造假”；每 4 轮还强制策略变异，防止它在一种套路上一路走到黑。防御方则把每次复核的结论沉淀成两种资产：**蒸馏样本**（以 3 倍权重回流训练）和**监考先例库**（“上次遇到这种话术是这么判的”）。于是你攻我守、你守我攻，双方水平螺旋上升——这正是第 12 讲 GAN“生成器与判别器互相逼强”思想的工程化落地，只是博弈的赌注从“以假乱真的图像”换成了“以假乱真的工程数据”。

考卷本身也一丝不苟：3,300 道题的封存考卷中含 300 道 **OOD**（out-of-distribution，分布外——出题时故意换上训练时没见过的造假手法，专测“遇上新骗术会不会抓瞎”）。造假手法登记了 **26 个**

**造假家族**（G1–G19 编号，其中 PURE_TEXT 纯文字家族下分 7 个亚型），并按“易、中、难”三层分账——易档改个数字，难档通篇语义撒谎、只字不差。下面两张图是这场攻防战的驾驶舱：

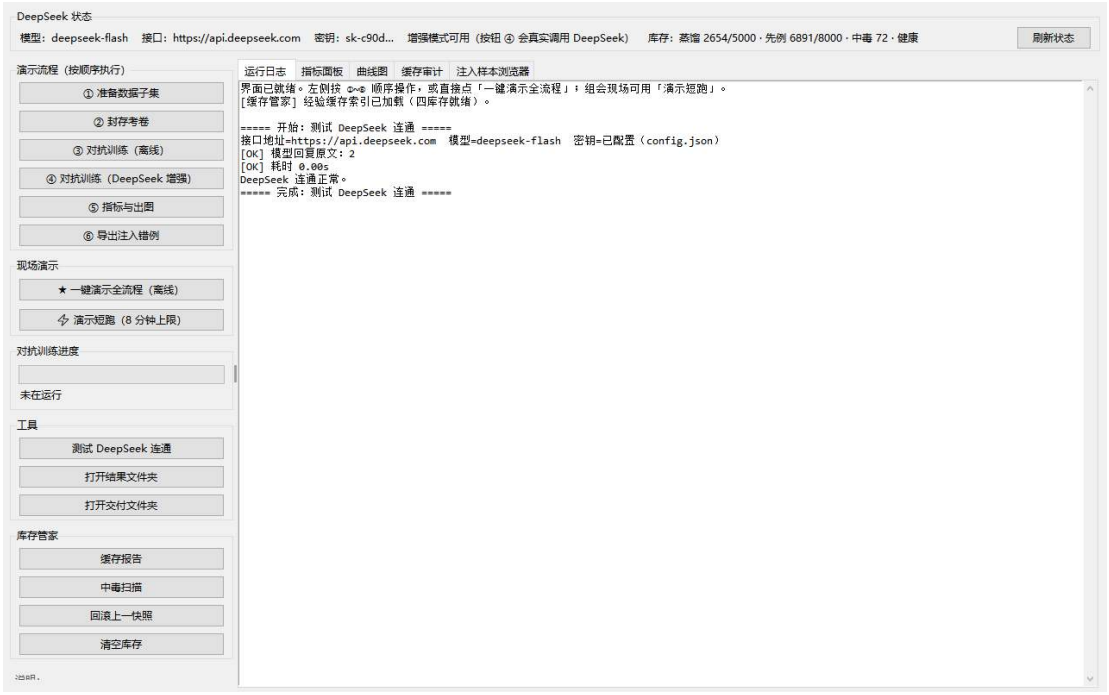



**图 3.10**　对抗训练 demo 主界面：左侧自上而下是六步演示流程（①准备数据子集 → ②封存考卷 → ③对抗训练（离线）→ ④对抗训练（DeepSeek 增强）→ ⑤指标与出图 → ⑥导出注入错例）、现场演示按钮（一键演示全流程/8 分钟演示短跑）、库存管家维护按钮（缓存报告/中毒扫描/回滚上一快照/清空库存）；顶部状态栏常驻库存水位（蒸馏 2,654/5,000、监考先例 6,891/8,000、中毒隔离 72）；右侧是多页签日志区，图示为一次 DeepSeek 连通性测试（模型回复 OK，用时 0.06 秒）

图片来源：本项目对抗训练 demo 截图（E:\中海油数据，demo_ 对抗训练 results）




**图 3.11**　对抗训练 demo 的指标面板（离线模式最近一次结果）：宏平均 F1 0.742（95% 置信区间 0.728–0.756）、综合区分度 0.937、监考抓包率 0.947、金标字段全量 13,475 条零缺失、交付 2,000 条分层齐全（易 300/中 400/难 300），下方是 16 项自检逐条“通过”的清单——每个成绩旁边都挂着可核查的证据，而不是一句“效果良好”

图片来源：本项目对抗训练 demo 截图（E:\中海油数据，demo_ 对抗训练 results）

攻防的最终交付物是一份 **2,000 条真假混装数据集**：假条目 1,000 条（易 300、中 400、难 300）加真条目 1,000 条（五个真话亚族），每条带双金标——`gold_label`（客观真假）与 `disc_label`（判官当时的判定口径）。往后任何人想检验“我的模型能不能识别工程数据造假”，拿这份数据集一测便知，不必从零再造一场攻防。

**动手时间：当一回监考——三道题，请人工判定**

下面 3 条候选知识，先别看答案，和同桌各自判定“通过/拒绝/转复核”，再对照第 3.4 节的 P-101 表逻辑对账：(1) “P-101 泵额定排量 120 $m^3/h$（摘自《设备操作手册》物理页 45，表 3-2）”；(2) “P-101 泵额定排量 135 $m^3/h$”；(3) “P-101 泵在冬季工况下排量可提升约 20%”。**参考判定**：(1) 出处具体、数值有据——通过；(2) 无出处，且数值与手册冲突——拒绝（数值冲突）；(3) 没说是哪个平台的 P-101（身份不清），“约 20%”也无从核实（证据不足）——转复核。判对了几条？如果你在 (3) 上犹豫了，你就亲身理解了为什么门控需要“转复核”这个出口、监考为什么“不确定必问”——**犹豫本身不是 bug，把犹豫假装成确定才是。**

## 3.7　0.88 的谎言与 0.74 的诚实

这一节讲本项目的“名场面”。先看下面这条曲线，它是 2026 年 9 月中旬以前系统的“官方战绩”：75 轮对抗，判官在封存考卷上的 macro-F1 始终稳稳停在 0.87–0.88，一条漂亮得像用尺子画出来的水平线。

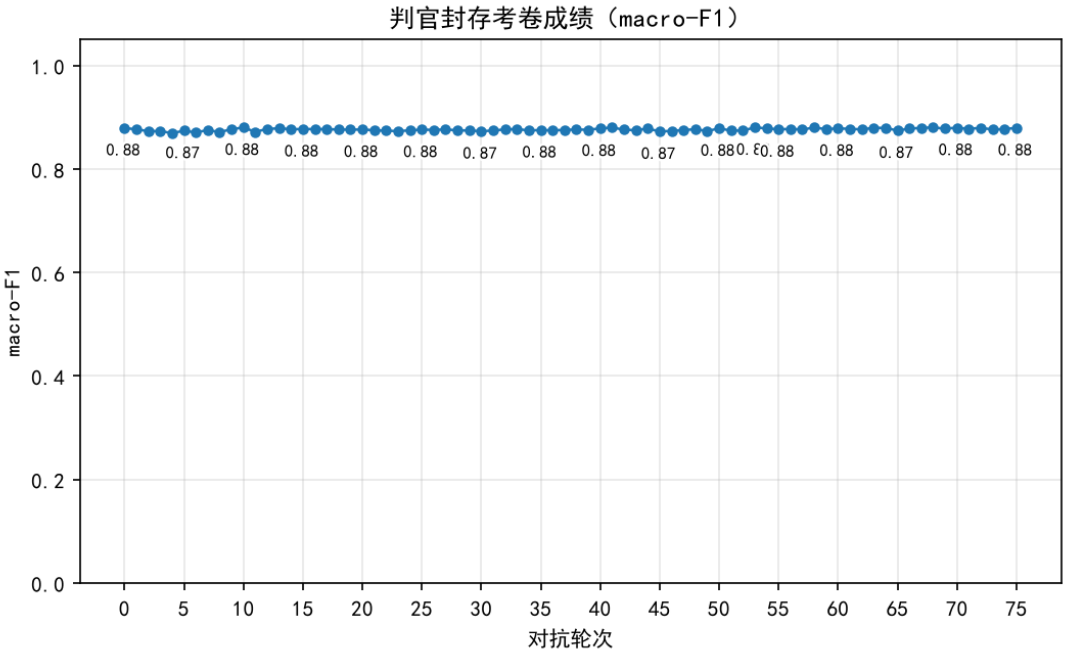



**图 3.12**　旧口径下的“判官封存考卷成绩（macro-F1）”曲线：75 轮对抗，成绩纹丝不动地钉在 0.87–0.88——这条漂亮得过分的水平线，恰恰成了 2026 年 9 月 15 日那次审计的起点：真实系统里，几乎没有哪条指标能这样一动不动

图片来源：本项目对抗训练 demo 产物（E:\中海油数据，demo_ 对抗训练 results）

2026 年 9 月 15 日，项目组对考卷本身动了审计——查的不是“系统考得多好”，而是“考卷配不配考”。结果三大水分，条条触目：

1. **考卷与训练数据事实级重叠 78%**：七成八的考题，其考查的事实在训练数据里出现过“同一事实的变体”——相当于期末考试考的全是课后原题的换皮。
2. **逐字重复率 15.7%**：一部分考题与训练样本存在整句级别的雷同——连换皮都懒得换了。
3. **判官 17 维特征中 11 维可查表**：判官赖以判卷的 17 个特征里，有 11 个能直接查表得到——等于开卷考试，还允许带标准答案进场。

结论只有一个：**旧口径下的 0.84–0.88 全部作废**。项目组重建考卷（斩断事实级重叠）、重建特征（砍掉可查表维度），把系统拉回真实水位重新考试——新的成绩落在 0.73 档。项目组在报告里写下一句后来被反复引用的话：“**0.72–0.74 是去掉水分后的真实水位，不是性能回退**。”数字变难看了，但数字第一次变得可信。

先补一句人话：本节反复出现的 macro-F1（宏平均 F1）是三分类（真话/造假/核不了）的综合成绩单：

$$\text{macro-F1} = \frac{1}{K}\sum_{k=1}^{K}\frac{2\,P_k R_k}{P_k + R_k}, \tag{3.1}$$

人话：$K$ 是类别数（这里 $K=3$），$P_k$、$R_k$ 是第 $k$ 类各自的查准率（判为第 $k$ 类的里面判对了多少）与查全率（真是第 $k$ 类的里面抓住了多少）。“宏平均”就是先给每一类各算一份 $F1$ 再取平均——少数类（比如“核不了”）不会被多数类淹没，谁差拉谁的平均，偏科藏不住。

重建之后，项目组做了四组对照实验，把每个组件的贡献摊在桌面上：

表 3.7　四跑对照：每个组件的贡献一次看清（新口径考卷）

| 轮次 | 配置 | macro-F1 | 备注 |
|---|---|---|---|
| Run A | 离线逻辑全开，64轮 | 0.7417 | 规则基线为 0.7261，离线闭环有效 |
| Run B | DeepSeek 增强对抗 | 0.7401 | 与 Run A 不可区分：增强是锦上添花，不是救命稻草 |
| Run C | 蒸馏后离线复测 | 0.7403 | 零 API 调用：云端能力已“蒸”进本地 |
| Run D | 全部关闭（消融） | 0.7190 | 跌破规则基线；ECE 0.210 全场最差 |

读这张表有三个收获。其一，Run A 比 0.7261 的规则基线高约 1.5 个百分点——离线闭环（蒸馏回流 + 先例库）真实有效。其二，Run B（0.7401）与 Run C（0.7403）几乎等于 Run A（0.7417）：云端大模型的增益**与离线不可区分**——铁律“LLM 只做增强，离线逻辑永远是兜底”不是口号，是可复现的实验结论。其三，Run D 把组件全关后成绩**跌破基线**，且 ECE 达到全场最差的 0.210（ECE，Expected Calibration Error，期望校准误差：模型“自信程度”与“实际对错”的偏差，越小越好——一个 ECE 很差的系统会一边出错一边理直气壮）。

还有一份更痛的诚实，写在 OOD 赛道上：300 道“纯文字/拼接造假”的分布外考题，F1 只有**0.071–0.233**。项目组没有把这几行数字藏进附录，而是原样写进验收材料，并附上一句诊断：“**系统对无数值痕迹的语义造假确实是盲的**。”品一品这句话的分量：当前防御体系依赖数值与单位等硬证据，一旦造假者只玩文字游戏，系统近乎失明——这正是组 3 的同学（第 3.8 节）接下来要啃的硬骨头。

**工程伦理：一份漂亮成绩单的死亡与重生——三条教训**

这次 0.88 → 0.74 的改判，值得每个工程人当伦理案例读。**教训一：掺水的地图比没有地图更危险。**指标是给决策者看的地图——拿着 0.88 的地图，甲方会放心把系统接进生产流程；水分挤破那天，信任一起破产。宁可交一份诚实的 0.74，不交一份虚胖的 0.88。**教训二：过于漂亮的曲线本身就是事故征兆。**75 轮纹丝不动的 0.88，与其说是成绩，不如说是“考卷漏了”的报警器——看到完美得反常的指标，第一反应应该是怀疑测量，而不是庆祝。**教训三：审计要审“考卷”，不只审“考生”。**系统没作弊、模型没作弊，出问题的是数据集与特征的构造方式——这类“无主观恶意的水分”只有靠制度化审计（查重叠、查重复、查特征泄漏）才能揪出。本项目从此立下规矩：**凡报指标，必附考卷说明与水分检查记录**——“如实条款”就是这样一章章攒出来的。

## 3.8　三层身份与验收文化

如果你只从本章带走一个工程习惯，我们推荐这一个：**给数据发身份证**。系统里任何一条信息，都要能回答三个“你是谁”：

- **内容身份**：这份内容本身是什么？——SHA-256指纹登记，同名不同内容分别记录（第 3.2 节的知识盒子）。
- **来源身份**：它从哪来？——锚点精确到“哪份资料、哪一页、哪一行、哪一格”，19,893个锚点就是 19,893 张户口页。
- **结果身份**：谁算出了它？——每次查询有唯一查询编号，答案、SQL、数据快照绑定存档，事后任何争议都能原样重放。

三层身份层层衔接，串成一条完整的责任链：用户问的每个数字（结果身份）能查到它来自哪张表（来源身份），那张表的每个字节又能验明正身（内容身份）。这套身份体系支撑着一套近乎偏执的验收文化。系统文档里写了三条格言，我们原文抄录：

- **“不以智能化、准确、高效等形容词代替说明。”**
- **“标准答案须独立编写并先行复审，系统生成的 SQL 不得反过来充当标准答案。”**
- **“静态截图、预填答案或人工拼接两个项目的结果不替代连接。”**

第一条治“吹”——形容词不算证据，数字才算；第二条治“自证”——考生不能自己批卷；第三条治“摆拍”——demo 里两个模块各自转得再漂亮，没真正接起来的管线就是没接起来。2026 年 10 月 12 日的验收就是按这套文化执行的，分三步：**贯通验收**——四条链（结构化链、问数链、门控链、图检索链）逐链判定，不许打总分和稀泥；**现场演示**——35 分钟五段，真数据、真提问；**三层核验**——按内容/来源/结果三层身份逐项对账。

验收的产出物里，最珍贵的一份是**缺口表**：G1–G11 共 11 条“系统还没做到的事”，逐条如实列出、逐条给出位置与建议。摘其中与本课程最相关的三条：**G8**——正式知识图谱尚未发布（现在沙箱里的图谱还没转正）；**G9**——R-GCN 尚未与业务数据联合验收（模型在测试集上能跑，还没在真实业务问题上验明正身）；**G10**——难负样本与对抗训练尚未形成成果（第 3.7 节那个 OOD 盲区，正是缺难负样本练手的症状）。缺口表不丢人——它恰恰是一份系统“活着”的证明；丢人的是那种通篇“达到国际先进水平”的验收报告。

而这三个缺口，恰好就是你们三个学生组的进场口。第 5–19 周，每组 5 人、每周投入 6–8 小时：

**表 3.8**　三个学生组与缺口表的对接（详见第三篇第 13 章至第 15 章）

| 组别 | 名称 | 对接的缺口与主线任务 |
|---|---|---|
| 组 1 | 数据结构化组 | 对接结构化侧缺口：39个局部回退区域、138 张列宽证据不足的表（第 3.2 节）——提升 B 类结构恢复的覆盖与质量 |
| 组 2 | 智能问答与图检索组 | 对接 G8/G9：推动正式图谱发布，完成 R-GCN 与业务数据的联合验收 |
| 组 3 | 虚假内容识别与对抗训练组 | 对接 G10：把难负样本与对抗训练做成成果，正面强攻 OOD 语义造假盲区 |

**项目连线：三个项目，同一条纪律**

把三个项目并排放，你会看到同一条纪律的三种长相。**数据资产化**：六目拼接的全景图带拼接参数（第 一 章）、水下双目的图对带 meta.json（第 二 章）、数据基座的表格带锚点与 SHA-256——都坚持“没有身份的数据不是资产”。**先过滤再入库**：伪标签要过四重过滤、知识要过门控、假条目要过监考与判官——都拒绝“生米直接下锅”。**如实文化**：拼接项目登记配准失败情形、双目项目公布误差的分布尾巴、数据基座发布缺口表 G1–G11——都把“没做到的”写进交付物。这套纪律不属于某个行业，它属于所有想把 AI 用在正经地方的人。

## 3.9　回头看：你将用到第一篇的哪些本事

本章出现了大量“黑话惊魂”：向量检索、量化部署、Agent、对抗训练、macro-F1……别慌，它们全部能在第一篇找到老巢。下表是对照清单——学第一篇时带着右栏的问题去学，到第 5 周进场时你就是组里“看得懂全局”的那个人：

**表 3.9**　第一篇知识在本项目中的落点

| 第一篇 | 讲什么 | 在本项目哪里用上 |
|---|---|---|
| 第 3 讲 | 训练与评估 | 500 题回归考卷、macro-F1（式 (3.1)）、置信区间、ECE——“先定怎么考，再谈学得好”是全项目的方法论底座 |
| 第 4 讲 | 自编码器与“表示” | Granite 的 768 维语义向量、BGE 的 1,024 维实体向量、投影到 256 维排序空间——“用向量表示意义”一以贯之 |
| 第 6 讲 | CNN 训练循环 | 蒸馏样本 3 倍权重回流、复盘再训练——“数据回流、模型更新”就是训练循环思想在数据侧的投影 |
| 第 8 讲 | 轻量化与部署 | Gemma 12B 用 llama.cpp 本地量化部署，上下文 8,192、并发 1——“本地小模型 + 严格边界”是涉密场景的务实路线 |
| 第 11 讲 | 注意力与大模型 Agent | 智能体“思考–行动–观察”循环、8 个白名单工具、8,192 上下文预算的精打细算 |
| 第 12 讲 | 生成对抗与蒸馏 | 造假者 vs 监考/判官是 GAN 博弈思想的“数据质检版”；蒸馏回流与判官先例库是博弈成果的沉淀方式 |

**前沿速递：2024–2026，数据治理撞上大模型**

本章的每一环都在 2024–2026 年经历着快速演化，四条线索值得关注。其一，**RAG → GraphRAG**：2024 年起，业界把知识图谱引入检索增强生成（GraphRAG），用社区摘要与多跳关系回答“全局性、关系性”问题——本章第 3.5 节正是这条路线的工程化迷你版。其二，**Text2SQL 的评测革命**：学术界 Text2SQL（自然语言转 SQL）基准从 Spider 演进到 Spider 2.0 与 BIRD，专门考“真实企业级数据库”上大模型的本事——结论很有趣：模型写 SQL 的“雅思分数”年年涨，但在真实库上“问数”远没到让人类放手的程度，本章 SQLGlot/Wren/DuckDB 的多关卡设计正是对这一现实的回应。其三，**LLM 标注的质检革命**：用大模型给数据打标已是行业常态，而 2024–2026 年的重心从“LLM 当裁判”转向质检——金标抽检、一致性审计、标注泄漏检查，与本章门控/监考/判官的三层设卡同构。其四，**合成数据的双刃剑**：Shumailov 等人在 Nature（2024）警告“模型在递归生成的数据上反复训练会崩溃”（model collapse）——合成数据省标注，但必须像本章这样“程序给定真值 + 双金标 + 水分审计”，否则是在用自己产生的回声训练自己。

章末把开篇那道题收个尾。“把 2023 年 11 月 18 日 20 点 27 分前后的所有信息给我”——现在你知道这句话背后的全部代价：1,001 页要拆出 484 张表、272 段公式、19,893 个锚点；3.1 亿行要登记、分区、建成四个视图；一个五秒钟的答案背后，是 8 个白名单工具、三道计算关卡、六成候选知识被拒之门外、26 个造假家族日夜攻防，以及一次把 0.88 改判成 0.74 的审计。三个组的位置已经留好：结构化在第 13 章，问答与图谱在第 14 章，攻防在第 15 章。去把缺口表上属于你们的那几行，划掉。

**数据结构化（data structuring）**

把 PDF/Word 等非结构化文档还原成“表格是表格、公式是公式、段落有出处”的结构化数据；本项目 1,001 页产出 484 张表、272 段公式源码、19,893 个锚点。

**锚点（anchor）**

“哪份资料、哪一页、哪一行、哪一格”式的出处坐标；全系统可追溯性的地基，每个数字“点回原件”靠它。

**SHA-256**

一种 256 位指纹函数：内容变一个字节、指纹面目全非；本项目给每份内容登记“内容身份证”，同名不同内容分别记录。

**智能体（agent）**

能理解意图、调用工具、组织回答的程序；本项目的大脑是本地部署的 Gemma 12B（上下文 8,192，温度 0，并发 1）。

**工具白名单（tool whitelist）**

只允许智能体调用事先登记的固定工具（本项目 8 个），无任意文件读写、无命令行——能力边界由程序而非“自觉”保证。

**Text2SQL**

自然语言转 SQL 的技术；本项目由大模型起草 SQL，但须经 SQLGlot 校验、Wren 规划后才由 DuckDB 执行。

**DuckDB**

单机分析型数据库，擅长对大表做筛选与聚合；本项目的“算数员”——所有数字由它算，模型只转述。

**Parquet**

列式存储文件格式，按列组织数据；本项目把 CSV 核验后按月分区转成 Parquet，查询时不必读全表。

**知识图谱（knowledge graph）与三元组**

用“主体–关系–客体”三元组把事实连成的网络：节点是实体（设备/系统/平台），边是关系；擅长回答“关系题”而非“相似题”。

**R-GCN（Relational Graph Convolutional Network，关系图卷积网络）**

按关系类型分别聚合邻居信息的图神经网络；本项目叠三层，把 1,024 维实体向量投影到 256 维排序空间给证据打分。

**GraphRAG**

基于图的检索增强生成：回答前先沿知识图谱取“实体与关系证据”，再交给生成模型组织答案。

**知识门控（knowledge gating）**

AI 抽取的候选知识入库前过三道检查（是否支持/是否同一对象/关系是否成立），按“通过/转复核/拒绝”三出口分流；本项目 9,505 条候选接纳 2,988、待复核 582、拒绝 5,935。

**对抗训练（adversarial training）**

让“造假者”与“防御方”持续博弈、互相逼强的训练范式；本项目四角色闭环——造假者、监考、判官、库存管家，登记 26 个造假家族。

**OOD（out-of-distribution，分布外）**

训练时没见过的数据分布；本项目考卷含 300 道 OOD 题，现成绩 F1 仅 0.071–0.233——“语义造假盲区”的如实公告。

# 第三部分

# 第三篇　任务书篇：15 个小组的任务、实验与验收

# 第三篇使用说明：任务书怎么读、怎么交、怎么评

**第三篇地图**

**讲什么**：15份任务书共同的游戏规则——任务书的固定结构怎么读、证据包三件套怎么留、AI与人怎么分工、成绩怎么算、第 19 周周五交什么，外加一张 15 组速查表。**为什么值得读**：本章是全篇的“总则”。先花二十分钟读通这里，再翻自己组的章，验收线上每一条你都见过同类，不会临场发懵。**学完你能**：说清“无证据 = 未完成”“AI 可以用、结果自己扛”两条铁律，把自己的提交清单与四个实验日期抄进日历。

第二篇讲的是三个项目“为什么做”（六目拼接见第 一 章，水下双目见第 二 章，数据基座见第 三 章），这一篇回答“你具体做什么”。全班 15 个小组（每组 5 人，每周合计投入 6–8 小时，组内自行排班），每组一章任务书。本章是 15 份任务书的总则：先读这里，再看自己组的章，两章合起来才是你这学期的完整说明书。

## 0.10 任务书怎么读：固定七节，先通则后本组

每份任务书的内部结构完全相同，固定七节：**组档案**（你是谁、跟谁交接）、**项目背景与你的位置**（为什么是你）、**核心任务**（要做什么）、**第 19 周周五提交清单**（最后交什么）、**AI 分工与人工必做**（谁干什么）、**防糊弄指南与满分标准**（怎么算分）、**四次实验安排**（什么时候做到哪）。建议的读法是“先两头后中间”：第一遍只读组档案和提交清单，五分钟知道自己组的边界；之后跟着四个实验节点的节奏，逐节精读其余部分。

全项目共用一套代码基座。六目拼接的八个组工作在 `stitcher_cpp` 目录（VS2022 工程，入口 `360stitcher.sln`），端到端路线在 `e2e_gpu` 目录；各组的训练代码与数据放在 `stitcher_cpp/training/groupXX`，训练好的模型统一交付到 `stitcher_cpp/build/Release/models/`。水下双目与数据基座项目的目录约定，见各自任务书的组档案一节。你的四个实验日期是全班统一的：实验一 2026 年 10 月 12 日、实验二 11 月 2 日、实验三 11 月 23 日、实验四 12 月 21 日；完全提交离场是 2027 年 1 月 22 日（第 19 周周五）。

## 0.11 证据怎么留：三件套、交付契约与求助通道

第一件事，**证据包三件套**。凡声称“我做出来了”，必须同时拿出三样东西：**命令全文**（你敲的是哪条命令，原样复制）、**完整日志**（终端从开始到结束的完整输出，不许截选）、**产物路径**（跑出来的文件在哪、叫什么）。三件缺任何一件，一律按“无证据 = 未完成”处理——截图可以摆拍，口说无凭，只有三件套能被别人原样复跑。

第二件事，六目拼接组还要记住**九头交付契约**，见下表。它设计的全部目的，是让“装样子”在技术上过不了关：探针会真去加载你的模型跑一遍，模型内容一变、输出必须跟着变，这就是“探针防绕过”。

表 10　六目拼接九头交付契约（逐条对照，缺一不过）

| 契约条款 | 具体内容 |
|---|---|
| 交付物 | 每个参数头一个`.onnx` 模型文件，配一个同名的`.metrics.json` 指标文件（文件名中的头名必须与头名完全一致，探针按文件名找文件） |
| 交付位置 | `stitcher_cpp/build/Release/models/` |
| 模型探针 | `python tools/nine_head_probe.py`，期望 `PASS (12/12)` |
| 引擎探针 | `python tools/engine_probe_test.py`，期望 `PASS (5/5)` |
| A/B 对比 | `python tools/ab_compare.py before.json after.json`：同一批数据，新模型（after）与基准（before）同场比武 |
| 回退保护 | 模型损坏或输出异常时系统经 `[ProviderFallback]` 自动退回人工参数，不崩溃——坏模型只会没用，不会坏事 |

第三件事，**求助通道**：组内互助 → 项目助教（研究生）→ 导师，逐级上升。求助必须附“完整命令 + 完整输出”——这不是刁难，而是排查问题的唯一捷径：只发一句“老师我跑不通”，助教只能让你重发；带上完整输出，八成的坑一眼可见。

## 0.12　AI 怎么用：分工通则与两条红线

这门课不禁止 AI，反而**要求**你用 AI——但用得明白。通则一张表：左列的事可以放心交给 AI，右列的事永远自己来。

表 11　AI 分工通则：左列交给 AI，右列必须亲自做

| AI擅长（可直接交给它） | 人必做（不许外包） |
|---|---|
| 解读项目代码、解释每段在干什么 | 在自己电脑上运行每一条命令 |
| 写解析脚本、画图代码、文档草稿 | 核对 AI 给出的每个结论与数字 |
| 解释报错信息、给出排查建议 | 亲眼看输出、看图、看训练曲线 |
| 复述论文与文档、翻译术语 | 在证据包上签字，对每个数字担责 |

两条红线之下，各有一份防糊弄指南，先在这里立此存照，各组任务书里还会结合本组任务重申。

**防 AI 糊弄指南（通则）**

**(1) AI 给的数字必须本机复跑**：它报的 loss、指标、耗时，只有你自己跑出来的那一份是真的。**(2) AI 引用的库版本与函数签名要核实**：它张口就来的 API 十有八九带着幻觉。**(3) AI 说“应该可以”“一般没问题”不等于跑通了**：跑没跑通，只认你的终端输出。**(4) 禁止把 AI 的回答直接粘进实验记录**：实验记录里只许出现你亲手跑出来的内容。

**防人工糊弄指南（通则）**

**(1) 跑一次成功就截图当十次的证据，不行**：每个节点各有验收线，一次表演补不齐四季庄稼。**(2) 改日志时间戳、拼接不同日期的输出 = 作弊**，按如实条款处理。**(3) 组内“代跑”必须留运行者姓名与时间**：谁跑的谁负责。**(4) 编造训练精度 = 该节点成绩作废**：这是如实条款，全

课程无弹性——工程世界里造假不是扣分，是出局。

## 0.13 成绩怎么算：建议方案（教师可调）

成绩构成的建议方案是：**实验一不计分（入场达标实验，必须通过验收）；实验二、三、四各 10 分，共 30 分；平时作业 20 分——学期两次、每次 10 分，从十二讲（第 1–12 讲）章后习题中任选 2 题作答；期末项目成果 50 分。总评满分 100 分。**实验分看你在二、三、四三个节点上是否达成当次验收线；作业分看两次习题作答的完成质量；成果分看第 19 周周五交出的整体质量。期末成果 50 分的典型构成见下表，各组任务书里的细则与此同构。

**表 12** 期末项目成果 50 分的典型构成

| 分值 | 项目 | 看什么 |
|---|---|---|
| 20 | 功能达成 | 任务书里“功能达成”一栏的动作是否全部落地（如模型交付、探针通过、标签达标） |
| 10 | 证据完整 | 四个节点的证据包三件套是否齐全、能否被他人原样复跑 |
| 10 | 报告质量 | 组报告按模板撰写，方法讲得清、图表齐全、结论有数据支撑 |
| 10 | 协作与台账 | 排班执行、组内互助与求助记录规范、台账条目无缺失 |

再次声明：**以上为建议方案，教师可调。**评定的唯一依据是各组维护的台账与证据包——不是谁嗓门大，是谁的记录全。

## 0.14 第 19 周周五交什么：总清单模板

2027年 1 月 22 日（第 19 周周五）完全提交离场。每个组都要交齐下表四件东西；各组任务书的提交清单表，是本表加上组内特有交付物（如六目各组的三头 ONNX 模型）之后的版本，以任务书为准。

**表 13** 第 19 周周五提交总清单模板（15 组通用）

| 内容 | 格式 | 说明 |
|---|---|---|
| 项目报告 | PDF，8–15 页 | 模板六段：摘要 / 背景 / 方法 / 实验 / 结论 / 分工声明 |
| 代码包 | zip，含 README | README 写清环境与运行步骤，别人拿到能跑 |
| 证据包 | 目录或 zip | 按四个实验节点分四包，命令 + 日志 + 产物三件套 |
| AI 使用声明 | 随报告附一页 | 哪些事 AI 做了、人做了什么核验，逐条列清 |

其中“分工声明”一节请认真写：谁负责哪部分、谁运行了哪些关键实验——它既是协作分的依据，也是你个人的项目履历。AI 使用声明则与防糊弄指南配套：用 AI 不是错，说不清 AI 做了什么、你核验了什么才是错。

## 0.15 15 个小组速查表

最后一张地图。找到自己组的章号，翻过去之前，先把本章的总则过一遍。

**表 14**　15 个小组与任务书章号速查

| 组号 | 组名 | 任务书 |
|---|---|---|
| 第 1 组 | 六目拼接 • 匹配决策组 | 第 1 章 |
| 第 2 组 | 六目拼接 • 特征质量组 | 第 2 章 |
| 第 3 组 | 六目拼接 • 缝合融合组 | 第 3 章 |
| 第 4 组 | 六目拼接 • 数据与标签组 | 第 4 章 |
| 第 5 组 | 六目拼接 • 测试集与评测组 | 第 5 章 |
| 第 6 组 | 六目拼接 • 端到端数据组 | 第 6 章 |
| 第 7 组 | 六目拼接 • 端到端训练组 | 第 7 章 |
| 第 8 组 | 六目拼接 • 端到端优化组 | 第 8 章 |
| 第 9 组 | 水下双目 • 数据组 | 第 9 章 |
| 第 10 组 | 水下双目 • 部署组 | 第 10 章 |
| 第 11 组 | 水下双目 • 训练组 | 第 11 章 |
| 第 12 组 | 水下双目 • 评测组 | 第 12 章 |
| 第 13 组 | 数据基座 • 数据结构化组 | 第 13 章 |
| 第 14 组 | 数据基座 • 智能问答与图检索组 | 第 14 章 |
| 第 15 组 | 数据基座 • 虚假内容识别与对抗训练组 | 第 15 章 |

# 第 1 组　六目拼接 · 匹配决策组（九头之一：匹配与几何）

本章地图

**讲什么**：第 1 组任务书——policy / mc / geometry 三个参数头各管什么、四个月里每个实验做到什么程度、最后交哪几样东西。**为什么值得读**：探针 12/12、A/B 对比、metrics.json 这些验收线，每一条都对应成绩单上的一栏；先读懂再动手，不做无用功。**学完你能**：向任何人说清三个头的输入输出与训练目标，照命令级步骤独立完成四次实验，知道每一步该留下什么证据。

一句话概括你们组：拼接流水线的“第二步半”——特征匹配之后、几何对齐前后，三只小网络替老师傅做三个决定：这一对画面**怎么拼**（policy）、**信几分**（mc）、**几何参数给多少**（geometry）。九个参数头里最靠前的一环，归你。

## 1.1　组档案

表 1.1　第 1 组档案

| | |
|---|---|
| 项目 | 六目拼接（项目一，360Stitcher实时全景系统，详见第 一 章） |
| 组号与组名 | 第 1 组 • 匹配决策组 |
| 人数与投入 | 5 人；每周合计 6–8 小时（组内自行排班，建议每周固定 2 次） |
| 负责的头 | policy（匹配策略）、mc（匹配质量分）、geometry（几何参数） |
| 上游依赖 | 开局即可训练：组 4 组的公开数据仿真标签（UDIS-D/COCO 图对喂 simulation 管线产出）；真机标签实验四接入 |
| 下游依赖 | 第 5 组：你的三个头交付后指标入双台账 |
| 工作目录 | `stitcher_cpp/training/group01_policy`、`group04_mc`、`group05_geometry` |
| 硬件要求 | 本机无 GPU 亦可：三个头都是几百个参数的小网络，CPU 训得动 |

先排一颗雷：三个工作目录名里的数字（group01 / group04 / group05）是源工程里参数头的**原始编号**，与本课程“第 1 组”无关——别看到 `group04_mc` 就以为自己走错了门。第 1 组的三个头分散在三个目录里，每个目录自带 `train.py`，结构相同，学会一个就会三个。

## 1.2　项目背景与你的位置

第 一 章讲过来龙去脉：360Stitcher 把六路视频实时拼成 360° 全景，五步曲的第二步“特征提取与匹配”、第三步“几何对齐”，至今靠人工经验参数撑着——ORB 取 1500 点、比例阈值多大、RANSAC 距离阈值多严，全是老师傅手调的写死的数。这学期你们的任务，就是把这三个决定交给数据。你在流水线里的位置：六路画面完成畸变校正、提完 ORB 特征之后，**先由 mc 给这对画面的匹配质量打分**（分太低就不硬拼，自动退回老参数），**再由 policy 三选一定拼接策略**，**最后由 geometry 给出几何参数**供对齐使用——三个头首尾相接，是整条九头链路的入口。下图圈出了你的三块地盘。

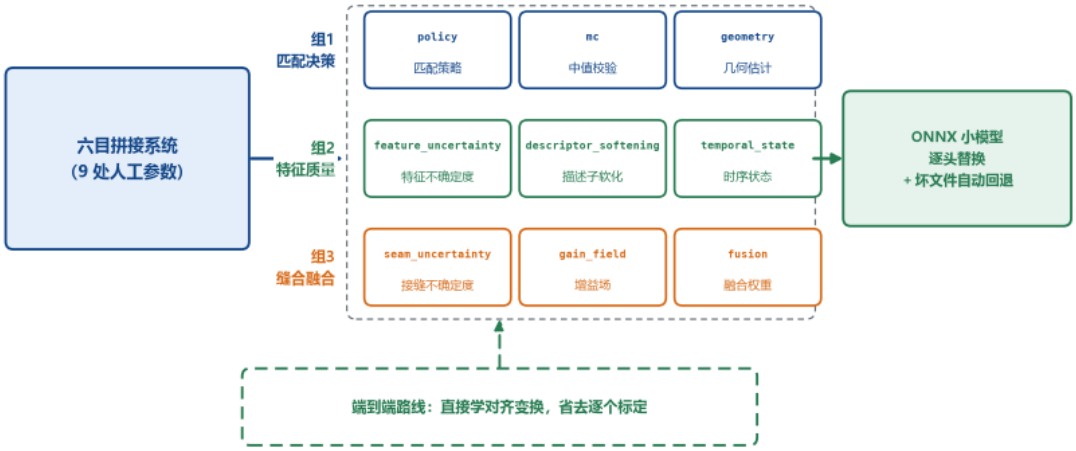



**图 1.1**　九头地图上的第 1 组：policy / mc / geometry 三个头（匹配决策组）站在九头链路的最前端，吃匹配统计量、吐策略与参数，每个头独立训练、独立交付

## 1.3　核心任务

三个头一张表看明白，随后逐头说人话。

**表 1.2**　第 1 组三个参数头：输入 → 输出与人话职责

| 头名 | 形状（输入 → 输出） | 一句话职责 | 第一篇血缘 |
|---|---|---|---|
| policy | [B,8] → [B,3] | 读 8 项匹配统计量，三选一决定这条接缝走哪条拼接策略 | 第 1–3 讲 |
| mc | [B,10] → [B,1] | 给本轮特征匹配打一个置信分，分低就别硬拼 | 第 0–3 讲 |
| geometry | [B,16] → [B,5] | 从 16 项统计特征回归 5 个几何参数，接管手调阈值 | 第 2–3 讲 |

方括号是张量形状：B 是一次一起送进网络的一批样本数（batch），[B,10] 就是“一批若干行、每行 10 个数”。逐头展开：**policy** 是一道三分类题——同一个重叠区，可以用常规策略拼、用保守策略拼、或者干脆放弃这帧，网络的输出是三选一；**mc** 是一道回归题——输入 10 项匹配统计量（内点数量、内点率、得分分布等），输出一个 0 到 1 的质量分，它是系统的“信心开关”；**geometry** 还是回归题——从 16 项统计特征预测 5 个几何参数，把“老师傅手感”变成确定数。每个输入列具体是什么字段，以 `train.py` 读到的数据列与 `metrics.json` 的字段为准——这正是实验一要你亲手弄清楚的事。

**项目连线：你的三个头与第一篇的哪几讲**

三个头都是“统计量进、几个数出”的小型全连接网络：骨架是第 1 讲的感知器，引擎是第 2 讲的反向传播；训没训好、会不会过拟合，全部按第 3 讲的验证集方法论执行；mc 打分的直觉原点是第 0 讲的“点积打分器”。参数量比第 6 讲的 LeNet 还小一个量级，所以本机无 GPU、CPU 训练毫无压力——训练瓶颈从来不是算力，而是数据与耐心。

## 1.4　第 19 周周五提交清单

提交线：2027年 1 月 22 日（第 19 周周五）完全提交离场。你的组交齐下表五样，每样都与第 三 章的总清单模板对应。

表 1.3　第 1 组提交清单（2027 年 1 月 22 日）

| 交付物 | 格式 | 说明 |
|---|---|---|
| 三头模型与指标 | 6 个文件 | `mc.onnx` + `mc.metrics.json`、`geometry.onnx` + `geometry.metrics.json`、`policy.onnx` + `policy.metrics.json`；文件名 = 头名，拷入 `stitcher_cpp/build/Release/models/`，交付后探针须仍 `PASS (12/12)` |
| 训练证据包 | zip | 四次实验各一包，命令 + 日志 + 产物三件套，日志为终端完整输出 |
| A/B 对比报告 | PDF，2–4 页 | `ab_compare.py` 的 before/after 数据表与结论，含 mc 三版对比 |
| 小组项目报告 | PDF，8–15 页 | 按第 三 章模板：摘要/背景/方法/实验/结论/分工声明 |
| AI 使用声明 | 一页 | 哪些事 AI 做了、人做了什么核验，逐条列清 |

## 1.5　AI 分工与人工必做

分工原则一句话：AI出草稿，人出签字——签字的人对数字负全责。

表 1.4　第 1 组 AI 分工与人工必做

| AI可以自动做 | 人必须亲自做 |
|---|---|
| 解读 `train.py` 与 `metrics.json` 字段含义 | 本机跑通全部命令：编译、训练、探针、A/B |
| 解释报错信息、给排查建议 | 逐条核对 AI 结论（在代码里找到对应行才算数） |
| 写数据预处理与画图脚本草稿 | 拷贝交付物到 `models/` 并重跑探针确认 |
| 起草报告与 A/B 报告的文字段落 | 在 metrics.json 与证据包上签字，对数字担责 |

## 1.6　防糊弄指南与满分标准

**防人工糊弄指南（第 1 组版）**

**(1)** 跑一次成功就截图当十次的证据，不行——四个节点各有各的验收线，一次表演补不齐全部证据。**(2)** 改日志时间戳、拼接不同日期的输出 = 作弊。**(3)** 组内“代跑”必须留运行者姓名与时间，谁跑的谁负责。**(4)** 编造训练精度 = 该节点成绩作废：metrics.json 里的每个数字都要能与训练日志对上账，对不上就是作废，没有商量。

**防 AI 糊弄指南（第 1 组版）**

**(1) AI 给的数字必须本机复跑**——它随口报的 loss 与准确率，只有你终端里的那份是真的。**(2) AI 解释字段爱“脑补”**：8 项匹配统计量到底是哪 8 项，以代码与数据文件为准，它背的教科书不算。**(3) AI 说“这样应该可以跑通”不等于跑通了**，跑没跑通只看你的终端。**(4)** 禁止把 AI 的回答直接粘进实验记录当结论。

成果满分 50 分，怎么拿每一分见下表。

**表 1.5**　第 1 组成果满分标准（50 分制）

| 分值 | 项目 | 怎么拿满 |
|---|---|---|
| 20 | 功能达成 | 实验三交付 mc 头并完成 A/B（8 分）；实验四交付 geometry、policy 两头（7 分）；每次交付后重跑探针仍 `PASS (12/12)`（5 分） |
| 10 | 证据完整 | 四次实验证据包三件套齐全、可复现；缺一件扣 5 分 |
| 8 | 报告质量 | 组报告按模板 8–15 页，方法讲得清、图表齐全、结论有 A/B 数据支撑 |
| 7 | 协作与台账 | 排班执行有记录；组 5 双台账中本组条目无缺失；求助记录规范 |
| 5 | 加分 | 如：为 mc 设计新的统计特征并用 A/B 证明提升；对三个头做联合分析并给出数据 |

## 1.7　四次实验安排

四个节点与全班同频，但每场都有本组的命令级动作。先看总览表，再逐场展开；每场实验按“目标—步骤—验收证据—常见坑”执行：步骤照抄就能跑，证据一栏就是成绩。

**表 1.6**　第 1 组四次实验总览

| 节点 | 日期 | 本组动作 | 核心验收证据 |
|---|---|---|---|
| 实验一 | 10月 12 日 | agent 解读 `train.py` 三问，人工核对 | 解读 + 核对记录一页 |
| 实验二 | 11 月 2 日 | 本机部署 + 基准 + 探针 | 三段日志 + `PASS (12/12)` |
| 实验三 | 11 月 23 日 | `--smoke`、`--telemetry` 训练并交付 mc，A/B | 训练日志、`metrics.json`、A/B |
| 实验四 | 12 月 21 日 | 交付 geometry、policy，真机标签复训 mc 联调 | 两头交付 + 三版 mc 对比表 |

### 1.7.1　实验一（10 月 12 日）：让大模型替你读代码，你来当裁判

**目标**：配好大模型编程助手的 agent 模式，让它解读 `group04_mc/train.py`，你负责核对它说得对不对——这是全组整个学期训练功课的第一份预习。

**步骤**：(1) 按课堂演示装好编程助手并切换到 agent 模式；(2) 打开 `stitcher_cpp/training/group04_mc/train.py`，让 AI 回答三个问题：数据从哪来（读哪个目录、哪些文件、哪些列）、网络结构长什么样（几层、每层多少神经元、什么激活函数）、输出是什么（形状与每个数的含义）；(3) 把 AI 的回答

逐条与代码对照，标注“对 / 错 / 存疑”；(4) 至少纠正 AI 一处错误，写明你在代码里找到的依据。

**验收证据**：一页“AI解读 + 人工核对”记录，含 AI 原话、你的核对结论、纠正错误至少一处。

**常见坑**：AI 会顺手“脑补”——比如说脚本用了 GPU 或 dropout，而它其实就是 CPU 上的小网络；提示词里不写清文件名，它就给你背教科书；只问一遍就信是最大禁忌——换个问法再问一遍，两次答案打架的地方，就是你必须翻代码的地方。

### 1.7.2　实验二（11 月 2 日）：本机部署，亲眼看见九个头上线

**目标**：在自己电脑上把 360Stitcher 编译出来，看到九个头以 ONNX 形式挂上系统，并用探针确认部署无误。

**步骤**：(1) 用 VS2022 打开 `stitcher_cpp/build/360stitcher.sln`，把配置切到 `Release` 后整体生成；(2) 命令行进入 `build/Release` 目录，运行 `360stitcher.exe --benchmark-stitch=deploy/simulation --provider=onnx`，日志应出现九行 `[Provider]` 加载记录（形如 `wired: orb=1500`）；(3) 再运行 `python tools/nine_head_probe.py`，期望输出 `PASS (12/12)`。

**验收证据**：编译成功末段日志、九行 [Provider] 日志、探针 `PASS (12/12)`，三段终端完整输出。

**常见坑**：配置忘了切 Release，Debug 版一路报错找不到 dll；探针 FAIL 先查 `models/` 下模型文件名是否与头名完全一致（探针按文件名找文件）；杀毒软件可能静默隔离 onnx 文件；不在 `build/Release` 目录下运行，程序找不到 `deploy/simulation` 基准图。

### 1.7.3　实验三（11 月 23 日）：mc 头真训练、真交付、真对比

**目标**：完整走通第一个头的全流程——先冒烟验证，再真数据训练，交付上系统，用 A/B 数据说话。

**步骤**：(1) 进入 `training/group04_mc`，先跑 `python train.py --smoke`——小数据几秒钟跑通全链路，验证代码与数据格式无误；(2) 再跑 `python train.py --telemetry`，用组 4 的公开仿真标签训练（真机标签实验四替换），记下最终指标并生成 `metrics.json`；(3) 把 `mc.onnx` 与 `mc.metrics.json` 拷入 `stitcher_cpp/build/Release/models/`，重跑探针必须仍 `PASS (12/12)`；(4) A/B 对比：`python tools/ab_compare.py before.json after.json`，before 是换模型前的基准结果，after 是换模型后的。

**验收证据**：`--smoke` 与 `--telemetry` 两段完整训练日志、`metrics.json`、探针 PASS、A/B 对比结果，四个证据缺一不可。

**常见坑**：跳过 `--smoke` 直接真训，数据少一列报错能排查半小时——冒烟 5 秒钟，救你一晚上；A/B 的文件路径写成相对路径而命令行不在对应目录，工具找不到文件；交付时顺手改了文件名（如 `mc_v2.onnx`），探针立刻 FAIL——文件名 = 头名，版本信息写进 `metrics.json`，别写进文件名。

### 1.7.4　实验四（12 月 21 日）：补齐两头，并用新标签复训 mc

**目标**：交付 geometry 与 policy 两个头；真机标签联调——组 4 录制的真实标签接入，重训 mc 得第三版，公开版/真机版 A/B 对账，用三版数据回答“数据变多到底管不管用”。

**步骤**：(1) 按实验三的流程（先 `--smoke` 再 `--telemetry`）先后训练并交付 geometry、policy；(2) 组 4 录制的真实标签（真机标签）接入，重训 mc 得到第三版；(3) 做三版 mc 对比表：基线（人工参数）/ 实验三公开仿真标签版 / 真机标签复训版，各列关键指标与 A/B 结果，公开版/真机版逐项对账，并写一段不少于 100 字的结论。

**验收证据**：两个头的训练日志与 `metrics.json`、交付后探针 PASS、三版 mc 对比表加结论段。

**常见坑**：复训覆盖旧模型前忘了保留 before.json——没有旧版数据就没有对比；三个头想一口吃成胖子，最后一头都没训完——按 mc → geometry → policy 串行推进，每个头交付验收后再开下一个。

# 第 2 组　六目拼接 · 特征质量组（九头之二：特征与时序）

**本章地图**

**讲什么**：第 2 组任务书——三个“特征质量”参数头的规格怎么读、时序头怎么训、两个图像头在数据到位前该干什么。**为什么值得读**：你们组的三个头形状最杂（有一维统计量、有 32×32 特征图、还有时序窗口），对应三种不同的网络——任务书把每种怎么落地写成了命令级步骤。**学完你能**：独立读懂 `head_specs.py` 的规格定义，训好并交付 temporal_state 头，把两个图像头从设计文档一路带到真训练交付。

一句话概括你们组：特征环节的“质检科”。第 1 组的 mc 头给“一对画面”打分，你们管得更细：一个头给**特征图本身**挑毛病（feature_uncertainty），一个头给**二进制指纹**学会看场合（descriptor_softening），一个头盯着**最近一段时间**判断跟踪状态（temporal_state）。

## 2.1　组档案

**表 2.1**　第 2 组档案

| | |
|---|---|
| 项目 | 六目拼接（项目一，360Stitcher实时全景系统，详见第 一 章） |
| 组号与组名 | 第 2 组 • 特征质量组 |
| 人数与投入 | 5 人；每周合计 6–8 小时（组内自行排班，建议每周固定 2 次） |
| 负责的头 | feature_uncertainty、descriptor_softening、temporal_state |
| 上游依赖 | 开局即可训练：组 4 组公开仿真时序标签 + HPatches/MegaDepth 公开 patch（32×32，与正式规格一致）；总体组真 patch 约第 11 周到位，实验四迁移转真 |
| 下游依赖 | 第 5 组：三个头交付后指标入双台账 |
| 工作目录 | `stitcher_cpp/training/` 下本组三个头的子目录；三头规格的单一事实来源是 `training/common/head_specs.py` |
| 硬件要求 | 本机无 GPU 亦可：三个头都是小网络，CPU 训得动 |

特别记一笔你们组的“数据节奏”：三个头的训练数据开局就有——temporal_state 吃组 4 组的公开仿真时序标签，两个图像输入头用 HPatches/MegaDepth 公开 patch 先行训练（32×32 与正式规格一致）；总体组在第 11 周前后提供的真 patch 数据，用于实验四的迁移转真。所以你的三条线从第一周就是“训练态”——公开版与真机版的指标在报告里分开写、如实注明数据来源。

## 2.2　项目背景与你的位置

第 一 章讲过：360Stitcher 的第二步用 ORB 提特征、做匹配，而特征这层一直是“盲盒”——提取出来的特征图质量如何、256 位二进制指纹可不可信、帧间跟踪的惯性还灵不灵，系统全凭经验假设。你们的任务就是给这个盲盒装上仪表盘。你在流水线里的位置：五步曲第二步的“质检岗”——

**feature_uncertainty 扫一眼特征图，报出这帧特征有多不可靠；descriptor_softening 给指纹比对加软权重；temporal_state 读最近 T 帧，告诉跟踪器“该信还是要重置”**。三个头的意见汇总起来，决定这一帧的特征信息被采信几分。

## 2.3　核心任务

三个头一张表，随后逐头说人话。

**表 2.2**　第 2 组三个参数头：输入 → 输出与人话职责

| 头名 | 形状（输入 → 输出） | 一句话职责 | 网络类型 |
|---|---|---|---|
| feature_uncertainty | [B,1,32,32] → [B,3] | 扫一遍 32×32 特征图，报出 3 个“这帧特征有多不可靠”的指标 | CNN |
| descriptor_softening | [B,1,32,32] → [B,256] | 给 256 位描述子的每一位输出“是 1 的概率”，让生硬的位比对学会看场合 | CNN |
| temporal_state | [B,T,24] → [B,4]×2 | 读最近 T 帧的 24 维时序特征，输出两个 4 维状态：跟踪该信谁、要不要重置 | 时序网络 |

**descriptor_softening** 最值得细品：BRIEF 指纹是 256 位 0/1，传统比对是“数不同位”（汉明距离）——硬碰硬，一位就是一位，没有商量。软化头给每一位输出一个 0 到 1 的概率 $P(\text{bit}=1)$，比对时按置信度加权：纹理清晰区域的位多说话，模糊区域的位少说话。匹配从“非黑即白”变成“看场合说话”。**temporal_state** 则是把第 9–10 讲的东西搬进真系统：滑动窗口读最近 T 帧的 24 维统计特征，输出两个 4 维向量，一个告诉帧间跟踪“惯性还剩几分”，一个告诉系统“要不要按下重置键”。三个头的精确规格——输入尺寸、输出含义、损失函数——全部写在 `training/common/head_specs.py`，那是规格的**单一事实来源**：AI 说的、你记的，都以它为准。

**项目连线：一组人吃两种网络，正好把两条课线走通**

temporal_state 是时序网络，第 9 讲循环神经网络的“按时间展开”就是它的骨架，第 10 讲的门控记忆告诉你怎么“该记的记、该忘的忘”；feature_uncertainty 与 descriptor_softening 是 CNN，第 5 讲的卷积与特征图、第 6 讲的训练技巧全覆盖。还有个巧合值得会心一笑：两个图像头的输入恰好是 32×32 patch——正是第 6 讲 LeNet 的原生输入尺寸。你组等于把“卷积 + 时序”两条课线各落地一遍。

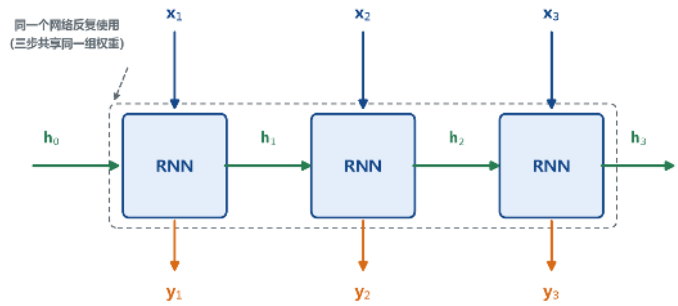



**图 2.1**　temporal_state 头的骨架：同一模块按时间步展开，隐藏状态把“最近发生了什么”一路传下去——第 9 讲的知识直接上岗

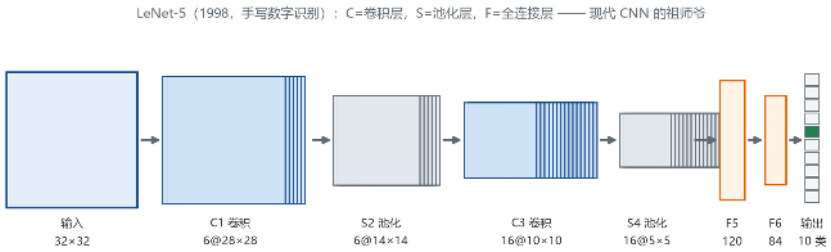



**图 2.2**　两个图像头的参照物 LeNet-5：输入恰好是 32×32——你的 patch 头与经典 CNN 同尺寸起步

## 2.4　第 19 周周五提交清单

提交线：2027年 1 月 22 日（第 19 周周五）完全提交离场。你的组交齐下表六样。

**表 2.3**　第 2 组提交清单（2027 年 1 月 22 日）

| 交付物 | 格式 | 说明 |
|---|---|---|
| 三头模型与指标 | 6 个文件 | 三个头各一套 `.onnx` + 同名 `.metrics.json`；文件名 = 头名，拷入 `stitcher_cpp/build/Release/models/`，交付后探针须仍 `PASS (12/12)` |
| 规格说明书 | 一页 | 三个头的输入尺寸/输出含义/损失函数，以 `head_specs.py` 为准（实验一产物，随报告附上） |
| 图像头设计文档 | 每头一页 | feature_uncertainty 与 descriptor_softening 的网络设计（实验三产物） |
| 训练证据包 | zip | 四次实验各一包，命令 + 日志 + 产物三件套 |
| A/B 对比报告 | PDF，2–4 页 | `ab_compare.py` 的 before/after 数据表与结论 |
| 组报告与 AI 声明 | PDF 8–15 页 + 一页 | 按第 三 章模板；AI 使用声明逐条列清 |

## 2.5　AI 分工与人工必做

分工原则一句话：AI出草稿，人出签字——签字的人对数字负全责。

表 2.4　第 2 组 AI 分工与人工必做

| AI可以自动做 | 人必须亲自做 |
|---|---|
| 解读 `head_specs.py` 三头规格并起草说明书 | 逐条核对规格（与代码逐行对照后才能写进文档） |
| 解释报错、起草设计文档与报告段落 | 本机跑通编译、`--smoke`、训练、探针、A/B |
| 写时序滑窗与画曲线的脚本草稿 | 亲眼看训练曲线、抽检 patch 样本 |
| 复述 RNN/CNN 原理、翻译术语 | 拷贝交付物并重跑探针；在证据包上签字担责 |

## 2.6　防糊弄指南与满分标准

**防人工糊弄指南（第 2 组版）**

**(1)** 图像头数据没到位的那几周，验收标准白纸黑字是“链路通 + 设计成文”——照实写就是满分答案；反过来，在报告里虚构训练精度 = 该节点成绩作废，这是你们组最容易踩的雷。**(2)** 跑一次成功就截图当十次的证据，不行；改日志时间戳 = 作弊。**(3)** 组内“代跑”必须留运行者姓名与时间。**(4)** 时序数据“先打乱再训练”是隐形造假——打乱的应该是滑窗样本，不是帧，打错了训练集验证集就串了，指标再高也是假的。

**防 AI 糊弄指南（第 2 组版）**

**(1) AI 念张量形状经常出错：**把 [B,1,32,32] 念成“批量 13232”它做得出来——每个维度什么含义，以 `head_specs.py` 为准。**(2) AI 给的数字必须本机复跑：**loss、指标，只有你终端里的那份是真的。**(3) AI 说“应该可以跑通”不等于跑通了，**跑没跑通只看你的终端输出。**(4)** 禁止把 AI 的回答直接粘进实验记录与规格说明书——说明书里的每句话都要能在代码里指出出处。

成果满分 50 分，怎么拿每一分见下表。

表 2.5　第 2 组成果满分标准（50 分制）

| 分值 | 项目 | 怎么拿满 |
|---|---|---|
| 20 | 功能达成 | 实验三交付 temporal_state 并完成 A/B（8 分）；实验四交付两个图像头（7 分）；每次交付后探针仍 `PASS (12/12)`（5 分） |
| 10 | 证据完整 | 四次实验证据包齐全可复现，含规格说明书与两页设计文档；缺一件扣 5 分 |
| 8 | 报告质量 | 组报告按模板 8–15 页；规格说明书与设计文档被助教判定“能照着实现” |
| 7 | 协作与台账 | 排班执行有记录；组 5 双台账中本组条目无缺失；求助记录规范 |
| 5 | 加分 | 如：时序窗口长度 T 的消融对比有数据；给软化头设计新的评测角度并用 A/B 验证 |

## 2.7　四次实验安排

四个节点与全班同频，每场按“目标—步骤—验收证据—常见坑”执行。先看总览表，再逐场展开。

表 2.6　第 2 组四次实验总览

| 节点 | 日期 | 本组动作 | 核心验收证据 |
|---|---|---|---|
| 实验一 | 10月 12 日 | agent 解读 head_specs.py，写规格说明书 | 说明书一页 + 核对记录 |
| 实验二 | 11 月 2 日 | 本机部署 + 探针 + 占位模型对照 | 三段日志 + wired 值对照 |
| 实验三 | 11 月 23 日 | temporal_state 训练交付；图像头公开 patch 先行训练 + 交设计 | 训练证据 + 两页设计文档 |
| 实验四 | 12 月 21 日 | 真机 patch 联调：图像头迁移转真交付 | 两头交付 + 公开版/真机版 A/B |

### 2.7.1　实验一（10 月 12 日）：读懂规格书，写出规格书

**目标**：用大模型 agent 解读 training/common/head_specs.py 中本组三个头的规格，人手留下一页“规格说明书”——它是你们组后面三个多月随查随用的字典。

**步骤**：(1) 装好编程助手并切到 agent 模式；(2) 打开 head_specs.py，让 AI 逐头回答三个问题：输入张量每一维是什么、输出的每个数什么含义、用什么损失函数；(3) 对照代码逐条核对，标“对 / 错 / 存疑”，至少纠正 AI 一处错误；(4) 汇成一页规格说明书。

**验收证据**：一页规格说明书 + AI 解读与人工核对记录。

**常见坑**：AI 把 [B,T,24] 的 T 说成“固定值”而不解释滑窗怎么来；[B,4]×2 是“两个输出”不是“八维向量”，AI 混着说的不少；提示词不点名文件路径，它就去背教科书——每个问题都把 head_specs.py 写进提示词。

### 2.7.2　实验二（11 月 2 日）：本机部署，外加一次“占位模型”对照

**目标**：部署流程与第 1 组一致——编译、看九头上线、跑探针；你们组多做一步：装一遍占位模型，验证“换模型”这件事本身干净无副作用。

**步骤**：(1) VS2022 打开 stitcher_cpp/build/360stitcher.sln，切 Release 编译；(2) 运行 360stitcher.exe --benchmark-stitch=deploy/simulation --provider=onnx，看到九行 [Provider]（形如 wired: orb=1500）；(3) python tools/nine_head_probe.py 期望 PASS (12/12)；(4) 运行 python tools/make_placeholder_models.py 生成九个占位模型，重跑同一基准，对照装前装后的 wired 值——应当**完全一致**（占位模型是输出恒定的空壳，它不该改变任何行为）。

**验收证据**：编译与九行 Provider 与探针三段日志；装占位模型前、后两份 wired 值对照记录。

**常见坑**：占位模型生成后忘了重跑探针；wired 值只截了装完的、没截装前的——对照实验两头都要留证据；忘了切 Release 报错找不到 dll。

### 2.7.3　实验三（11 月 23 日）：temporal_state 真训练交付，图像头交设计

**目标**：交付 temporal_state 头；两个图像头用 HPatches/MegaDepth 公开 patch 先行训练（真机 patch 未到位——如实注明数据来源，不冒充真机指标）。

**步骤**：(1) temporal_state：先 python train.py --smoke 验证链路，再 --telemetry 真数据训练，交付.onnx + .metrics.json 到 build/Release/models/，重跑探针；(2) 两个图像头：各交一页设计文档（网络几层、为什么这么设计、损失函数、打算怎么验证），并用 HPatches/MegaDepth 公开 patch 先

行训练一版（交付前仍跑 `python train.py --smoke` 验证链路）——**不等真数据，公开数据先跑通**，报告里如实标注训练数据来源。

**验收证据**：temporal_state 的训练日志、`metrics.json`、探针 PASS、A/B 结果；两页设计文档；两段 `--smoke` 完整日志。

**常见坑**：时序数据“先打乱再切分”——正确做法是先滑窗成样本再切训练/验证集，打乱的是窗口不是帧；图像头开局就在训练——公开 patch 先行训练与设计文档正是这几周的作业，真数据一到就能迁移。

### 2.7.4　实验四（12 月 21 日）：真机 patch 联调，图像头迁移转真

**目标**：总体组真机 patch 数据到位（合并联调）：把两个图像头从“公开 patch 预训版”迁移成“真数据版”，并做公开版/真机版 A/B 对账。

**步骤**：(1) 先抽检 20 张 patch 肉眼过目——确认是有效样本而不是空图与重复图；(2) 两个头先后 `--smoke`、`--telemetry` 训练；(3) 交付`.onnx` + `.metrics.json` 并重跑探针仍 `PASS (12/12)`；(4) 各跑 A/B：`python tools/ab_compare.py before.json after.json`。

**验收证据**：两个头的训练日志、`metrics.json`、探针 PASS、A/B 结果。

**常见坑**：数据刚到位就全量训练——先抽检再放量；样本量小先跑 `--smoke` 确认 batch 维度对得上，别到真训练才发现形状不匹配。

# 第 3 组　六目拼接 · 缝合融合组（九头之三：缝合与融合）

本章地图

**讲什么**：第 3 组任务书——缝合线、增益补偿、融合三处“老师傅手艺”怎么换成网络，以及为什么 fusion 头是必须压轴的最后一块拼图。**为什么值得读**：你们组要读懂的代码从 Python 换成了 C++，还要管一次全项目最复杂的上游依赖（fusion 头要等其余八个头全部上岗才能开训）。**学完你能**：读懂 `stitch_initialize.cpp` 里的补偿与融合逻辑，交付 gain_field 与 fusion 两个头，为 seam 头画好图纸。

一句话概括你们组：流水线“最后两步”的质检与手感——缝合线**走哪**（seam_uncertainty 给动态规划递眼色）、画面亮度**怎么拉齐**（gain_field 接管补偿）、两幅画面**怎么混**（fusion 定权重）。前两步的老师傅手艺，这学期开始一件件换成学出来的参数。

## 3.1　组档案

**表 3.1**　第 3 组档案

| | |
|---|---|
| 项目 | 六目拼接（项目一，360Stitcher实时全景系统，详见第 一 章） |
| 组号与组名 | 第 3 组 • 缝合融合组 |
| 人数与投入 | 5 人；每周合计 6–8 小时（组内自行排班，建议每周固定 2 次） |
| 负责的头 | seam_uncertainty、gain_field、fusion |
| 上游依赖 | 开局即可开工：gain_field/seam 用组 4 组公开仿真标签；fusion 彩排用公开数据，正式版等八头齐 + 总体组第 13 周前后放行 |
| 下游依赖 | 第 5 组：三个头交付后指标入双台账 |
| 工作目录 | `stitcher_cpp/training/` 下本组三个头的子目录；实验一解读对象是 `stitcher_cpp/src/stitch/stitch_initialize.cpp` |
| 硬件要求 | 本机无 GPU 亦可：三个头都是小网络，CPU 训得动 |

你们组的排班是“两头开局就跑、压轴先彩排”：gain_field 与 seam_uncertainty 用公开仿真标签第 1 周就能训练；fusion 虽然正式版要等八头齐 + 总体组放行（约第 13 周），但放行前用公开数据把训练管线与超参先彩排一遍——放行当天就能开工。

## 3.2　项目背景与你的位置

第 一 章讲过五步曲的后两步：第四步用动态规划在重叠区里找一条差异最小的**缝合线**，第五步用增益补偿加多频段/feather 融合把接缝抹到看不见——这两步至今是写死在 C++ 里的手艺：缝合代价怎么算、增益系数怎么调、过渡带多宽，全是工程师凭经验定的数。你们的任务，就是让这三处由数据说话。你在流水线里的位置：全系统的**出口段**——匹配靠不靠得住由第 1 组打分，特征靠不靠

得住由第 2 组把关，而观众最终看到的画面好不好，接缝穿不穿帮、亮度匀不匀，都从你们手里过。三个头按下表分工。




**图 3.1**　全景拼接五步曲上的第 3 组：seam_uncertainty 给第四步的缝合线“递眼色”，gain_field 与 fusion 接管第五步的增益补偿与融合——五步曲的最后两步归你

## 3.3　核心任务

**表 3.2**　第 3 组三个参数头：输入 → 输出与人话职责

| 头名 | 形状（输入 → 输出） | 一句话职责 | 网络类型 |
|---|---|---|---|
| seam_uncertainty | [B,2,128,256] → [B,1,128,256] | 逐像素标注重叠区里“哪里走缝容易穿帮”，给动态规划的代价图加上学习式权重 | CNN |
| gain_field | [B,12] → [B,6] | 从 12 个亮度统计量算 6 个增益系数，接管手写补偿逻辑 | 小回归网络 |
| fusion | [B,12] → [B,3] | 从 12 项统计量算 3 个融合权重，决定多频段与 feather 怎么混 | 小回归网络 |

逐头展开。**seam_uncertainty** 是三个头里唯一的“图像进、图像出”：输入重叠带（128×256 的区域，2 通道，具体含义以规格与代码为准），输出一张同尺寸的“穿帮风险图”——动态规划找缝合线时照着它绕道，行人、车灯这些高频穿帮点被自动避开。它和第 8 讲的 FCN 是同一思路：逐像素预测。**gain_field** 与 **fusion** 是两只“统计量进、几个数出”的小回归头，与第 1 组的头是同一物种，只是吃的统计量不同：前者吃 12 个亮度统计量（各路画面的均值方差与重叠区亮度差等），后者吃 12 项融合相关统计量。**fusion 还有个特殊身份：九头链路的最后一块拼图**——它的输入统计量要算在“其余八个头都上岗”的新系统上；头一换，统计量的分布就变，所以 fusion 的**正式版**必须等八头齐、总体组第 13 周前后放行后才能开训。但放行前允许用公开数据**彩排**：把训练管线、超参、验收动作先跑顺，彩排版标注“公开数据彩排”、不进台账成绩——区分彩排与抢跑的标就是标注与用途，不是做不做。

**项目连线：出口段的三个头与第一篇的哪几讲**

seam_uncertainty 是逐像素预测，第 5–6 讲的卷积网络打底，第 8 讲 FCN 的“热图输出”就是它的样子；gain_field 与 fusion 是第 1–3 讲的小回归头，方法论的根在第 3 讲：留验证集、看曲线、防过拟合。而“fusion 等八头”这件事本身，是第 3 讲“数据分布要一致”原则的工程版：训练数据长什么样，上线时就该长什么样。

## 3.4　第 19 周周五提交清单

提交线：2027年 1 月 22 日（第 19 周周五）完全提交离场。你的组交齐下表六样。

表 3.3　第 3 组提交清单（2027 年 1 月 22 日）

| 交付物 | 格式 | 说明 |
|---|---|---|
| 三头模型与指标 | 6 个文件 | 三个头各一套 `.onnx` + 同名 `.metrics.json`；文件名 = 头名，拷入 `stitcher_cpp/build/Release/models/`，交付后探针须仍 `PASS (12/12)` |
| seam 头设计文档 | 一页 | 网络结构与依据、损失函数、验证方案（实验三产物） |
| 训练证据包 | zip | 四次实验各一包，命令 + 日志 + 产物三件套；实验二的基线计时一并入包 |
| A/B 对比报告 | PDF，2–4 页 | `ab_compare.py` 的 before/after 数据表与结论，覆盖 gain_field 与 fusion |
| 组报告与 AI 声明 | PDF 8–15 页 + 一页 | 按第 三 章模板；AI 使用声明逐条列清 |

## 3.5　AI 分工与人工必做

分工原则一句话：AI出草稿，人出签字——签字的人对数字负全责。

表 3.4　第 3 组 AI 分工与人工必做

| AI可以自动做 | 人必须亲自做 |
|---|---|
| 解读 `stitch_initialize.cpp` 的补偿与融合段落 | 本机跑通编译、基准、训练、探针、A/B |
| 起草 seam 头设计文档与报告段落 | 核对 AI 对 C++ 的每句解释（落到行号才算数） |
| 写日志解析与计时画图脚本草稿 | 亲眼记录 luminance/blend 基线计时数字 |
| 解释报错、复述 DP 与融合原理 | 拷贝交付物并重跑探针；在证据包上签字担责 |

## 3.6　防糊弄指南与满分标准

**防人工糊弄指南（第 3 组版）**

**(1)** 实验二的基线计时（luminance、blend）是实验四 A/B 的“before”，**必须至少跑三遍取稳定读数**，只跑一遍就当基线 = 埋雷。**(2) 区分 fusion 的“彩排”与“正式”**：放行前用公开数据彩排不仅允许、还鼓励，但彩排版必须标注“公开数据彩排”且不得冒充正式版交付——把彩排版当正式版交，按未完成处理。**(3)** 跑一次截图当十次证据不行；改日志时间戳 = 作弊；组内“代跑”必须留运行者姓名。**(4) 编造训练精度 = 该节点成绩作废**——seam 头没数据就照实写“链路通 + 设计成文”，写个编出来的精度反而全丢。

**防 AI 糊弄指南（第 3 组版）**

**(1) AI 读 C++ 爱脑补**：变量名、行号、模板参数张口就来——核对时要它引用行号，你打开文件对上才算数。**(2) AI 给的数字必须本机复跑**：计时、指标，只有你终端里的那份是真的。**(3) AI 说“应该可以跑通”不等于跑通了**，跑没跑通只看你的终端输出。**(4)** 禁止把 AI 的回答

直接粘进实验记录与设计文档——文档里每句结论都要有代码行号或日志页码撑腰。

成果满分 50 分，怎么拿每一分见下表。

**表 3.5**　第 3 组成果满分标准（50 分制）

| 分值 | 项目 | 怎么拿满 |
|---|---|---|
| 20 | 功能达成 | 实验三交付 gain_field（公开仿真标签版）并完成 A/B（8 分）；实验四放行后交付 fusion 正式版，附彩排版与正式版两版记录（7 分）；每次交付后探针仍 `PASS (12/12)`（5 分） |
| 10 | 证据完整 | 四次实验证据包齐全可复现，基线计时三遍记录在包；缺一件扣 5 分 |
| 8 | 报告质量 | 组报告按模板 8–15 页；seam 设计文档达到“能照着实现”的水平 |
| 7 | 协作与台账 | 排班执行有记录；对第 1 组、第 2 组交付进度的跟踪记录完整；组 5 台账无缺失 |
| 5 | 加分 | 如：seam 设计文档被总体组采纳进实现；对 fusion 做八头联动分析并给出数据 |

## 3.7　四次实验安排

四个节点与全班同频，每场按“目标—步骤—验收证据—常见坑”执行。先看总览表，再逐场展开。

**表 3.6**　第 3 组四次实验总览

| 节点 | 日期 | 本组动作 | 核心验收证据 |
|---|---|---|---|
| 实验一 | 10月 12 日 | agent 解读 `stitch_initialize.cpp` 增益补偿段 | 解读 + 核对记录 + 参数清单 |
| 实验二 | 11 月 2 日 | 本机部署 + 探针 + 记录基线计时 | 三段日志 + 三遍基线表 |
| 实验三 | 11 月 23 日 | gain_field 训练交付 + A/B；seam 交设计 + `--smoke` | 训练证据 + 设计文档 |
| 实验四 | 12 月 21 日 | fusion 正式版：八头放行后用真实输出训练交付（联调） | 上游确认 + 彩排/正式两版记录 |

### 3.7.1　实验一（10 月 12 日）：让大模型替你读 C++，你来当裁判

**目标：**用大模型 agent 解读 `src/stitch/stitch_initialize.cpp` 的增益补偿段落，回答三个问题——为什么亮度不均、现在的补偿逻辑是什么、哪些是写死的数。你们的改造对象，今天先看个通透。

**步骤：**(1) 装好编程助手并切到 agent 模式；(2) 让 AI 定位增益补偿段落，回答三问：为什么各路画面亮度不均（曝光、增益、朝向差异）、现在的补偿按什么逻辑算系数、哪些常数是拍脑袋写死的；(3) 逐条核对——让 AI 引用行号，你打开文件对上才算数；(4) 把“写死的数”列成一张清单，它们就是 gain_field 头未来要接管的对象。

**验收证据：**一页“AI 解读 + 人工核对”记录，含行号引用、三问答案、写死参数清单。

**常见坑**：C++ 的模板与宏最容易让 AI 看走眼，别让它把标定参数当成运行时补偿系数；提示词不贴文件路径，它就去讲教科书式的“增益补偿综述”；只问一遍就信——换个问法再问一遍，答案打架的地方翻代码。

### 3.7.2　实验二（11 月 2 日）：本机部署，顺手记下基线数字

**目标**：部署流程与第 1 组一致；你们组多做一件只有你们组用得上的事——把日志里增益补偿与融合两段的计时基线记下来。

**步骤**：(1) VS2022 打开 `stitcher_cpp/build/360stitcher.sln`，切 `Release` 编译；(2) 运行 `360stitcher.exe --benchmark-stitch=deploy/simulation --provider=onnx`，确认九行 `[Provider]`（形如 `wired: orb=1500`）；(3) `python tools/nine_head_probe.py` 期望 `PASS (12/12)`；(4) 从日志里找出增益补偿段（luminance 计时行）与融合段（blend 计时行）的数字，同一基准**至少跑三遍**，记下稳定读数——这是你实验三、四 A/B 的 before。

**验收证据**：编译、九行 Provider、探针三段日志；三遍基线计时记录表。

**常见坑**：日志级别没开，计时行压根不打印——先确认日志配置；只跑一遍就当基线；忘了切 Release。

### 3.7.3　实验三（11 月 23 日）：gain_field 真训练交付，seam 头交设计

**目标**：交付 gain_field 头并用 A/B 说话；seam_uncertainty 头此时以“链路通 + 设计成文”为验收标准。

**步骤**：(1) gain_field：先 `python train.py --smoke` 验证链路，再 `--telemetry` 真数据训练，交付 `.onnx` + `.metrics.json` 到 `build/Release/models/`，重跑探针仍 `PASS (12/12)`；(2) A/B：`python tools/ab_compare.py before.json after.json`，与实验二记的基线对照；(3) seam_uncertainty：交一页设计文档（网络几层、为什么、损失函数、怎么验证），并用随机数数据跑 `python train.py --smoke` 证明全链路通。

**验收证据**：gain_field 训练日志、`metrics.json`、探针 PASS、A/B 结果；seam 设计文档与 `--smoke` 日志。

**常见坑**：A/B 文件路径写成相对路径而命令行不在对应目录，工具找不到文件；seam 头等数据闲着——设计文档与 `--smoke` 就是这几周的正经作业；`--smoke` 用完的随机数据目录留着别删，那是“链路通”的证据。

### 3.7.4　实验四（12 月 21 日）：八头齐，fusion 正式版压轴上场（合并联调）

**目标**：总体组放行后，用其余八个头的真实输出启动 fusion 正式训练并交付；本场同时是八组合并联调——真机标签接入、彩排版与正式版对账，给九头链路装上最后一块拼图。

**步骤**：(1) 放行前先做“上游验收”：确认第 1 组、第 2 组的头已交付且探针 `PASS (12/12)`，把确认记录写进证据包；(2) 总体组放行后，采集八头在岗时的新统计量作为训练数据；(3) `python train.py --smoke` 后 `--telemetry` 训练 fusion，交付并重跑探针；(4) A/B：用实验二基线与 fusion 上岗后的结果做对比，注意对比的是“八头在岗、fusion 换新”这一件事。

**验收证据**：上游确认记录、放行后的训练日志、`metrics.json`、探针 PASS、A/B 结果。

**常见坑**：等放行的几周是彩排期——彩排训练与上游验收清单都在这几周完成，放行当天就能开工；放行时间若推迟，与总体组确认新时间并留书面记录；把彩排版冒充正式版交付，或八头没齐就把旧系统 fusion 当新版交，均按无效处理。

# 第 4 组　六目拼接 · 数据与标签组

**本章地图**

**讲什么**：第 4 组任务书——全项目唯一“喂所有人”的组：遥测怎么变成三段标签、两条硬验收线（各 200 行、各 500 行）怎么达成、数据质量报告怎么写。**为什么值得读**：第 1–3 组的模型上限，从你们交出标签的那天起就定了——garbage in, garbage out，你们的手就是全项目的天花板。**学完你能**：独立跑通遥测校验与标签生成全流程，组织一次真机六路采集，写一份能让人信服的数据质量报告。

一句话概括你们组：别的组训模型，你们组造模型——九个参数头学得好不好，先看你们喂的标签对不对、够不够。八个小节点名最不起眼、分量最重的一环。

## 4.1　组档案

**表 4.1**　第 4 组档案

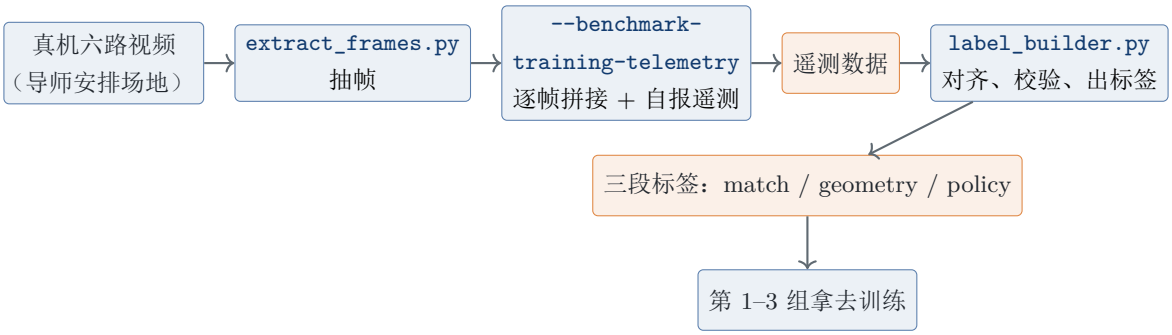
| | |
|---|---|
| 项目 | 六目拼接（项目一，360Stitcher实时全景系统，详见第 一 章） |
| 组号与组名 | 第 4 组 • 数据与标签组 |
| 人数与投入 | 5 人；每周合计 6–8 小时（组内自行排班；实验三需随导师赴真机场地，另计） |
| 负责的产出 | match / geometry / policy 三段标签 + 数据质量报告（不训练模型，喂所有人训练） |
| 上游依赖 | 开局任务就续：公开图对（UDIS-D 拼接对、COCO 单应图对）+ 系统自带 simulation 管线，第一周就能产仿真标签；真机场地录制由导师安排（实验三出行） |
| 下游依赖 | 第 1–3 组第一周就吃你的公开仿真标签；真机标签实验四接入；第 5 组按数据量台账验收 |
| 工作目录 | `stitcher_cpp/training_data/` 与 `stitcher_cpp/tools/` |
| 硬件要求 | 本机无 GPU 亦可；本组不做模型训练，瓶颈在组织与细心 |

## 4.2　项目背景与你的位置

第 一 章展示过系统运行时的遥测：终端每帧滚出一行 `[Latency]` 日志，系统逐帧“自报家门”。这些自报的家门，加上匹配与几何的中间统计量，就是全项目最宝贵的原料——**把“系统跑的时候自己说了什么”变成“网络学的标准答案”**，就是你们组的全部工作。你在流水线里的位置：最上游。**match 段标签喂第 1 组的 mc 与 policy 头，geometry 段标签喂 geometry 头，policy 段标签喂策略训练**；第 5 组的数据量台账，记的就是你交的每一批。公开仿真标签让三个组第一周就能开训——真机标签决定的是精度上限，不再是开工日期；但标签的真实性依旧是生命线：交得虚一线，三个组的精度集体塌方。

## 4.3　核心任务

你的产品是三段标签，生产线是一条数据流：




三段标签各管一头：**match 段**记录特征匹配的统计量与结果，是 mc、policy 头的口粮；**geometry 段**记录几何统计量与参数，喂 geometry 头；**policy 段**记录策略相关字段。每行一个样本，**行数就是数据量，字段质量就是标签质量**。课程给两条硬验收线：实验三 **match / geometry 各 $\geq$ 200 行并过校验**，实验四扩量到**各 $\geq$ 500 行**外加一份数据质量报告。注意下图的遥测实录——那行 `[Latency]` 就是你们原料的模样，以后你看到它该像厨师看到食材。

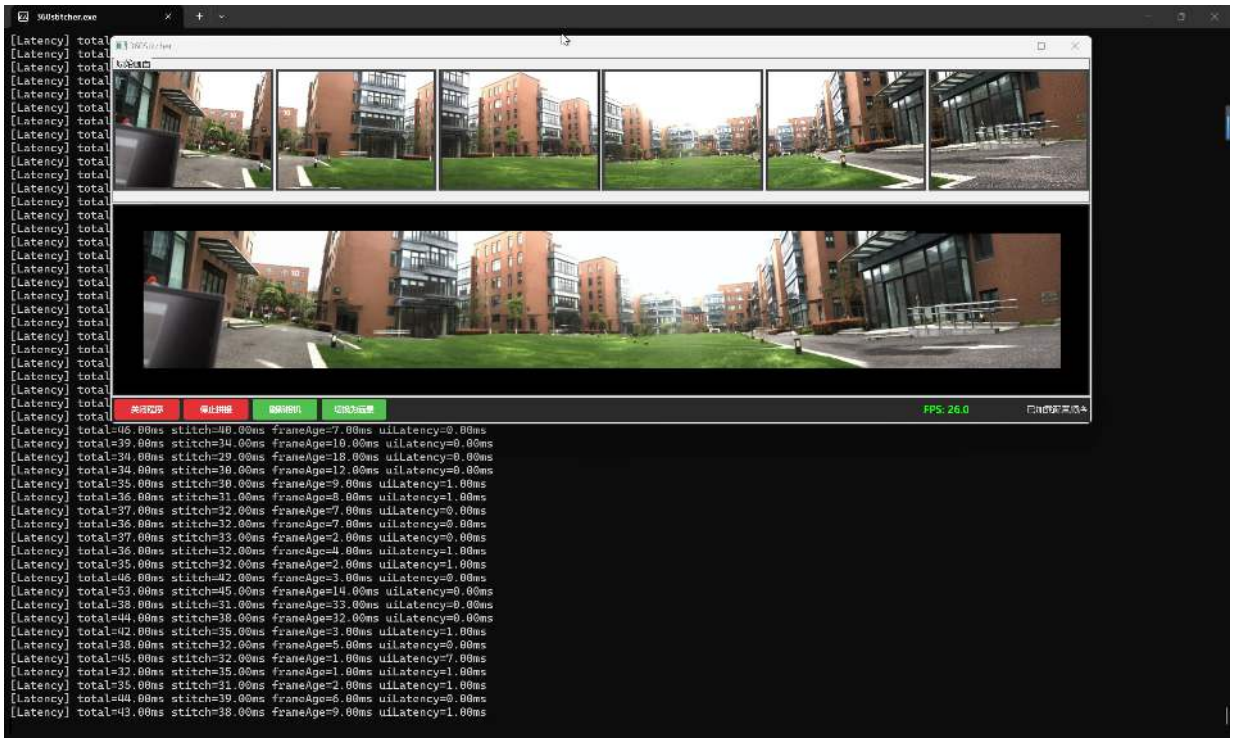



**图 4.1**　你们的原料长这样：系统运行遥测实录，终端每帧打印一行 [Latency] 日志（total / stitch / frameAge / uiLatency）——label_builder.py 的上游就是这股数据流

图片来源：360Stitcher 项目组真实运行截图（遥测终端实录）

**项目连线：数据组与第一篇的哪几讲**

第 6 讲“数据增强与训练技巧”里有半章在讲数据质量与分布——那正是你们的主战场：训练/验证怎么切、异常样本怎么处理、分布偏不偏，全是这一讲的方法论。第 3 讲的评估思想也站在你这边：校验器（`check_telemetry.py`）就是你给上游数据做的“验证集体检”。而“数据决定上限”这句话的出处，是第 2 讲就埋下的伏笔——反向传播只能从数据里学规律，数据里没有的，

它变不出来。

## 4.4　第 19 周周五提交清单

提交线：2027年 1 月 22 日（第 19 周周五）完全提交离场。你的组交齐下表六样。

表 4.2　第 4 组提交清单（2027 年 1 月 22 日）

| 交付物 | 格式 | 说明 |
|---|---|---|
| 三段标签数据集 | CSV（`training_data/`） | match / geometry 各 ≥ 500 行且过校验；policy 段按任务要求 |
| 数据流图与脚本说明 | 一页图 + README | “遥测 → 标签”数据流图（实验一产物）与复现步骤 |
| 数据质量报告 | PDF，4–8 页 | 字段分布、异常样本清单与处理依据、训练/验证划分建议 |
| 采集证据包 | zip | 真机视频清单、抽帧统计、遥测校验 `RESULT: PASS`、标签校验日志 |
| 组报告与 AI 声明 | PDF 8–15 页 + 一页 | 按第 三 章模板；AI 使用声明逐条列清 |

## 4.5　AI 分工与人工必做

分工原则一句话：AI出草稿，人出签字——签字的人对数字负全责。

表 4.3　第 4 组 AI 分工与人工必做

| AI可以自动做 | 人必须亲自做 |
|---|---|
| 解读 `label_builder.py` 三段标签的字段含义 | 核对 AI 的解释（逐字段对上代码才算数） |
| 起草数据流图与 README、写画分布图的脚本 | 本机跑通校验、抽帧、遥测、标签全流程 |
| 起草质量报告的文字段落 | 亲眼抽检样本、亲手核对行数与校验结果 |
| 解释报错、复述遥测字段含义 | 现场采集组织；在证据包上签字担责 |

## 4.6　防糊弄指南与满分标准

**防人工糊弄指南（第 4 组版）**

**(1) 行数凑数 = 掺假：**复制已有行、同一帧连抽二十张凑行数，数据量台账一查同源重复就露馅，按如实条款该节点成绩作废。**(2)** 校验不过就手改 CSV——校验器 `check_telemetry.py` 就是为防这个而存在的，改数据不如改流程。**(3)** 现场录制不留场景与时间记录，数据不可追溯 = 白录。**(4)** 异常样本一删了之不留记录，质量报告里每一个被剔除的样本都要有名有姓有理由；编造数据 = 该节点成绩作废，全课程无弹性。

**防 AI 糊弄指南（第 4 组版）**

**(1) AI 数行数会数错：**行数、字段数、分布统计，一律以脚本输出为准，AI 口算不算。**(2) AI 画的“分布图”必须由真实 CSV 生成**——它编的直方图漂亮得毫无意义。**(3) AI 说“应该可以跑通”不等于跑通了**，校验器报 `RESULT: PASS` 才算。**(4)** 禁止把 AI 对字段含义的解释直接写

进报告——每个字段以 `label_builder.py` 代码为准。

成果满分 50 分，怎么拿每一分见下表。

表 4.4　第 4 组成果满分标准（50 分制）

| 分值 | 项目 | 怎么拿满 |
|---|---|---|
| 20 | 功能达成 | 实验三达成验收线：match / geometry 各 ≥ 200 行并过校验（10 分）；实验四扩量至各 ≥ 500 行且质量报告合格（10 分） |
| 10 | 证据完整 | 采集证据包齐全可追溯:视频清单、抽帧统计、两道校验的 PASS 日志；缺一件扣 5 分 |
| 8 | 报告质量 | 组报告按模板 8–15 页；质量报告字段分布讲得清、异常处理有据 |
| 7 | 协作与台账 | 排班与现场分工有记录；向第 1–3 组交付有交接记录；组 5 数据量台账无缺失 |
| 5 | 加分 | 如：质量报告发现并追溯出遥测或标签管线的 bug 且被总体组采纳修复；给下游提供“按场景切片”的标签使用建议 |

## 4.7　四次实验安排

四个节点与全班同频，每场按“目标—步骤—验收证据—常见坑”执行。先看总览表，再逐场展开。

表 4.5　第 4 组四次实验总览

| 节点 | 日期 | 本组动作 | 核心验收证据 |
|---|---|---|---|
| 实验一 | 10月 12 日 | agent 解读 `label_builder.py`，画数据流图 | 解读 + 核对记录 + 流程图 |
| 实验二 | 11 月 2 日 | 基准图走通“遥测 → 三段标签”全流程 | `RESULT: PASS` + 标签文件 |
| 实验三 | 11 月 23 日 | 真机采集 → 抽帧 → 遥测 → 标签 | 各 ≥ 200 行并过校验 |
| 实验四 | 12 月 21 日 | 扩量 ≥ 500 行 + 数据质量报告 | 校验日志 + 质量报告 |

### 4.7.1　实验一（10 月 12 日）：读懂标签生产线，画出数据流图

**目标**：用大模型 agent 解读 `tools/label_builder.py` 中三段标签（match / geometry / policy）的数据来源与字段含义，画出“遥测 → 标签”数据流图——你以后的每一步操作，都建立在这张图上。

**步骤**：(1) 装好编程助手并切到 agent 模式；(2) 让 AI 逐段回答：三段标签各从哪股遥测来、每列字段什么含义、校验逻辑查什么；(3) 逐字段与代码对照核对，标“对 / 错 / 存疑”，至少纠正 AI 一处；(4) 画“遥测 → 标签”数据流图，工具不限（手绘拍照也行），交图。

**验收证据**：一页“AI 解读 + 人工核对”记录 + 一张数据流图。

**常见坑**：AI 对字段含义张口就来——让它引用代码行号；三段标签的字段有同名不同义的情况，AI 最爱混着讲；图只画了主流程没画校验环节——校验是生产线的质检站，必须上图。

### 4.7.2　实验二（11 月 2 日）：先在基准图上跑通全流程

**目标**：真机采集之前，先用 `deploy/simulation` 的五张基准图把“遥测 → 三段标签”全流程跑通；再批量喂入 UDIS-D 公开拼接对，产出公开仿真标签交给第 1–3 组——他们的开工口粮就在你这场实验里。

**步骤**：(1)遥测体检：`python scripts/check_telemetry.py` 后接你的遥测目录参数（如 `training_data/simulation_run1`），期望输出 `RESULT: PASS`；(2) 命令行运行 `360stitcher.exe --benchmark-training-telemetry`（部署流程见第 1 组实验二：VS2022 切 `Release` 编译 `360stitcher.sln`），对五张基准图逐帧拼接并记录遥测；(3) 运行 `label_builder.py` 生成三段标签，跑一遍标签校验，确认行数与字段齐全。

**验收证据**：遥测校验 `RESULT: PASS` 日志、基准图跑遥测的完整日志、三段标签文件与校验输出。

**常见坑**：遥测目录路径写错，校验器查了个空目录也“不报错”——先看它统计了多少帧再下结论；标签生成后没跑校验就往下传——校验 5 秒钟，能省下游一晚上；五张基准图验证流程；随后批量喂 UDIS-D 图对（至少 50 对），产出公开仿真标签并过校验，match/geometry 各不少于 200 行——这是第一份交付物，第 1–3 组第一周开训就靠它。真机标签的行数指标留给实验三。

### 4.7.3　实验三（11 月 23 日）：真机场地采集，交出第一条验收线

**目标**：随导师到真机场地录制六路视频，走完“抽帧 → 遥测 → 标签”全流程，达成 **match / geometry 各 ≥ 200 行并过校验**。

**步骤**：(1) 随导师赴真机场地录制六路视频，逐段记录场景与时间；(2) 回校后 `python tools/extract_frames.py` 抽帧，统计抽帧数量；(3) 运行 `--benchmark-training-telemetry` 对抽出的帧逐帧拼接、记录遥测；(4) `python scripts/check_telemetry.py` 体检遥测，期望 `RESULT: PASS`；(5) `label_builder.py` 生成三段标签并过校验，核对行数达标。

**验收证据**：视频清单（场景/时长）、抽帧统计、遥测校验 PASS、三段标签行数与校验通过日志。

**常见坑**：现场忘了记每段视频的场景与时间——标签要能追溯，回头对不上就是白录；六路相机没同步，时间戳对不齐，遥测对不上帧，标签全废；抽帧太密全是相似帧——行数够了但“有效多样性”不够，校验会卡。

### 4.7.4　实验四（12 月 21 日）：扩量与质检，交出最终数据家底

**目标**：真机标签扩量到**各 ≥ 500 行**并全部过校验，交数据质量报告；本场同时是八组合并联调：真机标签接入第 1–3 组复训、与公开仿真标签做对比报告（公开版 vs 真机版：指标差异与原因）。

**步骤**：(1) 补充采集或补充抽帧，扩量到验收线以上，全程留证；(2) 逐字段做分布统计，画分布图；(3) 清点异常样本（缺失字段、离谱值、同源重复），逐条记录处理方式与依据；(4) 写数据质量报告：字段分布、异常样本处理、训练/验证划分建议；(5) 向第 1–3 组交接，并在组 5 数据量台账登记。

**验收证据**：真机标签 500 行以上的校验通过日志、字段分布图、异常样本处理清单、质量报告 PDF、台账登记；另附公开仿真标签与真机标签的对比说明。

**常见坑**：为凑行数复制样本——台账查同源重复一查一个准；异常样本静默删除不留记录——质量报告必须有名有姓有理由；分布图让 AI “随手画一张”——每一张图都要能从真实 CSV 一条命令重新生成。

# 第 5 组　六目拼接 · 测试集与评测组

本章地图

**讲什么**：第 5 组任务书——测试集怎么用哈希“冻结”、基线指标台账怎么记、A/B 与回归评测怎么跑、组间评分怎么汇成离场验收会那张打分表。**为什么值得读**：八个组里唯一“给所有人打分”的组——双台账是全项目“节点验收与成绩评定唯一依据”，而台账在你手里。**学完你能**：亲手冻结一份不可抵赖的测试集，把两条技术路线都部署验收一遍，独立完成一次全绿回归，主持一场有数据支撑的组间评分。

一句话概括你们组：别的组**造**东西，你们组**验**东西——测什么、怎么测、分数怎么算，全项目听你们组的。这叫**数据生产与质量验收分离**：第 4 组生产数据，你们验收数据；各组交付模型，你们验收模型。裁判不下场踢球，但哨子必须过硬。

## 5.1　组档案

表 5.1　第 5 组档案

| | |
|---|---|
| 项目 | 六目拼接（项目一，360Stitcher实时全景系统，详见第 一 章） |
| 组号与组名 | 第 5 组 • 测试集与评测组 |
| 人数与投入 | 5 人；每周合计 6–8 小时（组内自行排班；交付季评测集中，建议交付周加开一次组会） |
| 负责的产出 | 测试集（冻结版）+ 哈希清单、基线与数据量双台账、A/B 与回归评测报告、组间评分汇总表 |
| 上游依赖 | 第 1–3 组与第 7 组：每一次交付都进你的台账；第 4 组：公开仿真场景（开局即有）+ 录制场景（实验四增补）是你的测试素材库 |
| 下游依赖 | 全体组：A/B 统一用你冻结的测试集；第 三 章离场验收会：你的组间评分汇总表是核心材料 |
| 工作目录 | `stitcher_cpp/tools/`（`ab_compare.py`、`run_regression.ps1`） |
| 硬件要求 | **两条路线都要能部署**：路线一 `360stitcher.exe` 与路线二 `360stitcher_e2e.exe` 都要能在你本机跑起来；无 GPU 亦可 |

代码基座两个目录都在你机器上：`stitcher_cpp`（VS2022 打开 `360stitcher.sln`，九头路线）与 `e2e_gpu`（端到端路线，自含程序与训练）。别的组可以只守一亩三分地，你必须两边都熟——两条路线的成绩，都从你手里过。

## 5.2　项目背景与你的位置

第 一 章讲过两条路线的赛马：九头学习路线用九个小参数头逐个接管经验参数，端到端路线用一个网络直接吃图像对、吐对齐变换。一个学期下来，谁进步了、该得多少分——不靠嗓门大，只靠**同一把尺子**：同一份冻结测试集、同一组指标、同一本台账。你在流水线里的位置是**终点站兼裁判席**：

九头交付跑九头探针、端到端交付跑引擎探针，指标进你的台账；两轮回归全绿由你执行；离场验收会那张组间评分汇总表，由你从台账里一笔一笔汇出来。下图就是你们天天打交道的被评对象——六路画面进、一条全景出，你的每一个数，都在给这条全景带的稳、快、好记账。

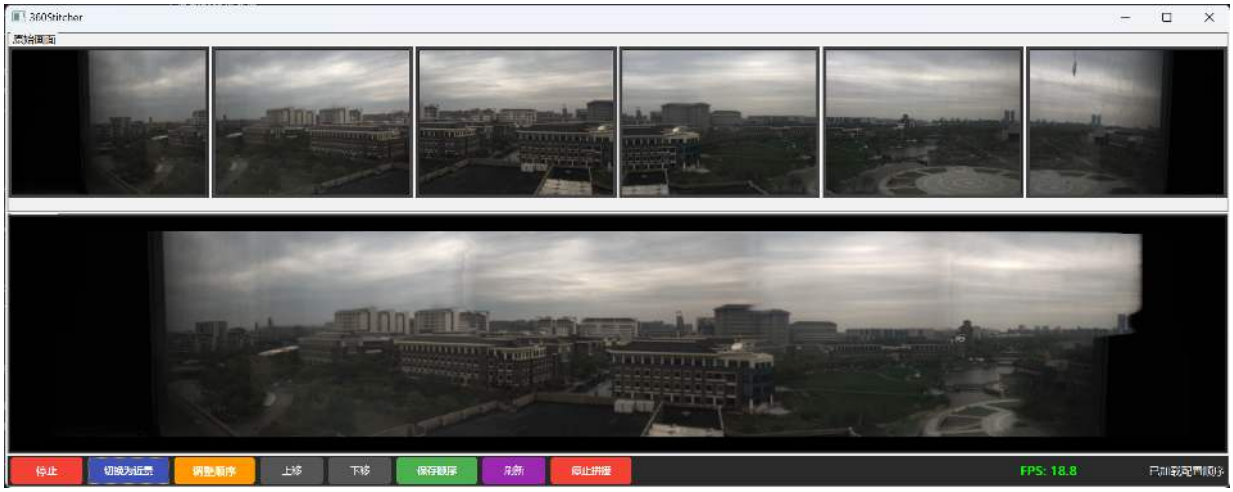



**图 5.1**　你们的被评对象长这样：360Stitcher 六路原始画面（上排）拼成一条 360° 全景（下排）——测试集里的每张图、台账里的每个数，最终都是在给这条全景带的稳、快、好记账

图片来源：360Stitcher 项目组真实运行截图

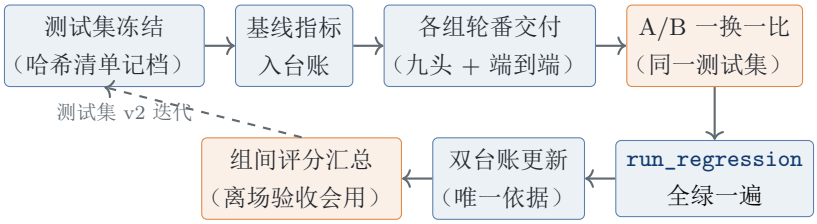
## 5.3　核心任务

四项职责一张表，随后说人话。

**表 5.2**　第 5 组四项职责：做什么、交什么、哪门课的血缘

| 职责 | 做什么 | 产出 | 血缘 |
|---|---|---|---|
| 测试集构建与冻结 | 选图（基准五图 + 录制场景抽帧）、生成 SHA-256 哈希清单、按难易分桶 | 测试集 v1 / v2 + 哈希清单 | 第 3 讲 |
| 基线指标台账 | 两条路线的初始化 / 稳态 / P95 指标、各组数据量、探针状态逐条入账 | 双台账 | 第 3 讲 |
| A/B 与回归评测 | `ab_compare.py` 一换一比；`run_regression.ps1` 全绿一遍 | A/B 报告、回归日志 | 第 6 讲 |
| 组间评分汇总 | 从台账汇成评分表，交叉复核后交离场验收会 | 评分汇总表 | 全项目 |

说人话版本：**测试集构建**是出考卷——考卷发下去就不能换题，每张图的“指纹”当场记档；**基线指标台账**是记账——各组每次交付的指标、数据量、探针状态逐行入账（数据量台账记第 4 组的产出，基线指标台账记各组的模型成绩），这本账是“节点验收与成绩评定唯一依据”；**A/B 与回归评测**是验货——A/B 是“只换一个模型、其余全不动”的控制变量对比，回归是“改完之后老功能还在不在”的例行体检；**组间评分汇总**是判卷——从台账汇总，不凭印象。闭环一圈一圈转，见下图。




**裁判的两件兵器：哈希与分桶**

**SHA-256 哈希**：给文件内容算一个 256 位的“指纹”，改一个字节就面目全非——冻结测试集 = 给每张图记指纹，谁换了图一查便知。PowerShell 一条命令：`Get-FileHash 文件名 -Algorithm SHA256`。**难度分桶**：测试样本按“易 / 中 / 难”分桶再报成绩——只报总平均会掩盖“人人都会”与“全军覆没”的差别；分桶之后，谁在难桶上掉分一目了然。

**项目连线：裁判的每件兵器都来自第一篇**

冻结测试集是第 3 讲“验证集”的工程化——训练组在验证集上防过拟合，你在冻结测试集上防“评测漂移”，方法论同一套；A/B 是控制变量实验，第 6 讲“一次只动一个因子”的对照思路直接搬用；台账是第 3 讲评估表格的加长版——把“准确率”换成“稳态毫秒”与“重投影误差”。你不需要发明新方法，只需把第一篇的方法用到全项目尺度。

## 5.4　第 19 周周五提交清单

提交线：2027年 1 月 22 日（第 19 周周五）完全提交离场。你的组交齐下表五样——离场验收会上，评委手里翻的就是前三样。

表 5.3　第 5 组提交清单（2027 年 1 月 22 日）

| 交付物 | 格式 | 说明 |
|---|---|---|
| 双台账终版 | xlsx / CSV | 数据量台账 + 基线指标台账，各组每次交付一条，可回溯到原始日志 |
| A/B 报告归档 | zip | 全部 before/after JSON 与对比表，按日期与组号编目 |
| 组间评分汇总表 | PDF + xlsx | 离场验收会用；每一个分数都能回溯到台账条目 |
| 测试集与哈希清单 | 目录 + 清单文件 | v1 冻结版 + v2 困难样本版；清单含每张图的 SHA-256 与分桶标注 |
| 组报告与 AI 声明 | PDF 8–15 页 + 一页 | 按第 三 章模板；AI 使用声明逐条列清 |

## 5.5　AI 分工与人工必做

分工原则一句话：AI出草稿，人出签字——签字的人对数字负全责。

表 5.4　第 5 组 AI 分工与人工必做

| AI可以自动做 | 人必须亲自做 |
|---|---|
| 解读 ab_compare.py 与 run_regression.ps1，起草评测指标说明书 | 本机部署两条路线，亲手点亮两个探针 |
| 写哈希清单与分桶统计的脚本草稿 | 亲手冻结测试集，抽三张复核哈希一致 |
| 起草 A/B 报告与评分表的表格框架 | 逐条核对 AI 结论（对着代码与日志才算数） |
| 解释报错、给排查建议 | 台账录入与评分复核；在账本上签字担责 |

求助纪律：组内 → 助教（研究生）→ 导师，求助必须附完整命令与完整输出。课程成绩建议构成：实验一不计分，实验二至四各 10 分共 30 分，平时作业 20 分（十二讲章后习题十二选二），项目成果 50 分，总评满分 100 分（教师可调）；无论怎么调，每次节点验收都以双台账为唯一依据——包括你们自己组。

## 5.6　防糊弄指南与满分标准

**防人工糊弄指南（第 5 组版）**

**(1) 改台账一个数 = 改全项目成绩**：台账是“节点验收与成绩评定唯一依据”，笔下每个数字都要能与原始日志对上；虚构或美化指标 = 该节点成绩作废，裁判组作假加倍严肃。**(2) 测试集不冻结就开测**：今天这五张、明天那五张，A/B 结论全作废——哈希清单是防“测试集漂移”的唯一手段，先冻结、后评测。**(3) 评分凭印象**：每个分数必须能回溯到一条台账记录。**(4) 回归没全绿就签字**：run_regression.ps1 全绿一遍才算数，红一项就如实记红并写明原因。

**防 AI 糊弄指南（第 5 组版）**

**(1) AI 背的“标准评测指标”不算数**：本项目测什么，以 ab_compare.py 的指标表与引擎日志为准。**(2) AI 口算的哈希一律不信**：哈希、均值、P95 必须由你本机命令生成并复核。**(3) AI 说“回归应该能全绿”不等于全绿**：只认你终端里的完整输出与退出码。**(4)** AI 起草的评分汇总表，每一格都要人对着台账核一遍再交——AI 录错一格，五个组的成绩跟着错。

成果满分 50 分，怎么拿每一分见下表。

表 5.5　第 5 组成果满分标准（50 分制）

| 分值 | 项目 | 怎么拿满 |
|---|---|---|
| 20 | 功能达成 | 实验二两条路线部署验收双双通过（8 分）；实验三测试集 v1 冻结 + 基线入账 + 回归全绿（7 分）；实验四台账 v2 全覆盖 + 困难样本测试集 v2（5 分） |
| 10 | 证据完整 | 四次实验证据包齐全：哈希清单、部署与探针日志、全绿回归日志、A/B 归档；缺一件扣 5 分 |
| 8 | 报告质量 | 组报告按模板 8–15 页；评测指标说明书讲得清，评分表可回溯、有交叉复核记录 |
| 7 | 协作与台账 | 排班有记录；各组交付登记无缺失、无迟到；评分汇总按时产出并用于离场验收会 |
| 5 | 加分 | 如：设计“易 / 中 / 难”分桶并给出各组在难桶上的差距分析；为回归脚本补充新检查项且被采纳 |

## 5.7　四次实验安排

四个节点与全班同频：实验一“大模型 agent 模式配置及应用”（2026 年 10 月 12 日）、实验二“匹配合适程序并执行本地部署”（11 月 2 日）、实验三“深度网络的模型训练实践”（11 月 23 日）、实验四“模型优化训练实践”（12 月 21 日）。每场按“目标—步骤—验收证据—常见坑”执行：步骤照抄就能跑，证据一栏就是成绩。

表 5.6　第 5 组四次实验总览

| 节点 | 日期 | 本组动作 | 核心验收证据 |
|---|---|---|---|
| 实验一 | 10月 12 日 | agent 解读两个评测工具，写指标说明书 | 说明书 + 核对记录一页 |
| 实验二 | 11 月 2 日 | 两条路线部署 + 双探针点亮 | PASS (12/12) 与 PASS (5/5) |
| 实验三 | 11 月 23 日 | 测试集 v1 冻结、基线入账、回归全绿 | 哈希清单 + 台账 + 全绿日志 |
| 实验四 | 12 月 21 日 | 台账 v2 + 困难样本测试集 v2 | 台账 v2 + v2 哈希清单 |

### 5.7.1　实验一（10 月 12 日）：让 agent 替你读评测工具，你来写说明书

**目标**：用大模型 agent 解读 `tools/ab_compare.py` 对比哪些指标、`tools/run_regression.ps1` 回归哪些环节，产出全组第一份武器：一页“评测指标说明书”。

**步骤**：(1) 装好编程助手并切到 agent 模式；(2) 让 AI 回答 `ab_compare.py` 三问：读的是哪两份文件、逐项列出对比的指标（初始化耗时、稳态均值、稳态 P95、输出宽高等）与各自“越高越好还是越低越好”、为什么强调“一次只换一个模型”；(3) 让 AI 拆解 `run_regression.ps1` 的环节链——配置、编译、单元测试、基准拼接、遥测——并回答“每一环失败会发生什么”；(4) 逐条与代码对照核对并至少纠正一处，成文一页“评测指标说明书”：每项指标的中文名、方向、进台账的格式。

**验收证据**：一页评测指标说明书 + “AI 解读 + 人工核对”记录。

**常见坑**：AI 把指标张冠李戴——让它逐项对代码里的指标列表报行号；把初始化耗时混进每帧耗时——加载只发生一次，稳态才是每帧的账；说明书只写指标名不写方向，两周后连自己都看不懂“这个数大了是好是坏”。

### 5.7.2　实验二（11 月 2 日）：两条路线都部署，两种探针都点亮

**目标**：裁判先当一次被评者——两条路线都在你本机部署并通过探针。以后各组交付，你验收的手感就来自今天。

**步骤**：(1)路线一：VS2022 打开 `stitcher_cpp/build/360stitcher.sln`，切 `Release` 整体生成；命令行进入 `build/Release` 运行 `360stitcher.exe --benchmark-stitch=deploy/simulation --provider=onnx`，日志应出现九行 `[Provider]` 加载记录；再运行 `python tools/nine_head_probe.py`，期望 `PASS (12/12)`。(2) 路线二：进入 `e2e_gpu`，运行 `bin/360stitcher_e2e.exe --benchmark-stitch=bin/simulation --engine=ropstitch`，日志应出现 `[Engine] loaded` 与 `engine active`；再运行 `python training/engine_probe_test.py`，期望 `PASS (5/5)`。

**验收证据**：两段部署日志 + 两个探针的完整输出（12/12 与 5/5），四样齐全。

**常见坑**：路线一不在 `build/Release` 目录下运行，程序找不到基准图；路线二的引擎探针会真实替换模型做实验——跑探针时别开着 `360stitcher_e2e.exe`，文件占用会让“交付还原”失败；两个探

针一个 12/12 一个 5/5，别背混——裁判先把自己手里的验收线背熟。

### 5.7.3　实验三（11 月 23 日）：冻结测试集 v1，基线入账，回归全绿

**目标**：“数据生产与质量验收分离”落地日——考卷 v1 提前冻结：基准五图 + 第 4 组公开仿真场景（UDIS-D 采样，开局即有）先行定版，录制场景到手后作为 v1.1 增补；基线记账、全项目第一轮回归跑绿。

**步骤**：(1) 选定测试集 v1：`deploy/simulation` 基准五图 + 首批录制场景抽帧，逐张生成 SHA-256 哈希清单并归档；(2) 在冻结测试集上跑两条路线基准，把初始化耗时、稳态均值、稳态 P95 记入台账基线栏；(3) 运行 `powershell -File tools/run_regression.ps1` 全绿一遍，留完整日志；(4) 台账登记本节点各组状态一览，向全班公示。

**验收证据**：测试集 v1 哈希清单、基线指标台账条目、`run_regression.ps1` 全绿日志，三样缺一不可。

**常见坑**：哈希没写，测试集就会漂移——两周后有人换了图你自己都发现不了，A/B 前后不可比；`run_regression.ps1` 的权限问题是新手第一坑——PowerShell 执行策略拦脚本是常态，按报错调整策略或用 `powershell -File` 方式运行，别管理员权限硬怼；把原始视频直接当测试集——先抽帧、写清单、再冻结。

### 5.7.4　实验四（12 月 21 日）：台账 v2 与困难样本，裁判的终场哨

**目标**：九头三组（第 1–3 组）与端到端组（第 7 组）全部交付完毕，台账升到 v2；再从录制素材里挑出“容易翻车”的样本，出第二版考卷。

**步骤**：(1) 逐行核对台账 v2：每个交付组每次交付的指标、A/B 结论、探针状态，查漏补缺；(2) 从录制场景人工挑困难样本（弱纹理、运动模糊、重叠区小、光照突变），按易 / 中 / 难分桶标注，冻结为测试集 v2 并写哈希清单；(3) 通知各组在 v2 上补测“难桶”成绩，入台账；(4) 汇总组间评分表初稿，组内交叉复核后定稿。

**验收证据**：台账 v2 全表、测试集 v2 哈希清单、各组难桶补测记录、评分汇总表定稿。

**常见坑**：困难样本只挑最难的、不挑有代表性的——分桶的意义在拉开区分度，全是地狱难度谁都零分；评分先斩后奏——先定评分规则并公示，再录分数；台账 v2 与 A/B 归档对不上号——每条指标都要能翻出对应的 before/after JSON。

# 第 6 组　六目拼接 · 端到端数据组

**本章地图**

**讲什么**：第 6 组任务书——端到端路线吃什么数据、32+8 对合成数据怎么一键生成、首批真实数据从组 4 的录制里怎么切出来、`build_dataset.py` 与 `benchmark.py` 两个工具怎么交付。**为什么值得读**：端到端模型的上限，从你交出数据集那天起就定了——第 7 组训得再努力，吃的全是你做的饭。**学完你能**：跑通“合成数据演练 → 真实数据实跑”的切换，写一份数据集卡片，交付能对任意 onnx 引擎算重投影误差与耗时的 benchmark 工具。

一句话概括你们组：九头路线吃“统计量 + 标签”（第 4 组把遥测变成 CSV 行），端到端路线吃**原始图像对 + 对齐变换标签**——后一碗饭归你做。合数据先“干跑”，真数据再“实跑”，切换的操作手就是你。

## 6.1　组档案

**表 6.1**　第 6 组档案

| | |
|---|---|
| 项目 | 六目拼接（项目一，360Stitcher实时全景系统，详见第 一 章） |
| 组号与组名 | 第 6 组 • 端到端数据组 |
| 人数与投入 | 5 人；每周合计 6–8 小时（组内自行排班，建议每周固定 2 次） |
| 负责的产出 | `build_dataset.py`（一键训练/验证清单）、`benchmark.py`（重投影误差与耗时 → `benchmark_report.json`）、合成 + 真实两代数据集与数据集卡片 |
| 上游依赖 | 开局即可构建：`make_demo_data.py` 合成对 + UDIS-D 公开拼接对（实验三的主粮）；组 4 组真机录制抽帧实验四接入 |
| 下游依赖 | 第 7 组吃你的 train/val 清单去训练（公开版 v1 实验三交付）；第 8 组吃你的 `benchmark.py` 测耗时与精度 |
| 工作目录 | `e2e_gpu/training/group10_e2e_data`（数据装载位 `e2e_gpu/training/datasets/`） |
| 硬件要求 | 本机无 GPU 亦可：造数据与算指标不吃 GPU，CPU 足够 |

目录名里的 `group10` 是源工程里端到端数据组的原始编号，与本课程“第 6 组”无关。两个提醒先立在此：其一，`datasets/` 的目录格式**写死**（按批次建夹，`left` 与 `right` 里**同名文件是一对**），以 `datasets/README.md` 为准；其二，读写图一律走 `io_cn.py` 的 `imread_cn`/`imwrite_cn`——OpenCV 原生函数遇到中文路径会静默失败，前人踩过的坑。

## 6.2　项目背景与你的位置

第 一 章讲过两条路线的赛马：九头路线是“系统老手带九个小学徒”，端到端路线是“一个学徒吃整对图像”——一次吃一对 512×512 的相邻相机画面，直接吐出把右图对齐到左图的 $3 \times 3$ 变

换矩阵。六目每采一帧可切 5 对相邻对（cam1+2、cam2+3、……、cam5+6），你的数据量单位是“对数”。你在流水线里的位置：端到端路线的**最上游**——合数据阶段把演练流程跑顺，真数据阶段从第 4 组的录制素材里切出训练对；“干跑到实跑”那一脚油门，由你踩。

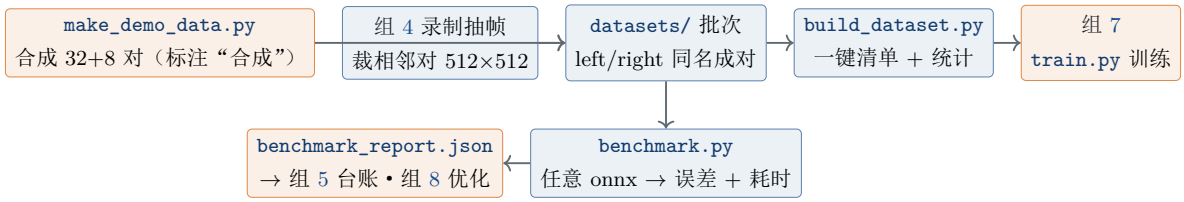
## 6.3　核心任务

两件工具、两代数据，一张表说清。

**表 6.2**　第 6 组核心交付：工具两件、数据两代

| 交付 | 要做到什么 | 谁在用 |
|---|---|---|
| `build_dataset.py` | 扫描 `datasets/` 全部批次，一键生成训练/验证清单与统计（对数、场景、划分比例） | 第 7 组 |
| `benchmark.py` | 对任意引擎 onnx：读测试图对，逐对推理，输出重投影误差与耗时，写 `benchmark_report.json` | 第 8 组、组 5 |
| 合成数据批次 | `make_demo_data.py` 生成 32+8 对（train 32 对、val 8 对，含已知平移真值），全程标注“合成” | 全流程演练 |
| 真实数据集 v1 → v2 | 首批 ≥ 50 对（来自组 4 录制抽帧的相邻对裁剪），实验四扩量并更新卡片 | 第 7 组 |

端到端数据的“标签”值得多嚼两口：合成对造的时候平移量是自己定的，真值天然已知；真实对没有现成答案，用传统拼接管线算出的变换当**伪真值**，再人工抽检兜底——与第 4 组“用系统遥测当标签”是同一种思想。模型学得好不好，用**重投影误差**来量（见下方知识盒）。整条流水线如下图：上游两个水龙头（合成、真实），下游两个用户（训练、评测）。




**重投影误差：对齐准不准的量尺**

把 right 图按模型预测的变换“拉”回 left 图的坐标系，公共内容应完全重合；逐对对应点算偏移再取平均，就是**重投影误差**（单位：像素），越小对齐越准。好比印刷的“套印误差”：红蓝两色没套准，边缘就发虚——误差就是那道虚边的宽度。第 8 组判断“量化后精度损没损”、你们判断“模型逐版有没有进步”，用的都是这把尺。

**零基础扫盲：数据集卡片四要素**

发布数据集要附一页“卡片”，四要素缺一不可：**来源**——谁、什么设备、什么场景，哪些是合成；**划分**——train/val怎么切、按什么原则（按场景切，不按帧切，防止“同一场景一半在训练一半在考试”）；**统计**——对数、场景数、分辨率、重叠率分布；**许可**——素材能用到什么范围（本课程项目内部使用）。

**项目连线：你的数据喂的是哪种网络**

端到端模型的输入是 512×512 图像对——第 5 讲卷积怎么“看”图，它就在看什么；第 4 讲“数据自己会说话”的思想，在这里变成“让图像对直接喂出变换”。扩量想加旋转、亮度扰动，第 6 讲的数据增强九宫格就是菜单。而“合成干跑、真实实跑”两步走，是机器人与自动驾驶领域的标准做法：仿真里先把流程磨顺，再上真数据——你这一学期走的正是这条路。

下图是“对齐变换”的直观样子：同一个平面在两张相邻视角照片里的四角，被一个 $3 \times 3$ 矩阵一一对应地搬过去——你的每对训练样本，学的就是这么一个搬运关系。

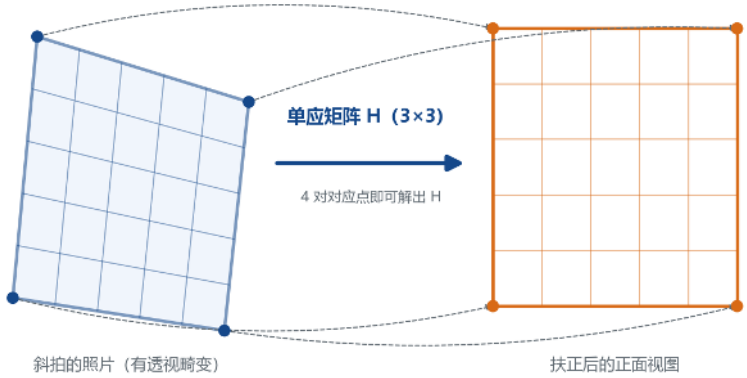



图 6.1　单应变换直观图：平面 A 的四角经一个 $3 \times 3$ 变换映射到平面 B——端到端模型每对输入要学的“搬运关系”，也是 `benchmark.py` 用重投影误差去量的对象

## 6.4　第 19 周周五提交清单

提交线：2027年 1 月 22 日（第 19 周周五）完全提交离场。你的组交齐下表五样。

表 6.3　第 6 组提交清单（2027 年 1 月 22 日）

| 交付物 | 格式 | 说明 |
|---|---|---|
| `build_dataset.py` + `benchmark.py` | py | 一键清单生成；对任意 onnx 输出 `benchmark_report.json`，README 写明命令与期望输出 |
| 端到端数据集 v2 | datasets 目录 + 清单 | 合成 32+8 对（标注“合成”）+ 真实 ≥ 150 对、≥ 3 场景 |
| 数据集卡片 | 一页 | 来源 / 划分 / 统计 / 许可四要素，v1 与 v2 各一版或合并说明 |
| benchmark 报告 | json + PDF 2–4 页 | `ropstitch.onnx` v1/v2 的重投影误差与耗时，均值与 P95 并列 |
| 组报告与 AI 声明 | PDF 8–15 页 + 一页 | 按第 三 章模板；AI 使用声明逐条列清 |

## 6.5　AI 分工与人工必做

分工原则一句话：AI出草稿，人出签字——签字的人对数字负全责。

表 6.4　第 6 组 AI 分工与人工必做

| AI可以自动做 | 人必须亲自做 |
|---|---|
| 解读 `datasets/README.md` 契约与 `make_demo_data.py` 逻辑 | 本机跑通合成、推理自测、清单、benchmark 全部命令 |
| 起草裁对脚本、统计图与数据集卡片的框架 | 亲手抽检真实对：同名成对、重叠区没被裁丢 |
| 起草两路线数据差异说明的表格框架 | 抽 10 对可视化逐对过目；在卡片与报告上签字担责 |
| 解释报错、给排查建议 | 与组 7/8 的交接当面确认，不靠转述 |

求助纪律：组内 → 助教（研究生）→ 导师，求助必须附完整命令与完整输出。课程成绩建议构成：实验一不计分，实验二至四各 10 分共 30 分，平时作业 20 分（十二讲章后习题十二选二），项目成果 50 分，总评满分 100 分（教师可调）；每次节点验收以组 5 维护的双台账为唯一依据——你的数据量，就是台账里记的那一行。

## 6.6　防糊弄指南与满分标准

**防人工糊弄指南（第 6 组版）**

**(1) 合成数据忘了标“合成”：**`demo` 批次是演练用的，混进真实数据台账就是数据事故——批次名、卡片、清单三处同标。**(2) 录屏当相机序列：**数据契约明文禁止——录屏里只有内嵌的 6 路小画面是真实相机画面。**(3) 对数凑数：**同一对图复制改名、同场景连抽五十张近似帧，台账查同源重复一查一个准；**虚构数据集统计 = 该节点成绩作废。(4) 考卷泄题：**`benchmark.py` 只准用 val 批次，训练集混进评测集，一切指标作废。

**防 AI 糊弄指南（第 6 组版）**

**(1) AI 数对数会数错：**对数、场景数、划分统计，一律以 `build_dataset.py` 的输出为准。**(2) AI 写的裁对脚本必须本机验证：**中文路径会不会静默失败、中心裁方会不会把重叠区裁丢，跑过才知道。**(3) AI 说“benchmark 应该能出数”不等于出数：**`benchmark_report.json` 落盘且字段齐全才算。**(4)** AI 起草的数据集卡片，来源与统计两栏逐条对真实目录核过再交。

成果满分 50 分，怎么拿每一分见下表。

表 6.5　第 6 组成果满分标准（50 分制）

| 分值 | 项目 | 怎么拿满 |
|---|---|---|
| 20 | 功能达成 | 实验二合成 32+8 对全流程跑通（5 分）；实验三真实 ≥ 50 对 + 数据集卡片合格（8 分）；实验四扩量 ≥ 150 对且 `benchmark.py` 对 `ropstitch.onnx` 出数（7 分） |
| 10 | 证据完整 | 四次实验证据包齐全：生成日志、推理自测可视化、清单统计、`benchmark_report.json`；缺一件扣 5 分 |
| 8 | 报告质量 | 组报告按模板 8–15 页；卡片四要素齐全，benchmark 报告均值与 P95 并列、结论有数据 |
| 7 | 协作与台账 | 向组 7/8 交接有记录；数据量按时入组 5 双台账；求助记录规范 |
| 5 | 加分 | 如：用数据增强扩量并用 A/B 证明不伤指标；产出“困难样本建议清单”交组 5 进难桶 |

## 6.7　四次实验安排

四个节点与全班同频：实验一“大模型 agent 模式配置及应用”（2026 年 10 月 12 日）、实验二“匹配合适程序并执行本地部署”（11 月 2 日）、实验三“深度网络的模型训练实践”（11 月 23 日）、实验四“模型优化训练实践”（12 月 21 日）。每场按“目标—步骤—验收证据—常见坑”执行。

表 6.6　第 6 组四次实验总览

| 节点 | 日期 | 本组动作 | 核心验收证据 |
|---|---|---|---|
| 实验一 | 10月 12 日 | agent 解读数据需求，写两路线差异说明 | 差异说明 + 核对记录一页 |
| 实验二 | 11 月 2 日 | 合成数据全流程：生成 + 推理自测 | 32+8 对 + `infer_overlay.jpg` |
| 实验三 | 11 月 23 日 | 首批真实数据 ≥ 50 对 + 卡片 | 清单统计 + 抽检可视化 |
| 实验四 | 12 月 21 日 | 扩量 + `benchmark.py` 初版 | `benchmark_report.json` |

### 6.7.1　实验一（10 月 12 日）：读懂端到端的数据餐单，写两路线差异说明

**目标**：用大模型 agent 解读端到端路线的训练数据需求——相邻相机对的图像对 + 对齐变换标签——并与九头路线的数据做对照，产出一页差异说明。

**步骤**：(1) 装好编程助手并切到 agent 模式；(2) 让 AI 读 `datasets/README.md` 与 `make_demo_data.py`，回答三问：一个训练样本长什么样（left/right 同名成对、512×512）、标签是什么（合成对的已知平移真值 / 真实对的伪真值变换）、训练与验证怎么分；(3) 让 AI 对照第 4 组的遥测标签，列“样本形态 / 标签来源 / 数据量单位 / 质量瓶颈”四行对照表；(4) 逐条与文件核对并至少纠正一处，成文一页。

**验收证据**：一页“两路线数据差异说明”+ AI 解读与人工核对记录。

**常见坑**：AI 说“端到端不需要标签”——合成对的平移真值是造数据时写死的，真实对也要伪真值标签，“不要标签”是它背错了课文；AI 把 left/right 讲成“左相机右相机”——口径是**对内前一路 / 后一路**；对照表写“数据不一样”级别的空话——每个格子都要落到具体文件与字段。

### 6.7.2　实验二（11 月 2 日）：合成数据全流程，32+8 对先干跑一遍

**目标**：真数据到位前，用合成对把“生成 → 训练口径 → 推理自测”整条流程跑顺——流程里的坑今天踩完，实验三就只剩数据问题。

**步骤**：(1)进入 `e2e_gpu/training`，`pip install -r requirements-cpu.txt`；(2) 运行 `python make_demo_data.py`，产出 `datasets/train/demo`（32 对）与 `datasets/val/demo`（8 对），含已知平移真值；(3) 运行 `python infer_test.py`（缺省用 val 第一对），到 `outputs/` 查看 `infer_overlay.jpg` 的叠加可视化，确认对齐合理；(4) 加餐：`python make_real_dataset.py` 生成 8 对真实测试批次（只放 val，评测用、不进训练）。

**验收证据**：32+8 对生成日志与目录清单、`infer_overlay.jpg`、真实测试批次的说明一句（来源与用途），全进证据包。

**常见坑**：清单或脚本路径含中文导致脚本崩——读写图一律用 `io_cn.py` 的 `imread_cn`/`imwrite_cn`；合成批次忘了标“合成”——批次名、清单、卡片三处同标；`infer_test.py` 报形状错误——模型输入是五维 `[1,2,3,512,512]`，脚本注释里有前车之鉴。

### 6.7.3　实验三（11 月 23 日）：首批公开数据集（UDIS-D），50 对起步

**目标**：从 UDIS-D 公开拼接对里切出首批训练数据——**≥ 50 对、覆盖 ≥ 2 个场景**——并发布数据集卡片（如实标注“公开数据版”）；真机录制抽帧实验四联调时增补为真实版。

**步骤**：(1) 取组 4 组录制视频的抽帧，六路同帧按相邻对切分（cam1+2、……、cam5+6，一帧切 5 对）；(2) 中心裁方 + 缩放至 512×512（对齐引擎输入口径），先算重叠区再下剪刀；(3) 按 `datasets/train/批名/left|right` 放置，**同名文件成对**；(4) 运行 `build_dataset.py` 出清单与统计，核对对数达标；(5) 写数据集卡片，抽 10 对出可视化逐对过目，交接组 7 。

**验收证据**：≥ 50 对清单与统计、数据集卡片一页、10 对抽检可视化、交接记录。

**常见坑**：left/right 同名不成对——文件名对不上一个字符，`train.py` 读出来就是张冠李戴的错配对；中心裁方把重叠区裁没了——重叠是标签的载体，裁丢等于白切；只在同一场景抽——50 对全长一个样，多样性不够，训练一上真场景就露馅。

### 6.7.4　实验四（12 月 21 日）：数据扩量与 benchmark 初版，双线收官

**目标**：真实数据集扩量到**≥ 150对、≥ 3 个场景**并更新卡片；交付 `benchmark.py` 初版——对当前 `ropstitch.onnx` 输出重投影误差与耗时。

**步骤**：(1) 扩量：补切批次或补场景（用第 6 讲的增强手段时，增强对单独标注来源），更新清单与卡片至 v2；(2) 写 `benchmark.py`：输入测试图对目录 + 任意 onnx 路径，逐对推理、算重投影误差与耗时，写 `benchmark_report.json`（含逐对明细与汇总均值/P95）；(3) 对当前 `ropstitch.onnx` 跑一遍出报告；(4) 结论登记组 5 台账，交接组 8 作优化基线。

**验收证据**：扩量统计与 v2 卡片、`benchmark_report.json`、`benchmark.py` 的 README（命令与期望输出）。

**常见坑**：benchmark 只报均值不报分布——P95 与最差对也要有，优化组的提升空间全藏在尾部；评测批次混入训练批次——考卷泄题，指标再好也作废；onnx 输入尺寸写死 512——从模型读输入形状或做成参数，下一版模型一换就崩。

# 第 7 组　六目拼接 · 端到端训练组

**本章地图**

**讲什么**：第 7 组任务书——端到端模型的结构与损失怎么读、CPU 冒烟怎么做、全书唯一一次 200 epoch 真训练怎么在 RTX 4080 上完成、v1/v2 两版怎么逐版交付。**为什么值得读**：你是唯一把“深度网络的模型训练实践”四个字全部走到实处的组——真 GPU、真 200 epoch、真交付上系统。**学完你能**：独立跑通“训练 → 导出 onnx → 一键交付 → 探针验收”全链路，管好自己的 GPU 时段，用 A/B 数据证明每一版都在进步。

一句话概括你们组：第 6 组把饭做好，你们组**吃饭长个子**——把端到端模型从随机初始化一路训到能上岗，交付给系统替换旧大脑。全书唯一需要 GPU 的组，唯一的 200 epoch，都在你手里。

## 7.1　组档案

**表 7.1**　第 7 组档案

| | |
|---|---|
| 项目 | 六目拼接（项目一，360Stitcher实时全景系统，详见第 一 章） |
| 组号与组名 | 第 7 组 • 端到端训练组 |
| 人数与投入 | 5 人；每周合计 6–8 小时（组内自行排班；实验三按 GPU 时段另排） |
| 负责的产出 | `ropstitch.onnx` v1/v2 + `ropstitch.metrics.json`、两次真训练完整日志、A/B 对比报告 |
| 上游依赖 | 开局即可训练：组 6 组的合成对与 UDIS-D 公开数据集（实验三的主粮）；真机数据实验四续训；组 5 冻结测试集是 A/B 考场 |
| 下游依赖 | 第 8 组拿你的 onnx 做推理优化；每次交付入组 5 双台账 |
| 工作目录 | `e2e_gpu/training/`（本组任务夹 `group11_e2e_train`；`train.py`、`export_onnx.py`、`deliver_model.bat` 在 `training/` 根目录） |
| 硬件要求 | **RTX 4080 机器**（训练用）；交付与验证走 CPU 推理，回本机即可 |

目录名里的 `group11` 是源工程里端到端训练组的原始编号，与本课程“第 7 组”无关。先立一条**特别纪律**：4080 机器只归第 7 组（第 8 组可申请时段）与导师使用——用前预约登记，用时守时段，训练完立刻把 `exports/` 与日志拷回本机。模型交付的形状契约别背错：输入 `[B,2,3,512,512]`（一对 512×512 图）、输出 `[B,3,3]`（一个变换矩阵）、opset 13——`export_onnx.py` 会按契约自检。

## 7.2　项目背景与你的位置

第 一 章讲过端到端路线的野心：不再让九个小参数头逐个学经验参数，而是一个网络直接看一对相邻画面、吐出把右图对齐到左图的 $3 \times 3$ 变换；六目每帧拆 5 个相邻对逐对推理，链式累积后由渲染管线合成全景。你在流水线里的位置：端到端路线的**发动机**——第 6 组的数据喂进来，你把模型训出来、导成 onnx、经 `deliver_model.bat` 交付进 `bin/models/`，再由引擎探针证明“系统真的在用你

的模型”。你的每一个版本都会被组 5 的台账与 A/B 记录在案：**重投影误差逐版不升**，是你给自己的军令状。

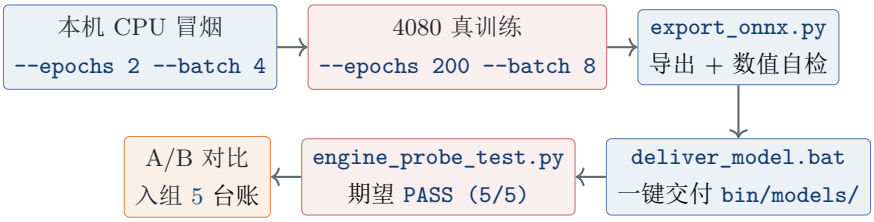
## 7.3　核心任务

两个版本、一条流水线，一张表加一张图说清。

**表 7.2**　第 7 组核心交付：两个版本一台流水线

| 版本 | 要做到什么 | 数据 |
|---|---|---|
| 冒烟版 | `python train.py --epochs 2 --batch 4` 本机 CPU 跑通全链路，验证代码、数据、交付、探针四处无堵点 | 合成 32+8 对 |
| v1 | 4080 上 `--epochs 200 --batch 8` 真训练，导出交付，A/B 证明优于基线（公开数据版） | 组 6 UDIS-D 公开数据集 ≥ 50 对 |
| v2 | 真机数据微调（从 v1 续训、调小学习率；合并联调收官），A/B 证明**逐版不升**，不升则如如实回退 v1 | 组 6 真机数据（联调日接入） |

整条流水线如下图：本机冒烟排除低级错误，4080 出真模型，导出自检守契约，一键交付换大脑，探针验收防绕过，A/B 入账定成绩——六步串成一圈，每转一圈出一个新版本。




**训练三旋钮与一个保险：epoch / batch / 学习率 + 断点续训**

**epoch**：全部训练数据过一遍网络算一个 epoch；**batch**：一次一起送进网络的样本数；这两个数加**学习率**（每步挪多大），就是第 3 讲的三大超参数：epoch 太少没学饱、太多会过拟合，batch 与学习率互相牵制。200 epoch 是长跑，中途断电、超时都可能有——`training/checkpoints/` 里的 `.pth` 就是保险：从上次存档续训，不用从头再来。每次训练的超参数与最终 `val_loss` 记成表格，这是你实验报告的骨架。

网络骨干带着**残差连接**——第 7 讲“让信息走捷径”的短路设计（下图），梯度可经捷径直通浅层，深网络才训得动；损失函数是**重投影误差**：用预测变换把 right 拉回 left，逐像素比对偏移。结构细节以 `train.py` 代码为准，实验一你要亲手把它读出来。

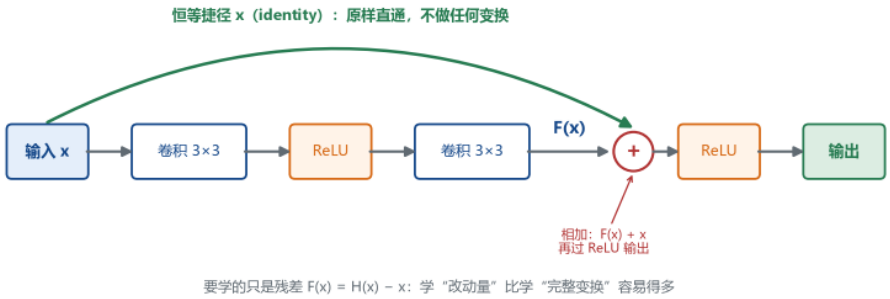



图 7.1　残差块：输入经两层卷积得到 $F(x)$，再与恒等捷径相加——你模型骨干里的“信息捷径”，第 7 讲的知识直接上岗

**项目连线：200 epoch 把第一篇串成一条线**

`train.py` 里的训练循环，就是第 6 讲那张“取 batch → 前向 → 算 loss → 反向 → 更新”的流程环原样运转；loss 降不动、会过拟合，按第 3 讲的曲线方法论盯着验证集说话；骨干里的残差连接出自第 7 讲；导出 onnx 后与 PyTorch 输出的数值一致性自检（误差 $< 10^{-3}$），是第 0 讲“浮点数不完全相等”的实战回响。一门课的知识在这一章拧成一股绳。

## 7.4　第 19 周周五提交清单

提交线：2027年 1 月 22 日（第 19 周周五）完全提交离场。你的组交齐下表五样。

表 7.3　第 7 组提交清单（2027 年 1 月 22 日）

| 交付物 | 格式 | 说明 |
|---|---|---|
| `ropstitch.onnx` v2 + `ropstitch.metrics.json` | onnx + json | 经 `deliver_model.bat` 交付、`bin/models/` 在位；交付后探针仍 `PASS (5/5)` |
| 训练证据包 | zip | 冒烟、v1（200 epoch）、v2（微调）三段完整日志，含运行者与起止时间；超参数记录表 |
| A/B 对比报告 | PDF，2–4 页 | 基线 / v1 / v2 同测试集重投影误差对比，“逐版不升”或如实回退的结论 |
| 数据流图与训练记录 | 一页图 + 表 | 实验一“图像对 → 变换估计”数据流图 + 各版本超参数与 `val_loss` 台账 |
| 组报告与 AI 声明 | PDF 8–15 页 + 一页 | 按第 三 章模板；AI 使用声明逐条列清 |

## 7.5　AI 分工与人工必做

分工原则一句话：AI出草稿，人出签字——签字的人对数字负全责。

表 7.4　第 7 组 AI 分工与人工必做

| AI可以自动做 | 人必须亲自做 |
|---|---|
| 解读 `train.py` 结构与损失函数，起草数据流图 | 本机跑通冒烟、导出、交付、探针全部命令 |
| 解读报错、给续训与调参建议 | 4080 上亲手启动训练、盯 loss 曲线、守时段 |
| 画 loss 曲线、整理超参数表格的脚本 | 亲手核对 `metrics.json` 每个字段与日志对账 |
| 起草 A/B 报告文字段落 | 在训练证据包上签字，对“本人运行”担责 |

求助纪律：组内 → 助教（研究生）→ 导师，求助附完整命令与完整输出。课程成绩建议构成：实验一不计分，实验二至四各 10 分共 30 分，平时作业 20 分（十二讲章后习题十二选二），项目成果 50 分，总评满分 100 分（教师可调）；每次节点验收以组 5 双台账为唯一依据——你的 `metrics.json`，就是台账里记的那一行。

## 7.6　防糊弄指南与满分标准

**防人工糊弄指南（第 7 组版）**

**(1) 铁律原文，一字不改：交付后程序输出的相机位置必须随模型内容变化（探针原理），训练指标必须来自本人运行**——探针会放大模型权重再跑一遍，输出不变就说明系统没用你的模型；日志没有你的账号与时间，就不算你训的。**(2)** 冒烟指标当真训练成绩报——`epochs_done: 2` 的 `metrics.json` 是流程验证，不是模型成绩。**(3)** v2 不如 v1 就悄悄只交 v1——如实条款：回退 v1 并写明原因同样是合格结论，**选择性上报按虚构处理**。**(4)** GPU 时段超时霸机、替组 8 代跑占用——时段登记白纸黑字，违规记录进台账。

**防 AI 糊弄指南（第 7 组版）**

**(1) AI 报的 loss 与 `val_loss` 一律不信：**只有你终端里滚出来的那份是真的，AI 现编的曲线一张都不许进报告。**(2) AI 讲网络结构爱“脑补”：**几层卷积、哪里有残差、损失怎么算，以 `train.py` 代码为准，让它逐行引用行号。**(3) AI 说“这个超参数肯定更好”不等于更好：**调参结论只认同测试集 A/B 数据。**(4)** AI 生成的训练曲线必须能从真实日志一条命令重画——画不出原始日志的那条，就是装饰品。

成果满分 50 分，怎么拿每一分见下表。

**表 7.5**　第 7 组成果满分标准（50 分制）

| 分值 | 项目 | 怎么拿满 |
|---|---|---|
| 20 | 功能达成 | 实验二 CPU 冒烟全链路 + 探针 `PASS (5/5)`（5 分）；实验三 4080 真训练交付 v1 且 A/B 优于基线（10 分）；实验四 v2 微调完成并给出对比结论或如实回退（5 分） |
| 10 | 证据完整 | 三段训练日志、`metrics.json`、交付后探针 PASS、A/B 报告，缺一件扣 5 分 |
| 8 | 报告质量 | 组报告按模板 8–15 页；训练曲线、超参数表、A/B 表齐全，结论有数据支撑 |
| 7 | 协作与台账 | GPU 时段预约与使用登记无违规；交付按时入组 5 台账；与组 6/8 交接有记录 |
| 5 | 加分 | 如：学习率 / batch 小实验（2–3 组对比曲线有结论）；对损失函数做改进且 A/B 有证据 |

## 7.7　四次实验安排

四个节点与全班同频：实验一“大模型 agent 模式配置及应用”（2026 年 10 月 12 日）、实验二“匹配合适程序并执行本地部署”（11 月 2 日）、实验三“深度网络的模型训练实践”（11 月 23 日）、实验四“模型优化训练实践”（12 月 21 日）。每场按“目标—步骤—验收证据—常见坑”执行。

表 7.6　第 7 组四次实验总览

| 节点 | 日期 | 本组动作 | 核心验收证据 |
|---|---|---|---|
| 实验一 | 10月 12 日 | agent 解读结构与损失，画数据流图 | 解读 + 核对记录 + 数据流图 |
| 实验二 | 11 月 2 日 | 本机 CPU 冒烟全链路 + 探针 | `PASS (5/5)` + 三段日志 |
| 实验三 | 11 月 23 日 | 4080 真训练 200 epoch，交付 v1 并 A/B | 训练日志 + `metrics.json` + A/B |
| 实验四 | 12 月 21 日 | 全量真实数据微调 v2，逐版对比 | `metrics.json` + v1/v2 对比表 |

### 7.7.1　实验一（10 月 12 日）：读懂你将要训练的东西，画出数据流

**目标**：用大模型 agent 解读端到端模型的网络结构与损失函数（重投影误差），画出“图像对 → 变换估计”的数据流图——先知道要训什么，再谈怎么训。

**步骤**：(1) 装好编程助手并切到 agent 模式；(2) 让 AI 读 `training/train.py`，回答三问：网络骨架长什么样（几层、卷积核多大、残差连接在哪）、输入输出什么形状、损失函数怎么定义（重投影误差如何逐像素算）；(3) 让 AI 画出“图像对 → 变换估计”数据流：左右图 → 骨干提特征 → 融合 → 回归 $3 \times 3$ 矩阵 → 重投影误差；(4) 逐条与代码对照核对，标“对 / 错 / 存疑”，至少纠正一处。

**验收证据**：一页“AI 解读 + 人工核对”记录 + 一张数据流图（手绘拍照亦可）。

**常见坑**：AI 把它讲成“分类网络”——输出是 9 个数的变换矩阵，不是类别得分；损失函数名张口就来——让它引用 `train.py` 行号，代码为准；数据流图漏掉预处理（缩放 512、归一化）——预处理不对，后面全错。

### 7.7.2　实验二（11 月 2 日）：本机 CPU 冒烟，三十分钟走通全链路

**目标**：不上 GPU，先把“训练 → 导出 → 交付 → 探针”四个环节在本机 CPU 上全走通——冒烟 5 分钟，救你半天 GPU 时段。

**步骤**：(1) 进入 `e2e_gpu/training`，`pip install -r requirements-cpu.txt`；(2) `python train.py --epochs 2 --batch 4`——CPU 几分钟，产出 `checkpoints/ropstitch.pth`；(3) `python export_onnx.py`——导出 `exports/ropstitch.onnx`，数值自检误差应远小于 $10^{-3}$；(4) 双击 `deliver_model.bat`——正本拷入 `bin/models/`，见 `[OK]`；(5) `python engine_probe_test.py`，期望 `RESULT: PASS (5/5)`。

**验收证据**：冒烟训练日志、导出自检输出、交付 `[OK]` 提示、探针 `PASS (5/5)`，四样齐全（`epochs_done: 2` 会如实写进 `metrics.json`——它就是冒烟的痕迹）。

**常见坑**：跳过冒烟直接上 4080——环境没装对、数据格式不对，半天时段烧在低级错误上；本机误装 `requirements-gpu.txt`——没有 CUDA 立刻报错，CPU 冒烟装 CPU 版；双击 `deliver_model.bat` 时 `360stitcher_e2e.exe` 还开着——文件占用拷贝失败，关程序再交付。

### 7.7.3　实验三（11 月 23 日）：4080 真训练，全书唯一的 200 epoch

**目标**：在 RTX 4080 上完成 `--epochs 200 --batch 8` 真训练，导出交付 v1，用 A/B 证明它优于人工参数基线。

**步骤**：(1) 按预约时段上 4080（纪律：只归第 7 组、第 8 组可申请时段、导师；用前登记、用后拷回）；(2) 整个 `e2e_gpu` 拷到 4080，`pip install -r requirements-gpu.txt`，`python check_gpu.py` 确

认识别到 GPU；(3) 组 6 的 UDIS-D 公开数据集放入 `datasets/`（按数据集卡片核对），先 `--epochs 2` 跑一遍确认无堵点，随后 `python train.py --epochs 200 --batch 8`——每半小时看一眼 loss，发散立刻停；(4) 训完 `python export_onnx.py`，把 `exports/` 与完整日志拷回本机；(5) 双击 `deliver_model.bat` 交付 v1，重跑探针须 `PASS (5/5)`，再在组 5 冻结测试集上 A/B：基线 vs v1 的重投影误差。

**验收证据**：200 epoch完整训练日志（含运行者与起止时间）、`ropstitch.metrics.json`、交付后探针 `PASS (5/5)`、基线 vs v1 的 A/B 对比。

**常见坑**：时段超时——先测单 epoch 耗时再决定本次时段训多少，`checkpoints/` 断点续训是保险不是摆设；中断后从头重训——续训从存档走，从头重来等于白烧电；loss 发散不肯停——学习率太大就早停排查，跑完 200 个 epoch 的废模型更浪费；日志残缺——没有完整日志，训练等于没发生过。

### 7.7.4　实验四（12 月 21 日）：全量真实数据微调，v2 收官

**目标**：真机数据到位（八组合并联调），从公开版 v1 续训微调出真实场景版 v2，用同测试集 A/B 回答“真实数据换掉公开数据，模型变好了吗”。

**步骤**：(1) 从 `checkpoints/` 的 v1 存档续训，`--epochs 50 --batch 8` 量级、学习率调小（微调不是推倒重来）；(2) 训完导出 v2，交付、重跑探针须 `PASS (5/5)`；(3) 同一冻结测试集 A/B：基线 / v1 / v2 三列重投影误差对比表，写不少于 100 字结论；(4) v2 若不如 v1，如实记录并回退交付 v1——“数据多了不一定赢”本身就是有价值的结论；(5) 交付与结论登记组 5 台账，交接组 8 做优化。

**验收证据**：v2 训练日志与 `ropstitch.metrics.json`、交付后探针 `PASS (5/5)`、三列 A/B 对比表与结论段。

**常见坑**：微调学习率不调小——大学习率把 v1 学到的冲掉，越微调越差；只挑好看的版本报——三列表里少一列都算选择性上报；新数据没过卡片就开训——先核对组 6 的 v2 卡片与清单，喂错数据训出的模型没有讨论价值。

# 第 8 组　六目拼接 · 端到端优化组

**本章地图**

**讲什么**：第 8 组任务书——一帧 800 ms 的预算怎么分、逐帧计时基线怎么测、组 7 组的模型怎么复现对账、fp32/fp16/int8 三档怎么比、5 对合并一次推理怎么设计。**为什么值得读**：实时系统的“实时”两个字由你负责——单次引擎调用压不进 300 ms，前面所有精度进步都上不了生产线。**学完你能**：独立测出一台机器的推理耗时账本，做一张“速度与精度权衡表”，掌握量化与批处理两条加速正道。

一句话概括你们组：第 7 组把模型训得准，你们组把它跑得**快**——目标只有一条：单次引擎调用 $\leq$ 300 ms（每帧总预算 800 ms），且优化前后全景输出**视觉无差异**。“又快又一样”，四个字，一学期。

## 8.1　组档案

**表 8.1**　第 8 组档案

| | |
|---|---|
| 项目 | 六目拼接（项目一，360Stitcher实时全景系统，详见第 一 章） |
| 组号与组名 | 第 8 组 • 端到端优化组 |
| 人数与投入 | 5 人；每周合计 6–8 小时（组内自行排班；fp16 档测试需向第 7 组申请 GPU 时段） |
| 负责的产出 | 逐帧计时基线、性能预算书（含复盘）、fp32/fp16/int8 三档“速度与精度权衡表”、batch 推理方案、（达标才交）量化版 onnx |
| 上游依赖 | 第 7 组：v1（公开数据版，实验三即交付）/v2（真机版，实验四联调）是你优化的对象；第 6 组：`benchmark.py` 是你的测量仪 |
| 下游依赖 | 量化版（若达标）经 `deliver_model.bat` 交付进系统；结论入组 5 双台账 |
| 工作目录 | `e2e_gpu/training/group12_e2e_optim` |
| 硬件要求 | 本机 CPU 为主战场（交付版引擎走 CPU 推理）；GPU 时段向第 7 组申请 |

目录名里的 `group12` 是源工程里引擎优化组的原始编号，与本课程“第 8 组”无关。口径先对齐：六目每帧把画面拆成 5 个相邻对，现状是逐对调 5 次引擎；**单次引擎调用 $\leq$ 300 ms** 是硬指标，**5 对合并成一次 batch 推理**是达成路径之一。加速有个铁门槛：**不能牺牲人眼可见的质量**——任何优化都要过“视觉无差异”这一关，所以精度测量和速度测量在你这里同等重要。

## 8.2　项目背景与你的位置

第 一 章给过一句话：实时全景系统每一帧从六路画面进来到全景上屏，总共只有 800 毫秒——采集预处理、引擎推理、渲染缝合、上屏显示，四段各分一块蛋糕。第 7 组的模型再准，单次推理要是慢吞吞，全景就得掉帧，用户看着就是幻灯片。你在流水线里的位置：端到端路线的**最后一公里**——先给整条链路做“耗时体检”（哪段慢、慢多少），再对大头（骨干网络推理）下刀：量化压缩、

批处理合并，刀刀要有数据佐证。第 5 组的台账会记下你的每一项优化：提速多少毫秒、精度劣化千分之几。

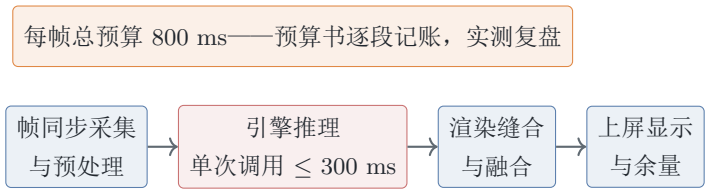
## 8.3　核心任务

一张预算书、一张基线表、一张权衡表、一份 batch 方案，四件套说清。

表 8.2　第 8 组核心交付：给整条链路做体检，再给推理下刀

| 交付 | 要做到什么 | 什么时候 |
|---|---|---|
| 性能预算书 | 800 ms总预算的分配表（预处理 / 引擎 ≤ 300 ms / 渲染 / 余量），每项写明测量方法；离场前复盘 | 实验一立项、离场复盘 |
| 逐帧计时基线 | 引擎模式跑 `--benchmark-stitch`，收集 `[BenchmarkFrame]` 行，汇总均值与 P95 | 实验二 |
| 速度与精度权衡表 | fp32 / fp16 / int8 三档：耗时（均值 / P95）× 重投影误差，同一冻结测试集、同一输入 | 实验四（招牌） |
| batch 推理方案 + 量化交付 | 5 对合并一次调用的方案设计；int8 劣化 < 10% 才交付量化版 | 实验四 |

一帧 800 ms 花在哪，先看下面的预算条：引擎推理是必须死磕的一段，其余三段也要有人盯着——预算书就是把每一段变成“有名有数有测法”的账。




**量化三档与校准集：把模型“压缩”但不压垮**

网络权重默认用 **fp32**（32 位浮点）存储与计算；**fp16** 减半，带宽立省；**int8** 用 8 位整数近似浮点，提速潜力最大、风险也最大。代价靠**校准集**控制：拿一小撮有代表性的样本跑一遍，统计各层取值范围，再决定每个数怎么映射到 8 位——校准集要用 val 批次的代表性样本，不能抓训练集凑数。另一条路是 **batch 推理**：5 个相邻对拼成 `[5,2,3,512,512]` 一次调用，省 4 次调用开销——GPU 爱吃“一大口”，不爱吃“五小口”。

**前沿速递：量化是这两年的主战场**

2024–2026 年大模型落地的关键词就是量化：手机上的 7B 级模型普遍以 4–8 位量化运行，端侧芯片把 fp16/int8 算力当核心卖点，推理框架把“选对硬件档位”做成显式选项。你的三档对比表，与工业界上线模型前的评测流程同构——做过一遍，以后聊“模型部署”就不虚。

**项目连线：第 8 讲的实战全对应**

第 8 讲“轻量化与分割”里有一条完整的方法链：模型大小与精度的取舍（下图）、压缩网络设计的思路、以及“快到能上手机/上板子”的工程诉求——你的三档权衡表就是把那一讲的方法论用在真模型上。batch 推理的思想在第 3 讲就埋了伏笔：向量化与成批计算总是快于逐个计算。而“视觉无差异”的验收观，来自第 5 讲——指标只是代理，最终要回到图像本身。

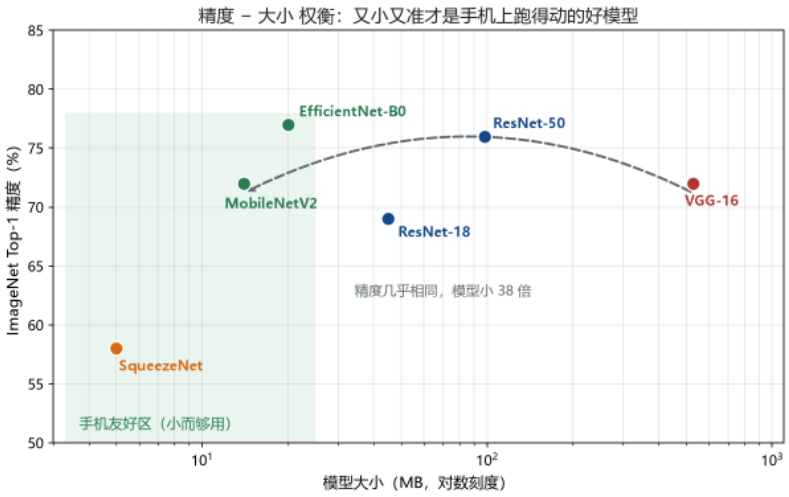



图 8.1　模型大小与精度的取舍：又大又准与又小又快之间永远要做选择——你的三档权衡表，就是给 `ropstitch.onnx` 在这张图上找点

## 8.4　第 19 周周五提交清单

提交线：2027年 1 月 22 日（第 19 周周五）完全提交离场。你的组交齐下表五样。

表 8.3　第 8 组提交清单（2027 年 1 月 22 日）

| 交付物 | 格式 | 说明 |
|---|---|---|
| 速度与精度权衡表 | PDF + xlsx | 招牌交付：三档 × 耗时（均值 / P95）× 重投影误差，注明测试集哈希与硬件环境 |
| 性能预算书（含复盘） | 一页 | 实验一的预算分配 vs 离场实测复盘，差异要有解释 |
| batch 推理方案设计 | 文档 2–4 页 | 5 对合并一次调用的方案、预期收益与验证计划 |
| 量化版 onnx（达标才交） | onnx + json | int8 劣化 < 10% 才交付；走 `deliver_model.bat`；探针结果如实附上 |
| 组报告与 AI 声明 | PDF 8–15 页 + 一页 | 按第 三 章模板；AI 使用声明逐条列清 |

## 8.5　AI 分工与人工必做

分工原则一句话：AI出草稿，人出签字——签字的人对数字负全责。

表 8.4　第 8 组 AI 分工与人工必做

| AI可以自动做 | 人必须亲自做 |
|---|---|
| 解读引擎侧源码的耗时构成，起草预算书框架 | 本机跑通引擎模式 benchmark 与逐帧计时采集 |
| 写三档汇总表与画耗时对比图的脚本 | 亲手复现组 7　v1 并与 `metrics.json` 对账 |
| 起草 batch 方案与量化步骤的文字 | 亲手执行量化与校准，核对劣化是否 < 10% |
| 解释报错、给排查建议 | 在权衡表与交付物上签字，对每个毫秒担责 |

求助纪律：组内 → 助教（研究生）→ 导师，求助附完整命令与完整输出。课程成绩建议构成：实验一不计分，实验二至四各 10 分共 30 分，平时作业 20 分（十二讲章后习题十二选二），项目成

果 50 分，总评满分 100 分（教师可调）；每次节点验收以组 5 双台账为唯一依据——你的每毫秒提速，都要能在台账里查到。

## 8.6　防糊弄指南与满分标准

**防人工糊弄指南（第 8 组版）**

**(1) 换输入比速度 = 换考卷**：三档对比必须同一冻结测试集、同一输入、同一机器同一状态，换任何一个条件，表就是废纸。**(2)** 只报耗时隐去精度（或反之）——权衡表的全部价值在“两列都在”；**选择性上报按虚构处理，虚构数据 = 该节点成绩作废**。**(3)** 量化版不达标硬交——劣化 $< 10\%$ 是硬线，越线就如实写“不交付”，交一个糊的全景比慢 50 ms 罪过大得多。**(4)** 交付不走 `deliver_model.bat` 手动拷文件——交付动作只有一条正道，绕道就走在了探针的枪口上。

**防 AI 糊弄指南（第 8 组版）**

**(1) AI 背的耗时数字一律不信**：哪段占多少毫秒，只认你机器上 `[BenchmarkFrame]` 日志算出来的数，它背的博客不算。**(2) AI 说“fp16/int8 一定更快”不等于更快**：CPU 上 fp16 可能跑不了甚至更慢——一切以实测为准，实测不了就注明环境限制。**(3) AI 算的均值与 P95 要复算**：用脚本从原始日志重算一遍再进表。**(4)** AI 起草的量化步骤先在小样本上验证过再推广——量化坏起来是“全军覆没”，不是“略有损耗”。

成果满分 50 分，怎么拿每一分见下表。

表 8.5　第 8 组成果满分标准（50 分制）

| 分值 | 项目 | 怎么拿满 |
|---|---|---|
| 20 | 功能达成 | 实验二逐帧计时基线完整（5 分）；实验三 v1 复现对账通过（5 分）；实验四三档权衡表 + batch 方案合格（10 分） |
| 10 | 证据完整 | 四次实验证据包齐全：预算书、逐帧日志、复现对账记录、三档原始数据；缺一件扣 5 分 |
| 8 | 报告质量 | 组报告按模板 8–15 页；权衡表讲得清、结论有数据、预算复盘诚实 |
| 7 | 协作与台账 | GPU 时段申请与使用有记录；与组 7/6 交接顺畅；结论按时入组 5 台账 |
| 5 | 加分 | 如：batch 方案做出原型并给出实测收益；单次引擎调用实测压进 300 ms；发现并反馈引擎管线的耗时 bug |

## 8.7　四次实验安排

四个节点与全班同频：实验一“大模型 agent 模式配置及应用”（2026 年 10 月 12 日）、实验二“匹配合适程序并执行本地部署”（11 月 2 日）、实验三“深度网络的模型训练实践”（11 月 23 日）、实验四“模型优化训练实践”（12 月 21 日）。每场按“目标—步骤—验收证据—常见坑”执行。

表 8.6　第 8 组四次实验总览

| 节点 | 日期 | 本组动作 | 核心验收证据 |
|---|---|---|---|
| 实验一 | 10月 12 日 | agent 解读耗时构成，写性能预算书 | 预算书 + 核对记录一页 |
| 实验二 | 11 月 2 日 | 引擎模式 benchmark，逐帧计时基线 | `[BenchmarkFrame]` 日志 + 基线表 |
| 实验三 | 11 月 23 日 | 复现组 7 v1，指标对账 | 复现报告 + 探针 `PASS (5/5)` |
| 实验四 | 12 月 21 日 | 三档权衡表 + batch 方案 ± 量化交付 | 权衡表 +（可选）量化版 onnx |

### 8.7.1　实验一（10 月 12 日）：拆解一帧的时间都花在哪，写性能预算书

**目标**：用大模型 agent 解读引擎一次调用的耗时构成——预处理、骨干网络、后处理各占多少——写一页“性能预算书”，给 800 ms 立规矩。

**步骤**：(1) 装好编程助手并切到 agent 模式；(2) 让 AI 读引擎侧源码（`e2e_gpu/cpp/src/` 的引擎通道），回答三问：一次调用前后各做了什么（缩放 512、归一化等预处理）、骨干网络前向在算什么、后处理（5 对变换链式累积与渲染）做了什么；(3) 让 AI 给出三段的先验占比估计，并追问“依据是什么”；(4) 逐条与代码对照核对，标“对 / 错 / 存疑”，至少纠正一处；(5) 写预算书：800 ms 的分配表（预处理 / 引擎 ≤ 300 ms / 渲染 / 显示余量），每项配一列“怎么测”。

**验收证据**：一页性能预算书 + “AI 解读 + 人工核对”记录。

**常见坑**：AI 把“模型推理”当成唯一耗时——预处理与后处理常常合计占到三成上下，漏记它们预算就是错的；预算书没有“怎么测”一列——不可测量的预算只是愿望清单；AI 拿通用模型的经验数字套本项目——引擎是自己家的，一切以本机实测为准，先验只当假设。

### 8.7.2　实验二（11 月 2 日）：引擎模式开起来，逐帧计时基线立起来

**目标**：跑通引擎模式 benchmark，采集逐帧计时，建立你组的第一本“耗时账”。

**步骤**：(1)部署路线二（部署与探针步骤同组 5 组实验二）：`bin/360stitcher_e2e.exe --benchmark-stitch=bin/simulation --engine=ropstitch`，确认日志出现 `[Engine] loaded` 与 `engine active`；(2) 让 benchmark 跑满 30 帧以上，逐帧收集 `[BenchmarkFrame]` 行（每帧各段耗时）；(3) 用脚本汇总均值与 P95，形成基线表（分：模型加载 / 预处理 / 引擎推理 / 后处理渲染）；(4) 与第 6 组的 `benchmark.py` 交叉验证一次，两本账对得上才算数。

**验收证据**：完整 `[BenchmarkFrame]` 日志、逐帧计时基线表（均值 + P95）、交叉验证记录。

**常见坑**：把模型加载（init）算进每帧耗时——加载只发生一次，逐帧账里要单列；后台浏览器、杀毒扫描让计时抖动——关干净、多跑几轮取稳定段；基线没固定输入就开跑——基线表必须注明测试图目录与帧数，下学期回看才知道测的是什么。

### 8.7.3　实验三（11 月 23 日）：复现第 7 组的公开版 v1，指标对得上账

**目标**：把第 7 组交付的 v1（公开数据版，实验三即交付——不用等真机数据）在你机器上完整复现一遍——指标可复现、数值对得上账，优化才有出发点。

**步骤**：(1) 确认 v1 在位：`bin/models/ropstitch.onnx` + `ropstitch.metrics.json`；(2) 跑 `python training/engine_probe_test.py`，期望 `RESULT: PASS (5/5)`——先证明系统用的就是这份模型；(3) 同一

冻结测试集复跑重投影误差与逐帧耗时，与 `metrics.json` 及组 7 的 A/B 数据对账，偏差应在 $10^{-3}$ 量级以内；(4) 写一页复现报告：命令、输出、与原记录的对照表，交组 5 入台账。

**验收证据**：探针 `PASS (5/5)`、复现对照表（你测的 vs `metrics.json`）、复现报告一页。

**常见坑**：对不上账先怪上一组——先查自己：测试集哈希对不对、输入尺寸与归一化一致不一致、机器状态相同不相同；中途动过 `bin/models/` 忘了还原——下次探针 FAIL 溯源半天才找到你头上；只复现精度不复现耗时——你的主战场是时间，耗时不复现等于白来。

### 8.7.4　实验四（12 月 21 日）：三档权衡表与 batch 方案，招牌交付收官

**目标**：完成 fp32/fp16/int8 三档“速度与精度权衡表”与 5 对合并 batch 推理方案；若 int8 精度达标（重投影误差劣化 $< 10\%$），交付量化版 onnx；本场同时是八组合并联调：对真机版 v2 补做三档对比，达标即交量化版，结论入台账。

**步骤**：(1) 三档准备：fp32 直接用 v1；fp16 需 GPU 或支持 fp16 的执行环境——向第 7 组申请时段，CPU 不适用就如实注明；int8 量化：校准集从第 6 组 val 批次选代表性样本（别抓训练集凑数）；(2) 同一冻结测试集、同一输入跑三档，记录耗时（均值 / P95）与重投影误差，成“速度与精度权衡表”；(3) 写 batch 方案：5 对拼成 `[5,2,3,512,512]` 一次调用的接口改动、预期收益与验证计划；(4) 达标判定：int8 劣化 $< 10\% \rightarrow$ 走 `deliver_model.bat` 交付量化版并重跑探针、如实记录；不达标 $\rightarrow$ 如实写结论不硬交；(5) 预算书复盘 + 全部结论入组 5 台账。

**验收证据**：三档权衡表（含原始数据）、batch 方案文档、量化交付与探针记录（或“不达标不交付”的结论说明）、预算书复盘。

**常见坑**：fp16 硬在 CPU 上跑——CPU 执行环境对 fp16 支持有限，跑不了是正常现象，注明条件去 GPU 上测；比较时忘了固定输入——换图等于换考卷，表作废；int8 校准集用了训练集——校准要有代表性，还要与训练数据拉开距离；量化版交付后探针表现与浮点版不同——如实记录现象，与助教确认判据，别为了绿灯改数据。

# 第 9 组　水下双目 · 数据组

**本章地图**

**讲什么**：第 9 组任务书——“视频体检 → 抽帧去冗 → 双目预处理 → 数据集组装”四步赛道怎么走，四个实验节点各做到什么程度，第 19 周周五交哪几样。**为什么值得读**：模型的天花板是数据踩出来的。四条并行赛道里你守的是“粮草”——训练组、部署组、评测组吃的每一对图，都从你手里过。**学完你能**：独立跑通体检与抽帧去冗全流程，说得清数据泄漏为什么必须按场景切分才防得住，用 2–3 种配比实验为全队选出更好的一口粮。

一句话概括你们组：整条水下双目流水线的“厨房”。别人讨论模型、算力、指标，你们负责把原料做成能下锅的粮食——本项目送给你们的一句话：**你们喂给模型的第一口粮，来自你们手中**。

## 9.1　组档案

**表 9.1**　第 9 组档案

| | |
|---|---|
| 项目 | 水下双目（项目二，中海油 HYSY123 FPSO 锚链 ROV 检查视频的双目立体感知，详见第 二 章） |
| 组号与组名 | 第 9 组 • 数据组 |
| 人数与投入 | 5 人；每周合计 6–8 小时（组内自行排班，建议每周固定 2 次） |
| 赛道 | 视频体检 → 抽帧去冗 → 双目预处理 → 数据集组装 |
| 上游依赖 | 开局资产包（500 对锚链图对、UWStereo 子集、零样本基线）；真机新视频由助教协调（实验四接入） |
| 下游依赖 | 第 10 –12 组共享你的数据集与划分（实验四联调对账） |
| 工作目录 | `tasks/u09_data/`（主线 `code/` 只读，见防糊弄指南） |
| 硬件要求 | 体检与抽帧用普通 CPU 电脑即可；实验三、实验四的小微调需向第 10 组预约 4080 时段 |

先把你们组的家底交个底：**开局资产包**已经发到组里——500 对锚链校正立体图对（`stereo_dataset`，无标签）、UWStereo 公开子集、零样本基线成绩，外加一版用公开数据初步微调的基线模型 v0。前期最小实现已经跑通，你的工作不再是“等米下锅”，而是把这套粮食管出质量：扩量、提质、调配比。UWStereo / FLSea / SQUID 等公开数据集用于扩量与对照；实验四联调时，助教协调的新录视频接入同一条流水线。另外把求助通道抄在工位上：组内互助 → 研究助理 → 项目负责人，逐级上升；求助必须附完整命令与完整输出，只发一句“跑不通”等于没发。

## 9.2　项目背景与你的位置

第 二 章讲过来龙去脉：中海油 HYSY123 FPSO 的系泊锚链靠 ROV 定期下水检查，检查视频要变成锚链的三维信息，靠双目立体匹配；技术方案是 FoundationStereo 当教师模型、给海量帧打伪标签、学生模型向它学习，资产包里的 500 对自建立体图对开局就在你手里，UWStereo 公开数据用于扩量对照。问一句：这 500 对、以及将来更多的训练图对，从哪来？——从你们手里来。你在流

水线最上游：体检决定哪些视频值得吃，抽帧去冗决定哪些帧入池，预处理决定图对是否对齐，配比直接决定下游微调的成色上限。教师打标、过滤、学生训练的闭环见下图——闭环入口的“数据”两个字，写的就是你们组。

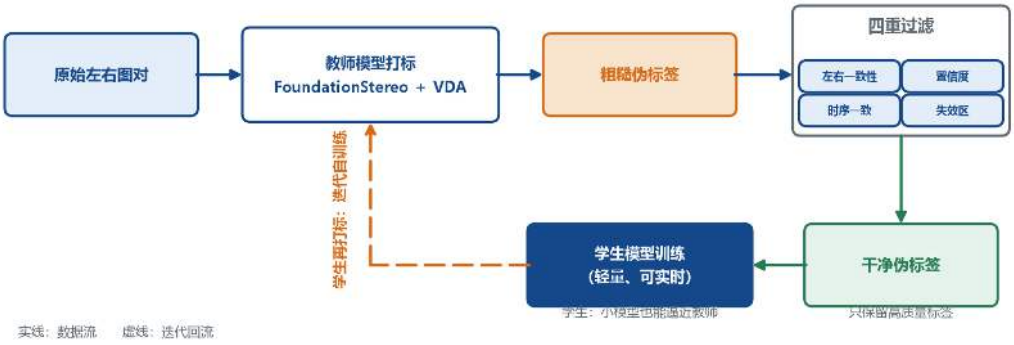



图 9.1　教师–学生伪标签闭环：教师打标、过滤、学生训练、再打标——闭环入口的数据供应正是第 9 组的交付物

## 9.3　核心任务

四步赛道一张表看明白，随后逐段说人话。

表 9.2　第 9 组四步赛道：干什么与产出什么

| 阶段 | 干什么 | 产出 |
|---|---|---|
| 视频体检 | 逐段视频出“体检单”：按场景切割，统计模糊度与低照度占比，排出抽帧优先级 | 体检报告 |
| 抽帧去冗 | 场景内均匀抽帧，近重复帧剔重，记录每帧溯源信息 | 候选帧池 |
| 双目预处理 | 左右目对齐成校正立体图对（第 二 章的双目校正） | 立体图对 |
| 数据集组装 | 按场景/日期级切分训练集与验证集，写数据卡 | 数据集 v0.1 |

逐段展开。**视频体检**是给每段 ROV 视频先做一次体检：脚本自动把视频按场景切割（镜头切换、内容突变处下刀），再统计每个场景的模糊度与低照度帧占比——浑浊水里又糊又黑的片段，抽了也是浪费模型胃口，体检单就是“哪段值得吃”的排序依据。**抽帧去冗**：相邻视频帧几乎一模一样，全抽等于让模型反复吃同一道菜；正确做法是场景内均匀抽帧，再用近重复剔除（感知哈希一类指标）去掉“双胞胎帧”，并记录每一帧来自哪段视频的哪一秒——没有溯源信息，评测组将来就没法按场景分桶。**双目预处理**把左右目画面校正成行对齐的立体图对，工程细节第 二 章讲过（下图是一对真实产出的左目），你们的职责是让它批量化、可复现。**数据集组装**是最后一步，也是最容易翻车的一步——翻车的名字叫数据泄漏，见下面的知识盒。



**图 9.2**　校正立体图对 • 左目（锚链链环清晰、行对齐）：数据组装就是把成百上千对这样的图对管起来

图片来源：本项目真实数据（E:\锚链视频\stereo_dataset）

**数据泄漏（data leakage）：验证成绩虚高的头号元凶**

一段锚链视频里相邻两帧几乎一样。如果按“帧”随机切分训练集与验证集，同一场景的“双胞胎帧”一边一张——模型在验证集上等于闭卷考试看了答案，验证 EPE（端点误差，越小越好，下一节细讲）好看得不得了，换一段没见过的视频就原形毕露。防法只有一条：**按场景/日期级切分**——同一天、同一场景的所有帧，只能进训练集或验证集中的一边。切分清单本身也是交付物，第 11 组微调、第 12 组评测都按它执行。

**项目连线：数据组的活与第一篇哪几讲有关**

第 5 讲告诉你，图像在卷积网络眼里只是一摞数字矩阵——你抽的每一帧最终都变成它，所以“模糊度”“低照度”这些体检指标，本质是在给这摞数字的质量打分；第 6 讲讲训练循环与数据增强，你们的去冗与配比本质是“配餐”：同一个模型，粮的好坏直接写进 loss 曲线。实验三你们还要亲手跑一次小微调，把这条因果链亲眼看一遍——数据组也要懂训练。

## 9.4　第 19 周周五提交清单

提交线：2027年 1 月 22 日（第 19 周周五）完全提交离场。你的组交齐下表各件，每样都与第 三 章的总清单模板对应；水下四个组还要共同填写《四组交接总表》，本表最后一行就是总表里你组的“交接内容”栏。

**表 9.3**　第 9 组提交清单（2027 年 1 月 22 日）

| 交付物 | 格式 | 说明 |
|---|---|---|
| 体检配置与体检报告 | 配置 + PDF | 主线 `code/tools/video_check.py` 的运行配置；样例视频的体检报告（场景切割 + 模糊度/低照度统计 + 抽帧优先级排序） |
| 水下立体图对数据集 v0.1 | 目录 + 清单 | 资产包 500 对的体检与重组子集 + UWStereo 扩量子集；附数据卡（数量/来源/配比）与训练/验证划分清单（场景/日期级） |
| 数据配比报告 | PDF，2–4 页 | 2–3 种配比的小微调验证 EPE 对比、推荐配比与理由 |
| 小组项目报告 | PDF，8–15 页 | 按第 三 章模板：摘要/背景/方法/实验/结论/分工声明 |
| 证据包 | zip | 四次实验各一包，命令 + 日志 + 产物三件套，日志为终端完整输出 |
| AI 使用声明 | 一页 | 哪些事 AI 做了、人做了什么核验，逐条列清 |
| 交接内容 | 《四组交接总表》一行 | 数据集 v0.1 与划分清单交第 10/11/12 组；体检报告与配比结论交第 11 组定训练范围 |

## 9.5　AI 分工与人工必做

分工原则一句话：AI出草稿，人出签字——数据集里每一个数字，签字的人负全责。

**表 9.4**　第 9 组 AI 分工与人工必做

| AI可以自动做 | 人必须亲自做 |
|---|---|
| 解读体检脚本参数、生成运行命令草稿 | **抽 30 帧人工看图核对体检指标排序是否合理（AI 不可代看）** |
| 起草抽帧、去冗、重命名脚本草稿 | 在本机与 4080 上实跑每一条命令 |
| 把体检数字画成图表，起草数据卡与报告文字 | 逐条核对 AI 结论（在脚本里找到对应行才算数） |
| 建议数据配比方案与过滤阈值候选 | 亲眼确认划分清单无泄漏（同场景只进一边） |
| 解释报错信息、给排查建议 | 在证据包与数据卡上签字，对数字担责 |

## 9.6　防糊弄指南与满分标准

**防人工糊弄指南（第 9 组版）**

先背全项目通用的纪律红线：**(1) 原始视频/帧不上云盘、不进在线 AI（保密!）**——体检统计数字可以写进提示词，视频与帧本体绝对不行；**(2) 主线 `code/` 只读不改**，你们的一切产出放进 `tasks/u09_data/`；**(3) 不直接联系甲方**，数据需求一律走项目负责人。本组再加三条：**(4) 抽帧只报数量不报去冗率 = 没做去冗**——“双胞胎帧”糊弄不了训练曲线；**(5) 划分清单混场景 = 数据泄漏**，验证 EPE 再好看也不算数，一经查实该节点成绩作废；**(6) 代跑必须留运行者姓名与时间**，谁跑的谁负责。

**防 AI 糊弄指南（第 9 组版）**

**(1) AI 产出必须实跑验证**——第四条纪律红线：它声称“脚本写好了、应该能跑”，只有你终端里跑出来的那份算数。**(2) AI 没看过你的视频**——它对“这段视频浑不浑”的任何回答都是编的，看图的事只能你来，这正是 30 帧人工核对 AI 不可代看的原因。**(3) AI 给的阈值（切割阈值、去重阈值）是“常见值”不是“你们数据的合理值”**——必须用体检数字调出来。**(4) 禁止把 AI 的回答直接粘进数据卡与实验记录**——记录里只许出现你亲手跑出来的内容。

按课程建议方案，实验一不计分，实验二至四各 10 分共 30 分，平时作业 20 分（十二讲章后习题十二选二）；期末成果满分 50 分（教师可调）。成果怎么拿满，看下面这张 50 分制细则。

**表 9.5**　第 9 组成果满分标准（50 分制）

| 分值 | 项目 | 怎么拿满 |
|---|---|---|
| 20 | 功能达成 | 实验一体检报告 + 30 帧人工核对记录（5 分）；实验二抽帧去冗全流程 + 防泄漏划分清单（5 分）；实验三资产包数据小微调、亲眼看 loss 下降（5 分）；实验四 2–3 种配比对比 + 推荐配比（5 分） |
| 10 | 证据完整 | 四次实验证据包三件套齐全、可复现；缺一件扣 5 分 |
| 8 | 报告质量 | 组报告按模板 8–15 页，数据卡清晰、配比结论有 EPE 数据支撑 |
| 7 | 协作与台账 | 排班执行有记录；4080 时段预约与用量记录无缺失；求助记录规范 |
| 5 | 加分 | 如：自研更聪明的去冗指标并用数字证明有效；为浑浊/低照度场景设计专门过滤规则并被训练组采用；把体检做成一键复跑的工具 |

## 9.7　四次实验安排

四个节点与全班同频（实验一 2026 年 10 月 12 日、实验二 11 月 2 日、实验三 11 月 23 日、实验四 12 月 21 日），但每场都有本组的命令级动作。先看总览表，再逐场展开；每场按“目标—步骤—验收证据—常见坑”执行：步骤照抄就能跑，证据一栏就是成绩。总则：实验一至三是离线自研——资产包开局就发到每组手里，谁也不用等谁；实验四是合并联调——四条赛道的成果在同一台机器、同一套考卷上会师。

表 9.6　第 9 组四次实验总览

| 节点 | 日期 | 本组动作 | 核心验收证据 |
|---|---|---|---|
| 实验一 | 10月 12 日 | 配 agent，实跑视频体检，人工核对 30 帧 | 体检报告 + 核对记录 |
| 实验二 | 11 月 2 日 | 工具链装通，抽帧去冗全流程 + 防泄漏切分 | 去冗日志 + 划分清单 |
| 实验三 | 11 月 23 日 | 干跑数据集小微调 1–2 个 epoch | 训练日志 + loss 曲线 |
| 实验四 | 12 月 21 日 | 2–3 种配比对比，选出更优配比 | EPE 对比表 + 推荐结论 |

### 9.7.1　实验一（10 月 12 日）：给视频开体检单，你亲自看 30 帧

**目标：**配好大模型编程助手（GLM类命令行 agent），实跑“视频体检脚本”，产出第一份体检报告，并由人工核对排序是否合理。

**步骤：**(1) 按课堂演示装好编程助手并切换到 agent 模式；(2) 让 agent 解读主线 `code/tools/video_check.py` 的参数——场景切割阈值怎么设、模糊度与低照度怎么判——并把它的解释逐条与源码对照，标注“对/错/存疑”；(3) 在 `tasks/u09_data/` 下写运行配置，对一段样例视频实跑：`python code/tools/video_check.py --input demo.mp4 --out tasks/u09_data/check/`，得到按“最值得抽帧”排序的体检报告；(4) **抽 30 帧人工看图**，核对排序是否合理——糊的、黑的是不是真排到了后面（AI 不可代看，这半小时没有人能替你）。

**验收证据：**agent 解读 + 人工核对记录一页（含 AI 原话、你的核对结论、至少一处纠正）、体检报告、30 帧看图核对记录。

**常见坑：****ffmpeg 装了但 PATH 没配**——命令行敲 `ffmpeg -version` 报“不是内部或外部命令”，脚本起不来，把 ffmpeg 的 bin 目录加进系统 PATH 再重开终端即可；agent 替你“想象”体检结果——它没看过视频，数字必须来自你的终端；只核对前 5 帧就签字——排序靠后的片段才是抽查重点。

### 9.7.2　实验二（11 月 2 日）：抽帧去冗一条龙，切分防住泄漏

**目标：**装通 ffmpeg / PySceneDetect / OpenCV 工具链，跑通“场景切割 → 段内抽帧 → 近重复去冗 → 按场景/日期切分”全流程，组装干跑数据集 v0.1 的雏形。

**步骤：**(1) 装三件套并逐个验证：`ffmpeg -version`、`scenedetect version`、`python -c "import cv2"`，三条都有版本号输出才算装通；(2) 用 PySceneDetect 切场景：`scenedetect -i demo.mp4 detect-content list-scenes`；(3) 场景内均匀抽帧，再用感知哈希做近重复剔除（阈值先抄默认值，再用体检数字调）；(4) 按场景/日期级切分训练集与验证集——同一天同一场景只进一边；(5) 生成划分清单与数据卡，每帧记录溯源信息（哪段视频、哪一秒、哪个场景）。

**验收证据：**工具链三条版本号输出；抽帧与去冗日志（抽了多少、去掉多少、留了多少）；训练/验证划分清单与数据卡。

**常见坑：****抽出来的重复图没去冗**——目录看着热闹，全是双胞胎帧，模型等于反复吃一道菜；按帧随机切分导致数据泄漏——切分粒度是场景/日期，不是帧；光顾着抽帧忘了记溯源信息——评测组没法按场景分桶，返工的是你们自己。

### 9.7.3　实验三（11 月 23 日）：数据组也要懂训练——亲眼看 loss 下降

**目标：**用资产包数据（500对采样与 UWStereo 混合），在 4080 上跑一次小规模微调（1–2 个 epoch 即可，FoundationStereo 或第 10 组降级链可跑的版本），亲眼看 loss 下降——理解“数据怎么

喂、模型怎么吃”。

**步骤**：(1)向第 10 组预约 4080 时段，按其部署手册激活环境；(2) 用第 11 组同款的启动命令，把数据路径换成你自己的数据集，跑 1–2 个 epoch 的小微调；(3) 记下初始 loss 与每个 epoch 结束的 loss，画成曲线；(4) 换一版你改过的数据子集（如去掉低照度帧）再跑一次，对比两条 loss 曲线——这就是“数据质量 → 训练效果”的最小实证。

**验收证据**：两段训练日志（含每个 epoch 的 loss 数字）、两条 loss 曲线、数据子集差异说明。

**常见坑**：数据路径写错，dataloader 空转却“看起来在训”——先确认日志里读到了多少对图；epoch 贪多超出 1–2 的范围，占用别人的 4080 时段——时段是全队的；只看最后一步 loss 不看走势——一条降、一条平还是要能说清原因。

### 9.7.4　实验四（12 月 21 日）：换配比、调阈值，用验证 EPE 选出更好的一口粮

**目标**：调整数据配比与过滤阈值，对比 2–3 种配比下小微调的验证 EPE，找出更优配比，形成“配比—效果”结论交给全队——本场同时是四组合并联调：新配比数据集接入第 11 组重训、与第 12 组统一口径总评测，成果进《四组交接总表》。

**步骤**：(1) 设计 2–3 种配比（如：全量 / 去低照度 / 清澈场景加权），验证集固定为同一份；(2) 每种配比各跑一次小微调，训练参数与实验三完全一致，只动数据；(3) 用第 12 组的一键评测脚本（统一口径）测每种配比下的验证 EPE；(4) 填对比表：配比、帧数、验证 EPE、训练耗时，写不少于 100 字的结论并给出推荐配比。

**验收证据**：配比设计说明、各配比训练日志、EPE 对比表、推荐结论。

**常见坑**：换配比时顺手改了学习率或 epoch——变量没控住，对比无效；验证集不固定，三个配比三份验证集，数字没法比；只比 EPE 不记耗时——推荐配比说不清代价，训练组没法排时段。

# 第 10 组　水下双目 · 部署组

**本章地图**

**讲什么**：第 10 组任务书——“环境部署 → 零样本冒烟 → 微调护航 → 性能优化”四步怎么走：FoundationStereo 怎么在一块真实的 GPU 上跑起来、跑得快，每个实验留什么证据。**为什么值得读**：教师模型是十亿级参数的立体大模型，它装不装得上、跑不跑得动，决定整个伪标签方案成不成立。全队的 4080，归你管。**学完你能**：独立完成 GPU 环境部署与故障降级，用 EPE/耗时/显存三项数字完成模型“选定”，给出有数据支撑的“精度–速度”权衡建议。

一句话概括你们组：全队的“发动机舱”。你们的任务书标题可以换成一句话：**让大模型在一块真实的 GPU 上跑起来、跑得快**——跑不起来，后面的训练评测全是纸上谈兵；跑不快，全队的粮食预算就要砍半。

## 10.1　组档案

表 10.1　第 10 组档案

| | |
|---|---|
| 项目 | 水下双目（项目二，中海油 HYSY123 FPSO 锚链 ROV 检查视频的双目立体感知，详见第 二 章） |
| 组号与组名 | 第 10 组 • 部署组 |
| 人数与投入 | 5 人；每周合计 6–8 小时（组内自行排班，建议每周固定 2 次） |
| 赛道 | 环境部署 → 零样本冒烟 → 微调护航 → 性能优化 |
| 上游依赖 | 开局资产包：FoundationStereo 权重与可跑环境（最小实现已完成——你是它的压测员与优化师，不是从零部署者） |
| 下游依赖 | 第 11 组（微调环境与闸门时段）、第 12 组（推理与评测链路）；实验四联调统一用你验证过的环境 |
| 工作目录 | `tasks/u10_deploy/`（主线 `code/` 只读，见防糊弄指南） |
| 硬件要求 | RTX 4080 工作站一台（32GB，全队共用，你组排班值守）；其余成员用 CPU 电脑做脚本与文档 |

先说清楚你们最值钱的资产：那块 4080 是全队共用的发动机——第 9 组微调、第 11 组训练、第 12 组批量推理都要来你这约时段。你们组的隐性职责是**时段台账**：谁、哪天、用了多久、跑了什么，一条不缺。台账记全，排班才不吵架，故障才能溯源。求助通道照旧：组内互助 → 研究助理 → 项目负责人，逐级上升，求助必须附完整命令与完整输出——GPU 报错千奇百怪，截半张图等于让医生看半张化验单。

## 10.2　项目背景与你的位置

第 二 章讲过方案：ROV 检查视频靠双目立体匹配变成锚链三维信息，路线是 FoundationStereo 当教师打伪标签、学生模型学习，；前期最小实现已把这条路走通——500 对数据、初步微调基线 v0 都在开局资产包里。你这一关的任务：把“能跑”带到“跑得快、跑得稳”。这套方案落地的第一关

就在你们组——教师模型参数量大，conda 环境、CUDA 版本、flash-attn 编译，任何一环卡住，全项目停摆。你在“发动机舱”：第 11 组的训练、第 12 组的评测、第 9 组的微调实践，全部跑在你验证过的环境里。你交付的不是模型，而是“别人开机即用”的环境、手册与三项数字。本组的特色纪律是：**基础模型的“选定”用实跑数字完成；失败则按降级链处置并书面报告**——降级链是 vitl（大杯）→ vits（中杯）→ IGEV++（另一款开源立体匹配模型），每一级都要留下“卡在哪、换了什么、数字多少”的记录。

## 10.3　核心任务

四项任务一张表，随后说人话。

**表 10.2**　第 10 组四项任务：干什么与产出什么

| 任务 | 干什么 | 产出 |
|---|---|---|
| 环境部署 | 按清单装通 conda / CUDA / flash-attn，跑通 `run_demo.py` | 部署手册 |
| 零样本冒烟 | UWStereo 50对零样本推理，EPE/耗时/显存三项都记 | 冒烟报告 |
| 微调护航 | 给第 9/11 组提供训练环境与时段，记录显存曲线与吞吐 | 显存/吞吐报告 |
| 性能优化 | 批量/分辨率/FP16 对比 + ONNX 导出初探，给出“精度–速度”权衡建议 | 权衡建议 |

先补一个词：**零样本**（zero-shot）——不做任何微调，拿预训练权重直接推理。零样本冒烟的意义是给全队一个“出厂底分”：教师模型没经过水下微调时到底几斤几两，第 11 组将来“微调提升多少”，全靠这个数当分母。逐项展开：**环境部署**是体力活加侦探活，照清单装，装不上按降级链换；**零样本冒烟**要三项数字一个不少——EPE（精度）、单对耗时（速度）、显存峰值（能不能装下）；**微调护航**是你们对全队的服务义务，训练时段的显存曲线与吞吐（对/分钟）由你们记录并公告；**性能优化**是学期末的收官戏，三个旋钮扫一遍，再摸一摸 ONNX 导出这条路通不通。模型大小与精度的取舍直觉见下图——你们实验四干的就是在真实模型上量这条曲线。

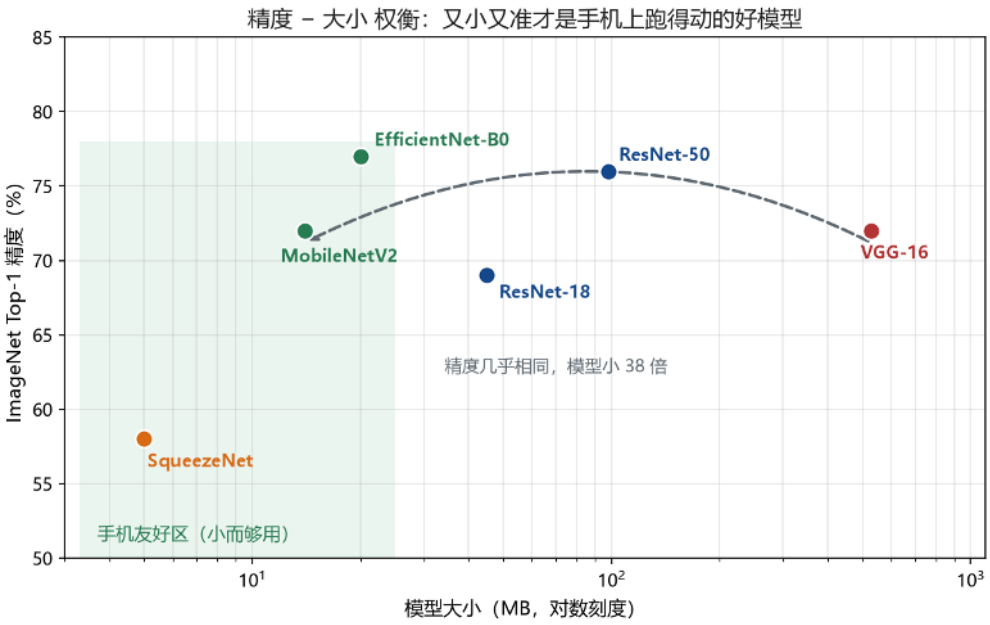



**图 10.1**　模型大小与精度的取舍散点（示意）：更大通常更准但更慢更吃显存——部署组的日常就是在真实权重上量出这条权衡

**显存去哪了：权重、激活、梯度三口锅**

模型参数本身（权重）占一口锅；推理时 layer 之间的中间结果（激活）占第二口；训练时还要额外存梯度与优化器状态，第三口最大——所以**微调远比推理吃显存**。可控的三个旋钮：**批量大小**（batch size，一次一起送几张图，越大越吃显存）、**分辨率**（图小一半，显存与耗时大幅下降）、**精度**（FP16 半精度，数字从 32 位存成 16 位，省显存提速，但可能掉精度）。实验四扫的就是这三个旋钮。

**项目连线：部署组的活与第一篇哪几讲有关**

第 8 讲是你们的方法论老家：轻量化与部署——模型压缩、深度可分离卷积、ONNX 格式、“精度换速度”的整套取舍，都在那一讲打过样。你们学期末的权衡建议，就是那一讲思想在十亿级立体大模型上的实弹演习。

## 10.4　第 19 周周五提交清单

提交线：2027年 1 月 22 日（第 19 周周五）完全提交离场。你的组交齐下表各件，每样都与第 三 章的总清单模板对应；《四组交接总表》由水下四组共填，本表最后一行就是总表里你组的“交接内容”栏。

表 10.3　第 10 组提交清单（2027 年 1 月 22 日）

| 交付物 | 格式 | 说明 |
|---|---|---|
| 部署手册 | PDF/Markdown | 从裸机到 `run_demo.py` 的每一条命令与每一个坑，别人照做必须能复现 |
| 冒烟报告 | PDF，2–4 页 | UWStereo 50 对零样本 EPE/耗时/显存三项数字；如走降级链，附完整处置记录 |
| 性能优化与权衡建议 | PDF，2–4 页 | 批量/分辨率/FP16 三组对比表 + ONNX 导出初探测速 + 推荐设置 |
| 小组项目报告 | PDF，8–15 页 | 按第 三 章模板：摘要/背景/方法/实验/结论/分工声明 |
| 证据包 | zip | 四次实验各一包，命令 + 日志 + 产物三件套，日志为终端完整输出 |
| AI 使用声明 | 一页 | 哪些事 AI 做了、人做了什么核验，逐条列清 |
| 交接内容 | 《四组交接总表》一行 | 部署手册与环境清单交第 9/11/12 组；显存曲线与吞吐数字交全队排时段用 |

## 10.5　AI 分工与人工必做

分工原则一句话：AI出草稿，人出签字——4080 上跑出的每个数字，签字的人负全责。

表 10.4　第 10 组 AI 分工与人工必做

| AI可以自动做 | 人必须亲自做 |
|---|---|
| 生成“4080 环境配置清单”草稿 | 在 4080 上实跑每一条命令 |
| 解读依赖冲突、CUDA 与 flash-attn 编译报错 | **逐条对照官方仓库核实清单（以仓库为准，不以 AI 为准）** |
| 写显存采样与测速脚本草稿 | 记全 EPE/耗时/显存三项数字，缺一不可 |
| 建议降级链处置方案、起草手册与报告 | 降级链处置与推荐设置逐项签字 |
| 解释报错信息、给排查建议 | 在证据包与台账上签字，对数字担责 |

## 10.6　防糊弄指南与满分标准

**防人工糊弄指南（第 10 组版）**

先背全项目通用的纪律红线：**(1) 原始视频/帧不上云盘、不进在线 AI（保密!）**——公开数据集不涉密，甲方 ROV 视频帧绝对涉密，测试输入只用公开数据；**(2) 主线 `code/` 只读不改**，你们的一切产出放进 `tasks/u10_deploy/`；**(3) 不直接联系甲方**。本组再加三条：**(4) 冒烟跑通 1 对就当 50 对交差 = 没做**——50 对的均值才有资格当基线；**(5) 显存只记平均值不记峰值 = 没记**——爆显存永远发生在峰值；**(6) 代跑与值守必须留姓名与时间**，时段台账一条不缺。

**防 AI 糊弄指南（第 10 组版）**

**(1) AI 产出必须实跑验证**——第四条纪律红线：它说“这样能装上”，只有你 4080 上的终端说了算。**(2) AI 报的版本号十有八九过时或虚构**——CUDA、PyTorch、flash-attn 的版本对应关系，以官方仓库与 `conda list` 输出为准，它背的是去年的黄历。**(3) AI 说“跳过 flash-attn 没**

**关系”不算结论**——跳过可以，但必须实跑确认对速度与显存的影响并写进降级链记录。**(4) AI 报的耗时/显存/EPE 必须与你的脚本和 `nvidia-smi` 输出对上账**，对不上以你的终端为准。

按课程建议方案，实验一不计分，实验二至四各 10 分共 30 分，平时作业 20 分（十二讲章后习题十二选二）；期末成果满分 50 分（教师可调）。成果怎么拿满，看下面这张 50 分制细则。

**表 10.5**　第 10 组成果满分标准（50 分制）

| 分值 | 项目 | 怎么拿满 |
|---|---|---|
| 20 | 功能达成 | 实验一配置清单经逐条核实（3 分）；实验二 `run_demo.py` 跑通 + 50 对冒烟三项数字齐全（10 分）；实验三微调护航显存曲线与吞吐（2 分）；实验四三组对比 + 权衡建议（5 分） |
| 10 | 证据完整 | 四次实验证据包三件套齐全、可复现；缺一件扣 5 分 |
| 8 | 报告质量 | 组报告按模板 8–15 页，部署手册可复现、权衡建议有数据支撑 |
| 7 | 协作与台账 | 4080 时段台账无缺失；值守与代跑记录规范；求助记录规范 |
| 5 | 加分 | 如：ONNX 导出真跑通并给出可用性结论；写一键环境校验脚本被三组采用；FP16 精度损失的定量分析 |

## 10.7　四次实验安排

四个节点与全班同频（实验一 2026 年 10 月 12 日、实验二 11 月 2 日、实验三 11 月 23 日、实验四 12 月 21 日），每场都有本组的命令级动作。先看总览表，再逐场展开；每场按“目标—步骤—验收证据—常见坑”执行：步骤照抄就能跑，证据一栏就是成绩。总则：实验一至三是离线自研——资产包开局就发到每组手里，谁也不用等谁；实验四是合并联调——四条赛道的成果在同一台机器、同一套考卷上会师。

**表 10.6**　第 10 组四次实验总览

| 节点 | 日期 | 本组动作 | 核心验收证据 |
|---|---|---|---|
| 实验一 | 10月 12 日 | agent 生成环境清单，逐条对照官方仓库核实 | 核实版清单 |
| 实验二 | 11 月 2 日 | 4080 部署 + `run_demo.py` + 50 对冒烟 | 三项数字表 + 处置记录 |
| 实验三 | 11 月 23 日 | 部署环境跑通小微调，记显存曲线与吞吐 | 显存曲线 + 吞吐数字 |
| 实验四 | 12 月 21 日 | 批量/分辨率/FP16 对比 + ONNX 初探 | 对比表 + 权衡建议 |

### 10.7.1　实验一（10 月 12 日）：让 agent 开清单，你逐条当审计

**目标：**用 agent 生成“4080 环境配置清单”草稿（conda / CUDA / flash-attn），并逐条对照官方仓库核实——清单是你们整个学期的施工图，错一条，实验二还你一夜。

**步骤：**(1) 让 agent 读 FoundationStereo 官方仓库说明，产出清单草稿：Python 版本、PyTorch 与 CUDA 版本、`flash-attn` 版本、其余依赖；(2) 逐条到官方仓库核对（README、环境文件、`requirements.txt`），标注“对/错/存疑”——**以仓库为准，不以 AI 为准**；(3) 每一条后面补一行“怎么验证装好

了”，如 `python -c "import torch; print(torch.cuda.is_available())"` 应输出 `True`；(4) 定稿存 `tasks/u10_deploy/env_checklist.md`。

**验收证据**：清单草稿、逐条核对标记版、带验证命令的终版清单，三样齐全。

**常见坑**：AI编造不存在的版本组合——它把“常见搭配”当“官方要求”；官方仓库更新后 AI 的记忆停在旧版本——核对日期写进清单；清单只有“装什么”没有“怎么验”——验收时按缺一半算。

### 10.7.2　实验二（11 月 2 日）：装上 4080，跑通第一对图（本组主力）

**目标**：按清单复现部署（资产包里已有可跑环境——你的价值是独立复现与压测），跑通 `run_demo.py`；UWStereo 50 对零样本冒烟，EPE/耗时/显存三项都记；受阻按降级链（vitl → vits → IGEV++）验证并记录。

**步骤**：(1) 按清单装环境，`conda list` 全量存档；(2) 下载权重并校验完整性，先跑官方样例 `python run_demo.py`，看到视差图输出才算部署成功；(3) 跑 UWStereo 50 对零样本推理，逐对记 EPE、单对耗时、显存峰值，汇总成三项数字表；(4) 受阻按降级链处置：vitl 装不上或跑不动 → vits → IGEV++，每级记录“卡在哪、换了什么、三项数字多少”；(5) 中途显存爆就降分辨率重试，同样记录。

**验收证据**：`run_demo.py` 成功日志；50 对三项数字表；降级链处置记录（如有）。

**常见坑**：**flash-attn 装不上**——十有八九是版本与 CUDA 不匹配，别死磕一整晚，按降级链走并记录，降级不是失败，无记录才是失败；**显存爆**——先降分辨率，再降批量；权重文件没下全，报错长得像“模型坏了”，先校验文件完整性再怀疑人生；驱动与 CUDA 版本不匹配——`nvidia-smi` 先看驱动支持的 CUDA 上限。

### 10.7.3　实验三（11 月 23 日）：微调护航——你的环境要能“载训练”

**目标**：在自己部署的环境里跑通小规模微调（与第 11 组同款配置、1–2 个 epoch），记录显存占用曲线与吞吐，给全队的训练排班拿出数字依据。

**步骤**：(1) 与第 11 组对齐启动命令与配置快照；(2) 用第 9 组的资产包数据（500 对采样）跑 1–2 个 epoch 微调冒烟；(3) 开后台采样记录显存曲线（按秒采 `nvidia-smi`）与吞吐（对/分钟）；(4) 给出估算：一块 4080 一周能吃多少对数据——这个数字直接决定全队的训练计划。

**验收证据**：微调成功日志、标了峰值的显存曲线、吞吐数字与排班建议。

**常见坑**：只记显存平均值不记峰值——爆显存永远发生在峰值时刻；训练时有人同时在上面临时推理，数字互相污染——训练时段 4080 独占，写进值守制度；曲线没标时间轴，事后没法与训练日志对账。

### 10.7.4　实验四（12 月 21 日）：三组旋钮 + ONNX 初探，给出权衡建议

**目标**：推理性能优化——批量大小/分辨率/精度（FP16）对比，ONNX导出初探与测速，给出有数字撑腰的“精度–速度”权衡建议——本场同时是四组合并联调：优化结论与第 11 组重训、第 12 组总评测对账，成果进《四组交接总表》。

**步骤**：(1) 固定同一批 20 对测试图与同一套评测口径（请第 12 组把关）；(2) 扫三组旋钮：批量 1/2/4、分辨率 1.0x/0.5x、FP32/FP16，每组记 EPE、单对耗时、显存峰值；(3) ONNX 导出初探：能导出则测速并记录不支持的算子，导不出则写清卡在哪一步、报错原文是什么；(4) 写一页权衡建议：推荐一组“够用且快”的设置，每个推荐后面都跟着数字。

**验收证据**：三组对比表（EPE + 耗时 + 显存）、ONNX 导出记录、权衡建议一页。

**常见坑**：FP16 只记提速不记精度损失——快了 30%、差了 3 个像素，白快；改了分辨率却拿不同口径的 EPE 比——对比无效；ONNX 导出失败就悄悄跳过——初探的价值恰恰在“为什么导不出”的记录里。

# 第 11 组　水下双目 · 训练组

**本章地图**

**讲什么**：第 11 组任务书——“零样本基线 → UWStereo 微调复现 → 训练策略与超参优化”三步赛道怎么走，四次实验各做到什么程度，第 19 周周五交哪几样。**为什么值得读**：你们是全项目唯一“改模型权重”的组。微调成不成立，直接决定伪标签闭环转不转得起来；而这一切的前提是先有零样本基线。**学完你能**：把训练配置读成人话，亲手让一个立体大模型在你的数据上 loss 下降，用“一次只动一个旋钮”的方法做超参对比实验。

一句话概括你们组：全项目的主战部队。项目送给你们一句话：**亲手把一个立体大模型“教会”一点水下本领**——它本来是岸上练出来的好眼力，你们负责把它掰得看得清浑水里的锚链。

## 11.1　组档案

**表 11.1**　第 11 组档案

| | |
|---|---|
| 项目 | 水下双目（项目二，中海油 HYSY123 FPSO 锚链 ROV 检查视频的双目立体感知，详见第 二 章） |
| 组号与组名 | 第 11 组 • 训练组 |
| 人数与投入 | 5 人；每周合计 6–8 小时（组内自行排班，建议每周固定 2 次） |
| 赛道 | 零样本基线 → UWStereo 微调复现 → 训练策略与超参优化 |
| 上游依赖 | 开局资产包：500 对图对、UWStereo 子集、初步模型 v0 与零样本基线（你的对照组开局就有）；第 10 组的环境与 4080 时段 |
| 下游依赖 | 第 12 组评测你的权重；实验四联调时接受第 9 组新配比数据续训 |
| 工作目录 | `tasks/u11_train/`（主线 `code/` 只读；改超参只改自己目录下的配置副本） |
| 硬件要求 | **需 4080 时段**（向第 10 组预约，时段与用量记入台账）；日常分析与画图用 CPU 电脑即可 |

先记住你们组最重要的一条战场纪律：**先基线，后微调**。没有零样本基线，一切“提升了多少”都无法证明——这句在实验三还会重申，因为它是最常见的翻车点。另一条是公共资源纪律：训练闸门与 4080 时段（见防糊弄指南），管住的是全队的发动机。求助通道照旧：组内互助 → 研究助理 → 项目负责人，逐级上升；求助必须附完整命令与完整输出——训练报错往往在几千行日志的最后一行。

## 11.2　项目背景与你的位置

第 二 章讲过：锚链 ROV 检查视频靠双目立体匹配变成三维信息，方案是 FoundationStereo 当教师打伪标签、学生模型学习；500 对自建立体图对已在开局资产包里，前期还用公开数据初步微调出基线模型 v0。FoundationStereo 零样本已经很强，但水下浑浊、低照度仍是它的软肋——这正是你们的战场。第 9 组喂你数据，第 10 组给你机房与时段，第 12 组给你打分：你们站在流水线正中央，

全项目唯一有权改模型权重的组。你们的产出很朴素：一版权重、一组曲线、一张“零样本 vs 微调”的对比表——但对全队来说，这就是主战成果本身。

## 11.3　核心任务

三步赛道一张表，随后说人话。

表 11.2　第 11 组三步赛道：干什么与产出什么

| 阶段 | 干什么 | 产出 |
|---|---|---|
| 零样本基线 | 预训练权重直接在验证集上推理打分，立起“赛前成绩” | 基线 EPE 数字 |
| 微调复现 | 用 UWStereo 数据短程微调，复现“零样本 → 微调”对照 | 微调权重 v1 + 曲线 |
| 超参优化 | 学习率/主干 lr 比例/数据增强中选 1–2 项做 2–3 组对比 | 对比表 + 结论 |

训练的全部旋钮都集中在 `configs/train_uw_fresh.yaml` 一个文件里：**lr**（学习率，每一步沿梯度方向挪多大步子）、**bs**（batch size，一次一起送进网络多少对图）、**iters**（总迭代步数）与 **epoch**（全量数据完整过几遍）。实验一的任务，就是让 agent 逐项解释它们，你核对后写成“配置说明书”——训练之前先读懂自己要拧的每一个钮。训练本身就是一个环（下图）：取一批数据、前向、算 loss、反向、更新，转一圈进一步；你们这学期的工作就是盯住这个环，让它在 4080 上转得稳、转得值。

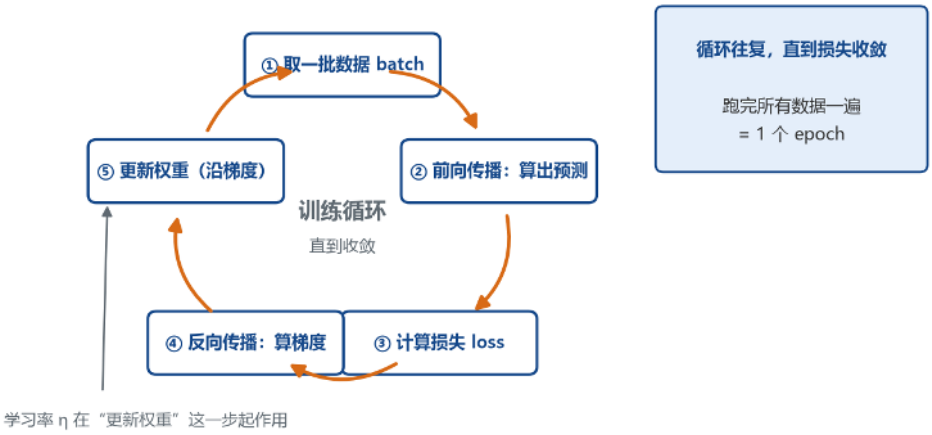



图 11.1　训练循环流程环：取 batch → 前向 → 算 loss → 反向 → 更新——实验二你要亲手让它转起来

**超参数：训练开始前就要拍板定好的数**

与权重（训练中学出来的数）相对，**超参数**是训练前你定下的数。学习率太大，loss 震荡甚至起飞；太小，磨蹭不前（见下图三条曲线）。batch size 影响显存占用与训练稳定性；epoch 太多会过拟合——训练 loss 还在降、验证指标反而变差，第 3 讲的双曲线就是它。改超参的正确姿势：**一次只动一个，其余全部不动**，否则说不清谁起了作用。

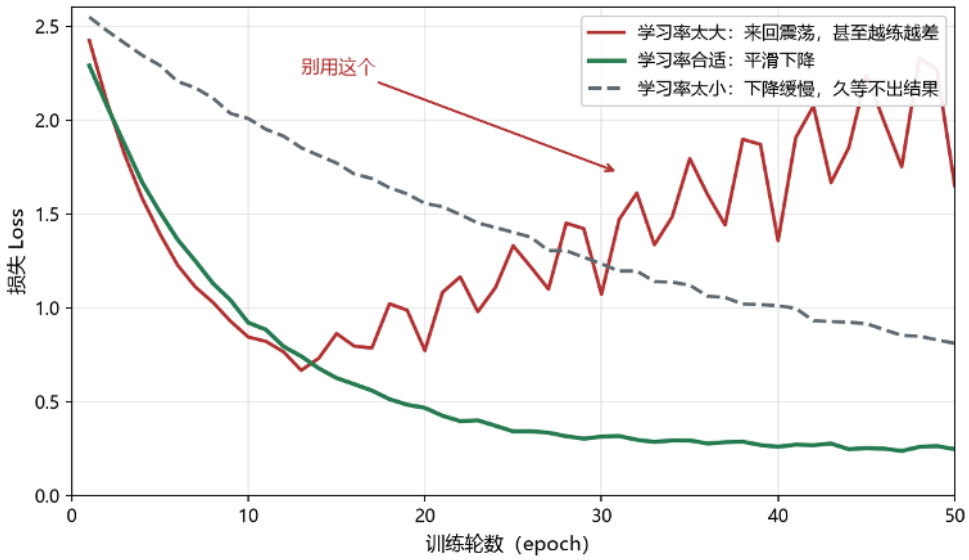



**图 11.2**　三条学习率下的 loss 曲线：太大震荡上行、合适平滑下降、太小磨蹭不前——这是实验四对比实验的直觉底图

**项目连线：训练组的活与第一篇哪几讲有关**

方法论底座是第 3 讲——loss、学习率、训练/验证双曲线、过拟合与早停，全部在那讲打过样，你们现在要在真实大模型上重演一遍。格局来源是第 7 讲——“预训练大模型 + 领域微调”正是那一讲迁移学习思想的当代放大版：FoundationStereo 在海量数据上学会了“看立体”，你只用几千对水下数据把它的眼光“掰”向浑浊水域。

## 11.4　第 19 周周五提交清单

提交线：2027年 1 月 22 日（第 19 周周五）完全提交离场。你的组交齐下表各件，每样都与第 三 章的总清单模板对应；《四组交接总表》由水下四组共填，本表最后一行就是总表里你组的“交接内容”栏。

表 11.3　第 11 组提交清单（2027 年 1 月 22 日）

| 交付物 | 格式 | 说明 |
|---|---|---|
| 微调权重 v1 | 权重文件 + 快照 | 经闸门批准的短程微调权重（不超过 5 个 epoch），附配置快照与随机种子 |
| 训练曲线包 | PNG + CSV | loss 与验证 EPE 双曲线，零样本基线同图对照 |
| 对照报告 | PDF，2–4 页 | “零样本 vs 微调”验证 EPE 对比、结论与下一步建议 |
| 配置说明书 | PDF/Markdown | `configs/train_uw_fresh.yaml` 逐项人话解释（实验一产出，随报告定稿） |
| 小组项目报告 | PDF，8–15 页 | 按第 三 章模板：摘要/背景/方法/实验/结论/分工声明 |
| 证据包 | zip | 四次实验各一包，命令 + 日志 + 产物三件套，日志为终端完整输出 |
| AI 使用声明 | 一页 | 哪些事 AI 做了、人做了什么核验，逐条列清 |
| 交接内容 | 《四组交接总表》一行 | 权重 v1 与训练曲线交第 12 组系统评测；最优超参结论交第 9 组配比实验参考 |

## 11.5　AI 分工与人工必做

分工原则一句话：AI出草稿，人出签字——每一行训练日志、每一个验证 EPE，签字的人负全责。

表 11.4　第 11 组 AI 分工与人工必做

| AI可以自动做 | 人必须亲自做 |
|---|---|
| 逐项解释配置文件与训练代码 | 在 4080 上实跑训练，全程盯 loss 曲线 |
| 写画曲线脚本与日志解析脚本 | **提交闸门申请（epoch 数、启动时机）并等批准** |
| 解读训练异常（loss 为 NaN、不下降） | 先测零样本基线，再开微调 |
| 起草配置说明书、报告与闸门申请文字 | 核对 AI 解释与配置原文逐条一致 |
| 建议超参对比方案 | 在权重交付单与证据包上签字，对数字担责 |

## 11.6　防糊弄指南与满分标准

**防人工糊弄指南（第 11 组版）**

先背全项目通用的纪律红线：**(1) 原始数据/帧/视频不上云盘、不进在线 AI（保密!）**；**(2) 主线 `code/` 只读不改**——改超参只改 `tasks/u11_train/` 下的配置副本；**(3) 不直接联系甲方**。本组再加三条：**(4) 训练闸门（epoch 数、启动时机）必须经小助手批准，不得自行开长训**——闸门保护的是公共显卡时段，也是你自己的成绩；**(5) 没做零样本基线就开微调 = 对照失效**，该节点返工；**(6) 拿训练 loss 冒充验证 EPE= 造假**，一经查实该节点成绩作废。

**防 AI 糊弄指南（第 11 组版）**

**(1) AI 产出必须实跑验证**——第四条纪律红线：它写的启动脚本、给的配置改法，只有你跑出来的日志说了算。**(2) AI 爱报“最佳超参”**——那是别人数据集上的经验，你的数据要自己扫，抄来的最优往往是你的最差。**(3) AI 说“loss 降了就是训好了”不可信**——不看验证集的回答一律存疑，训练 loss 降、验证 EPE 升是常见的假繁荣。**(4) AI 贴的命令在你的环境里可能路径对不上、版本对不上**——以你的 `conda list` 与目录结构为准。

按课程建议方案，实验一不计分，实验二至四各 10 分共 30 分，平时作业 20 分（十二讲章后习题十二选二）；期末成果满分 50 分（教师可调）。成果怎么拿满，看下面这张 50 分制细则。

**表 11.5**　第 11 组成果满分标准（50 分制）

| 分值 | 项目 | 怎么拿满 |
|---|---|---|
| 20 | 功能达成 | 实验一配置说明书 + 逐项核对（3 分）；实验二数据加载 + 前向/反向冒烟（5 分）；实验三零样本基线 + 闸门内短程微调 + 对照曲线（10 分）；实验四 1–2 项超参 2–3 组对比（2 分） |
| 10 | 证据完整 | 四次实验证据包三件套齐全、可复现；缺一件扣 5 分 |
| 8 | 报告质量 | 组报告按模板 8–15 页，对照结论有基线数据支撑、曲线图齐全 |
| 7 | 协作与台账 | 闸门申请与批准记录齐全；4080 时段台账无缺失；求助记录规范 |
| 5 | 加分 | 如：数据增强消融实验；按浑浊度分层训练子集并给出 EPE 对比；冻结主干只调局部的对照 |

**存档点思维：checkpoint 是你的“读档”**

游戏打 boss 前先存档，训练同理：每个 epoch 存一次 checkpoint（权重存档点）。断电、爆显存、超参试错，都能从最近的存档读档重来，而不是从零再训。实验三要求逐 epoch 存档，就是给你们上的这份保险。

## 11.7　四次实验安排

四个节点与全班同频（实验一 2026 年 10 月 12 日、实验二 11 月 2 日、实验三 11 月 23 日、实验四 12 月 21 日），每场都有本组的命令级动作。先看总览表，再逐场展开；每场按“目标—步骤—验收证据—常见坑”执行：步骤照抄就能跑，证据一栏就是成绩。总则：实验一至三是离线自研——资产包开局就发到每组手里，谁也不用等谁；实验四是合并联调——四条赛道的成果在同一台机器、同一套考卷上会师。

**表 11.6**　第 11 组四次实验总览

| 节点 | 日期 | 本组动作 | 核心验收证据 |
|---|---|---|---|
| 实验一 | 10月 12 日 | agent 逐项解释训练配置，人工核对成“配置说明书” | 说明书 + 核对记录 |
| 实验二 | 11 月 2 日 | 一个 batch 过模型，前向/反向冒烟 | 形状打印 + loss + 显存快照 |
| 实验三 | 11 月 23 日 | 零样本基线 + 闸门内短程微调对照 | 基线数字 + 批准记录 + 曲线 |
| 实验四 | 12 月 21 日 | 选 1–2 项超参做 2–3 组对比 | 配置快照 + EPE 对比表 |

### 11.7.1　实验一（10 月 12 日）：把 yaml 读成人话

**目标**：用 agent 逐项解释 `configs/train_uw_fresh.yaml`（lr/bs/iters/epoch），人工核对后写成“配置说明书”——训练前先读懂每一个要拧的钮。

**步骤**：(1)把配置文件拷贝到 `tasks/u11_train/`（主线只读，改也只改副本）；(2) 让 agent 逐项解释：每个键控制什么、默认值多少、往大往小各会怎样；(3) 逐项与代码核对——在主线代码里找到读取该键的那一行，行号写进说明书，才算核对通过；(4) 给每项配一句人话（如 lr 是“步子迈多大”，bs 是“一口吃几对图”）；(5) 汇成 1–2 页说明书。

**验收证据**：配置说明书 + 逐项核对记录（含代码行号与“对/错/存疑”标注）。

**常见坑**：AI 把 iters 和 epoch 混为一谈——一个数的是“总共走多少步”，一个数的是“全量数据走几遍”，两者关系要自己在说明书里算清；只解释不核对——AI 的解释看着都对，一对代码就露馅；说明书抄成名词字典，没有“动了会怎样”的判断。

### 11.7.2　实验二（11 月 2 日）：让一个 batch 走完全程

**目标**：跑通数据加载（UWStereo一个 batch 过模型）+ 前向/反向冒烟——确认“数据能进、loss 能算、梯度能传”，为实验三扫清路障。

**步骤**：(1) 在第 10 组环境里激活 conda 环境；(2) 数据加载冒烟：取一个 batch，打印左目/右目/视差真值的形状与数值范围；(3) 前向一步：输出视差形状与初始 loss 记进日志；(4) 反向一步：优化器 `step()` 成功、loss 可反传、显存占用正常（`nvidia-smi` 拍快照）；(5) 全程输出存档。

**验收证据**：batch 形状打印、初始 loss 数字、反向与优化器更新成功日志、显存快照。

**常见坑**：数据路径大小写或后缀不对，dataloader 静默空转——“在训了”其实是空转，先确认日志里读到了多少对图；数据里有坏样本（全黑图、尺寸异常）导致 loss 为 NaN——找第 9 组要体检报告提前排查；忘了 `model.train()`，跑通的是推理不是训练。

### 11.7.3　实验三（11 月 23 日）：先立基线，再开微调（本组主力）

**目标**：复现“零样本 → UWStereo 微调”对照：短程微调（不超过 5 个 epoch，经小助手闸门批准）+ 零样本基线，对比验证 EPE，画出 loss/EPE 曲线。

**步骤**：(1) **先测零样本基线**：资产包里的预训练权重在第 9 组验证集上的验证 EPE，资产包还带了一版前期初步微调的基线模型 v0，一并测了当参照——这是对照的“赛前成绩”，没有它后面一切提升无从谈起；(2) 填闸门申请：epoch 数（不超过 5）、启动时段、配置快照，**经小助手批准后方可开训**；(3) 训练中盯 loss 曲线，每个 epoch 存一次 checkpoint；(4) 训完在同一验证集上测验证 EPE，与基线同表对比；(5) 画 loss 与验证 EPE 双曲线，标出最优 checkpoint。

**验收证据**：基线 EPE 数字、闸门批准记录、完整训练日志、基线/微调对比表、双曲线图。

**常见坑**：**没做零样本基线就开微调**——“提升 0.4”从此无人能证，流程上先基线后微调，顺序不能反；**epoch 超闸门**——多训的每一分钟都是借全队的，超闸门按违纪处理；验证集混入训练场景——开工前拿第 9 组的划分清单逐条核对；中途断连没存 checkpoint，只能从头再来。

### 11.7.4　实验四（12 月 21 日）：一次只动一个旋钮

**目标**：在学习率、主干 lr 比例、数据增强中选 1–2 项，做 2–3 组对比实验，回答“哪个设置更适合水下数据”——本场同时是四组合并联调：采纳第 9 组新配比数据续训、与第 12 组统一口径总评测，成果进《四组交接总表》。

**步骤**：(1) 选 1–2 个旋钮（如 lr 减半对加倍、开/关颜色抖动增强）；(2) 每组配置存快照并固定随机种子，**其余参数与实验三完全一致**；(3) 各组走同样的闸门流程跑短程微调，分别记验证 EPE；(4) 填对比表：配置差异、基线 EPE、各组 EPE、训练耗时，写不少于 100 字的结论——哪个好、好多少、你猜为什么。

**验收证据**：各组配置快照与训练日志、EPE对比表、结论段。

**常见坑**：一次动两个旋钮——差异说不清归谁，对比作废；没固定随机种子——随机波动比超参差异还大，白比；贪多开四组对比——一组都没跑完，踏实的两组结论比仓促的四组值钱。

# 第 12 组　水下双目 · 评测组

**本章地图**

**讲什么**：第 12 组任务书——“指标体系 → 误差分桶 → 误差可视化 → 阈值扫描”四步赛道怎么走：EPE/bad-N 指标怎么造准、怎么给全队出成绩单、怎么找模型的软肋。**为什么值得读**：训练组说“我微调好了”，数据组说“我的配比更优”，部署组说“FP16 更快”——谁说了算？你说了算，拿数字说话。**学完你能**：定义并亲手验证 EPE/bad-N/D1 指标，搭一条“推理 → 视差 → 指标”的一键评测链路，用分桶归因找出模型最差的那一块短板。

一句话概括你们组：全项目的裁判组。裁判的公信力只来自一样东西：**口径统一、可复现**——同一把尺子量所有人，量几次结果都一样。你们给出的基线数字表就是全队的成绩单，你们找出的最差桶就是下一轮改进的路标。

## 12.1　组档案

**表 12.1**　第 12 组档案

| | |
|---|---|
| 项目 | 水下双目（项目二，中海油 HYSY123 FPSO 锚链 ROV 检查视频的双目立体感知，详见第 二 章） |
| 组号与组名 | 第 12 组 • 评测组 |
| 人数与投入 | 5 人；每周合计 6–8 小时（组内自行排班，建议每周固定 2 次） |
| 赛道 | EPE/bad-N 指标 → 误差分桶（浑浊度/距离）→ 误差可视化 → 阈值扫描 |
| 上游依赖 | 开局资产包：零样本基线与初步模型 v0（第一周就有两版可评）；第 10 组推理链路；第 9 组验证集与划分清单 |
| 下游依赖 | 全队：基线数字表是成绩单；阈值 $\tau$ 推荐值回灌伪标签过滤；实验四联调出四组总成绩单，最差桶归因交第 9/11 组 |
| 工作目录 | `tasks/u12_eval/`（主线 `code/` 只读，见防糊弄指南） |
| 硬件要求 | 批量推理需 4080 时段（向第 10 组预约）；指标计算与画图用 CPU 电脑即可 |

先说清楚你们与别的组的本质区别：别的组交付“东西”（数据集、环境、权重），你们交付“判断”。判断要服人，必须满足两条——**口径固定**（分辨率、子集、随机种子全部写进评测配置，交表时一并交出）与**可复现**（任何人按你的配置重跑，数字必须对得上）。求助通道照旧：组内互助 → 研究助理 → 项目负责人，逐级上升，求助必须附完整命令与完整输出。

## 12.2　项目背景与你的位置

第 二 章讲过：锚链 ROV 检查视频靠双目立体匹配变成三维信息，方案是 FoundationStereo 教师–学生伪标签闭环。注意“过滤”两个字的分量——教师打的标签不是全收，要按置信分过一道闸门，闸门的松紧（阈值 $\tau$）就是你们的实验四要定的参数。你在流水线的终点，也是反馈的起点：向

终点方向，你给零样本与微调两版权重出成绩单，让第 11 组的劳动被量化；向起点方向，你的 $\tau$ 推荐值与最差桶归因回流给数据组和训练组，驱动下一轮改进。裁判失职的代价是全队迷路：指标造错了，后面所有人的“提升”都是幻觉。

## 12.3　核心任务

四步赛道一张表，随后说人话。

**表 12.2**　第 12 组四步赛道：干什么与产出什么

| 阶段 | 干什么 | 产出 |
|---|---|---|
| 指标体系 | 定义并实现 EPE、bad-1/2/4px、D1，手工验证 1 对样例 | 指标白皮书 |
| 全链路 | 跑通“推理 → 视差输出 → 指标计算”，串成一键命令 | 一键评测脚本 |
| 系统评测 | 微调权重 vs 零样本，每模型评不少于 50 对 | 基线数字表 |
| 扫描与归因 | 阈值 $\tau$ 扫描画保留率–精度曲线；按浑浊度/视差分桶找最差桶 | 扫描报告 + 归因 |

先把尺子造准（三个指标见下面的知识盒），再跑链路，再系统评测，最后干裁判最有价值的事——**指出模型差在哪**。为什么非要分桶？看下图：视差越大物体越近，远处的锚链视差小、误差天然大——如果只报一个总 EPE，近处的好成绩会把远处的灾难平均掉，**总分会撒谎，分桶才见真章**。按视差大小（近/中/远）与浑浊度（低照度、浑浊代理指标）分桶之后，“最差桶”三个字才有着落——实验四的人工归因就盯住它。

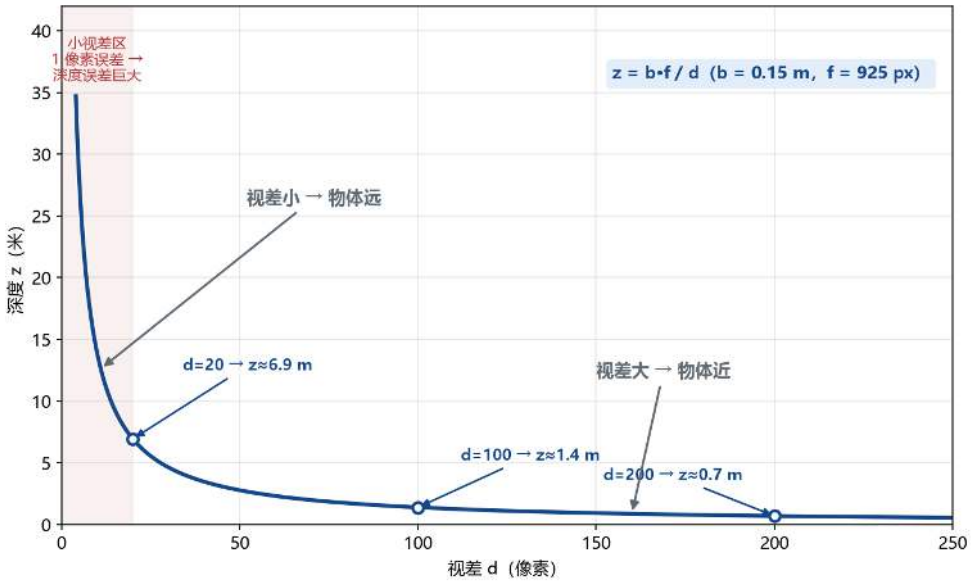



**图 12.1**　视差与深度成反比：视差越大物体越近——这正是评测要按视差分桶的道理，远处视差小、天然难，必须单独看

**三把尺子把“差多少”说清楚**

**EPE**（End-Point Error，端点误差）：所有像素上“预测视差与真值之差的绝对值”取平均，单位是像素，越小越好——这个方向性是全组成绩单的生死线，永远别写反。**bad-N**：误差超过 N 个像素的像素占比，bad-1/2/4px 就是 N 取 1、2、4，管的是“错得明显”的像素比例。**D1**：误差既超过 3 个像素、又超过真值 5% 的像素占比，专抓“错得离谱”的硬伤。四个数一起看：EPE 管平均，bad-N 与 D1 管最差的那部分像素。

**项目连线：评测组的活与第一篇哪几讲有关**

评估方法论出自第 3 讲——训练集/验证集的划分、泛化与过拟合、“指标方向性”这些评估的底层常识，那一讲全都打过样；你们是它在立体匹配任务上的执行人。EPE 算的是逐像素误差的平均，数学上就是那一讲的平均损失思想；“分桶看指标”则是它的分层版——平均会撒谎，分布不会。

## 12.4　第 19 周周五提交清单

提交线：2027年 1 月 22 日（第 19 周周五）完全提交离场。你的组交齐下表各件，每样都与第 三 章的总清单模板对应；《四组交接总表》由水下四组共填，本表最后一行就是总表里你组的“交接内容”栏。

**表 12.3**　第 12 组提交清单（2027 年 1 月 22 日）

| 交付物 | 格式 | 说明 |
|---|---|---|
| 指标白皮书 | PDF，2–4 页 | EPE/bad-1/2/4px/D1 的定义、公式、算法与出处；1 对样例的手工验算记录 |
| 基线数字表 | CSV + 表格 | 零样本 vs 微调 v1，每模型评不少于 50 对：总分 + 分桶数字 + 评测配置 |
| 阈值扫描报告 | PDF，2–4 页 | 阈值 $\tau$ 扫描的保留率–精度曲线、推荐 $\tau$ 与理由；最差桶归因结论 |
| 误差可视化集 | 图册 | 不少于 20 张误差可视化图，每张附人工归因“错在哪” |
| 小组项目报告 | PDF，8–15 页 | 按第 三 章模板：摘要/背景/方法/实验/结论/分工声明 |
| 证据包 | zip | 四次实验各一包，命令 + 日志 + 产物三件套，日志为终端完整输出 |
| AI 使用声明 | 一页 | 哪些事 AI 做了、人做了什么核验，逐条列清 |
| 交接内容 | 《四组交接总表》一行 | 基线数字表交全队与教师；推荐 $\tau$ 交伪标签过滤环节；最差桶归因交第 9/11 组 |

## 12.5　AI 分工与人工必做

分工原则一句话：AI出草稿，人出签字——成绩单上每一个数字、每一句归因，签字的人负全责。

表 12.4　第 12 组 AI 分工与人工必做

| AI可以自动做 | 人必须亲自做 |
|---|---|
| 起草指标白皮书的文字与公式排版 | **看误差可视化图不少于 20 张并写“错在哪”的人工归因（AI 不可代看）** |
| 写指标计算脚本与分桶统计脚本草稿 | 挑 1 对样例手工验算 EPE，与脚本数字对账 |
| 画曲线、热图与误差可视化图 | 固定随机种子，核对分桶结果可复现 |
| 解释异常数字（为何这一桶 EPE 特别高） | 逐条核对 AI 公式与代码实现一致 |
| 建议阈值扫描的档位设计 | 在基线数字表上签字，对每个数字担责 |

## 12.6　防糊弄指南与满分标准

**防人工糊弄指南（第 12 组版）**

先背全项目通用的纪律红线：**(1) 原始数据/帧/视频不上云盘、不进在线 AI（保密！）**——评测用的甲方帧同样涉密，出门的只有统计数字；**(2) 主线 `code/` 只读不改**，评测脚本与产出都放 `tasks/u12_eval/`；**(3) 不直接联系甲方**。本组再加三条：**(4) 只报总 EPE 不报分桶 = 没评完**——总分会被简单样本撑好看，裁判不能只报总分；**(5) 中途更换评测口径（换分辨率、换子集）不声明 = 成绩单作废**；**(6) 代跑必须留运行者姓名与时间**，谁跑的谁负责。

**防 AI 糊弄指南（第 12 组版）**

**(1) AI 产出必须实跑验证**——第四条纪律红线：它给的指标实现、算出的任何数字，只有你脚本跑出来的那份算数。**(2) AI 写指标公式常有下标与符号幻觉**——逐条与你核对过出处的白皮书对账，公式的每个绝对值、每个平均范围都要有着落。**(3) AI“帮你看图”写的归因只能当候选答案**——必须你亲眼看图确认后才可采用，AI 不可代看是本组铁律。**(4) AI 报的指标数必须与你的脚本输出一致**，对不上以脚本为准、以你的终端为准。

按课程建议方案，实验一不计分，实验二至四各 10 分共 30 分，平时作业 20 分（十二讲章后习题十二选二）；期末成果满分 50 分（教师可调）。成果怎么拿满，看下面这张 50 分制细则。

表 12.5　第 12 组成果满分标准（50 分制）

| 分值 | 项目 | 怎么拿满 |
|---|---|---|
| 20 | 功能达成 | 实验一指标白皮书 + 手工验算对账成功（5 分）；实验二“推理 → 视差 → 指标”一键链路（5 分）；实验三每模型不少于 50 对的基线数字表（含分桶）（8 分）；实验四 $\tau$ 扫描曲线 + 最差桶归因（2 分） |
| 10 | 证据完整 | 四次实验证据包三件套齐全、可复现；缺一件扣 5 分 |
| 8 | 报告质量 | 组报告按模板 8–15 页，评测口径交代清楚、结论有基线与分桶数据支撑 |
| 7 | 协作与台账 | 为三组提供统一评测口径并有记录；4080 时段台账无缺失；求助记录规范 |
| 5 | 加分 | 如：自动浑浊度分桶的量化指标被数据组采用；评测封装成一条命令被三组复用；提出并论证一个新指标 |

## 12.7　四次实验安排

四个节点与全班同频（实验一 2026 年 10 月 12 日、实验二 11 月 2 日、实验三 11 月 23 日、实验四 12 月 21 日），每场都有本组的命令级动作。先看总览表，再逐场展开；每场按“目标—步骤—验收证据—常见坑”执行：步骤照抄就能跑，证据一栏就是成绩。总则：实验一至三是离线自研——资产包开局就发到每组手里，谁也不用等谁；实验四是合并联调——四条赛道的成果在同一台机器、同一套考卷上会师。

**表 12.6**　第 12 组四次实验总览

| 节点 | 日期 | 本组动作 | 核心验收证据 |
|---|---|---|---|
| 实验一 | 10月 12 日 | agent 写指标白皮书，实跑 10 对样例，手工验证 1 对 | 白皮书 + 验算记录 |
| 实验二 | 11 月 2 日 | 跑通“推理 → 视差 → 指标”全链路 | 一键脚本 + 批量日志 |
| 实验三 | 11 月 23 日 | 微调 vs 零样本系统评测，每模型不少于 50 对 | 基线数字表（总分 + 分桶） |
| 实验四 | 12 月 21 日 | 阈值 $\tau$ 扫描 + 分桶归因 | 曲线 + 分桶表 + 20 张归因 |

### 12.7.1　实验一（10 月 12 日）：先把尺子造准

**目标**：用 agent 写“指标白皮书”（EPE/bad-1/2/4px/D1 的定义与算法），实跑 10 对样例算指标，并手工验证其中 1 对——尺子不准，后面全错。

**步骤**：(1) 让 agent 逐个指标写：定义、公式、取值范围、适用场景；(2) 逐条与主线代码及文献核对——公式里每个下标、每个绝对值、每个平均范围都要有出处；(3) 对 10 对样例实跑指标脚本，存全部输出；(4) 挑 1 对，把视差真值图缩小成 $10 \times 10$，用计算器把 100 个像素的绝对误差加起来除以 100，与脚本算出的 EPE 对账；(5) 白皮书定稿，写明每条定义的出处。

**验收证据**：指标白皮书、10 对样例的指标输出、手工验算过程（可拍照存档）。

**常见坑**：**指标算错符号（EPE 越小越好）**——报告里把方向写反，整套结论全盘皆输；手工验算跳过——脚本错一个绝对值，往后所有数字都是错的，而没人知道；AI 把 D1 写成另一种变体定义——以你核对过的文献为准，白皮书里注明出处。

### 12.7.2　实验二（11 月 2 日）：一键打通“推理 → 视差 → 指标”

**目标**：跑通全链路：对 1 对图完成推理、保存视差、算出指标，再扩展成 10 对批量版，为实验三的大规模评测铺路。

**步骤**：(1) 在第 10 组环境里对 1 对 UWStereo 图跑推理，视差图存盘（float 格式）；(2) 把视差画成可视化色图，肉眼检查是否合理（锚链处视差应连续、不应有大片空洞）；(3) 对同一对算 EPE/bad-1/2/4px/D1，核对中间文件；(4) 把三步串成一条命令（脚本放 `tasks/u12_eval/`），一键完成“推理 → 保存 → 算分”；(5) 跑 10 对批量版，存完整日志。

**验收证据**：一键脚本、单对与批量日志、视差可视化样例。

**常见坑**：视差图存成 uint8 被截断——大于 255 的视差全变 255，指标全错，存 float 才稳；左右目路径接反——视差符号全反，EPE 大得离谱反而“提醒”了你；可视化用非线性配色，小误差看着触目惊心，干扰后面的人工归因。

### 12.7.3　实验三（11 月 23 日）：两版权重同场比武，每模型不少于 50 对

**目标**：对微调权重 vs 零样本权重做系统评测，每模型评不少于 50 对，建立全队基线数字表——这是你们给全队出的成绩单。

**步骤**：(1)从资产包取验证集划分与两版权重（零样本基线 + 前期初步模型 v0——开局即评，不必等人）；第 11 组联调新版本实验四补测；(2) 固定随机种子与评测口径，分辨率、对齐方式、子集名单全部写进评测配置并随表交出；(3) 每模型评不少于 50 对，记 EPE/bad-1/2/4px/D1 总分；(4) 按场景与视差范围分桶统计；(5) 汇成基线数字表，并给每版权重写一句“体检结论”。

**验收证据**：评测配置、两模型批量日志、基线数字表（总分 + 分桶 + 体检结论）。

**常见坑**：**分桶没固定随机种子**——每次分桶结果都不一样，两次报告数字对不上，裁判自己先打架；两模型用了不同分辨率——同场比武变成各跑各道，对比无效；50 对里混进训练集场景——拿第 9 组的划分清单逐条核对；只报平均不报分桶——总分会撒谎。

### 12.7.4　实验四（12 月 21 日）：扫阈值、找最差桶、当个好裁判

**目标**：伪标签过滤阈值 $\tau$ 扫描（保留率–精度曲线）；按浑浊度/视差大小分桶，找出最差桶并归因——这是你对伪标签闭环最直接的贡献；本场同时是四组合并联调：补测第 11 组新权重、统一口径出四组总成绩单，成果进《四组交接总表》。

**步骤**：(1) 对伪标签置信分扫描 $\tau$：每档记保留率与精度（在有真值的子集上算 EPE）；(2) 画保留率–精度曲线，给出推荐 $\tau$ 与理由——太松，噪声标签混进训练集；太紧，粮食不够吃；(3) 按浑浊度代理指标与视差大小（近/中/远）分桶，找出 EPE 最差的桶；(4) **看不少于 20 张误差可视化图，人工写“错在哪”归因（AI 不可代看）**；(5) 归因结论与推荐 $\tau$ 交给第 9/11 组与伪标签过滤环节。

**验收证据**：$\tau$ 扫描表与曲线、分桶数字表、不少于 20 张的人工归因记录。

**常见坑**：归因写成“效果不好”四个字——合格的归因要具体到“远距离 + 低照度 + 细杆状结构”这一档；曲线只画不选——没有推荐 $\tau$ 等于没扫；老毛病再犯：符号方向与随机种子，本组两大惯犯，交表前组内互查一遍。

# 第 13 组　数据基座 · 数据结构化组

**本章地图**

**讲什么**：第 13 组任务书——把几份含表格与公式的工程文档 PDF，做成“机器可查”的结构化 JSON：表格抽成表、公式转成 TeX、每个元素登记锚点，外加一份敢说真话的质量报告。**为什么值得读**：真实系统 1,001 页拆出 484 张表、19,893 个锚点，靠的就是结构化流水线；你做的是它的教学版——规模缩小、方法论一份不打折，而且你的产物是下游问数组的口粮。**学完你能**：独立跑通“PDF 进、JSON 出”的小流水线；说得清锚点与“如实条款”为什么是命根子；亲手训一个“这块是表格、公式还是正文”的分类小模型。

一句话概括你们组：整条数据基座流水线的“拆迁办”。下游各组都在“新房”里干活——问数、查图谱、抓造假——而你们负责把 PDF 这栋老房子拆成登记造册的梁、柱、管线：表格是表格、公式是公式，每块碎片都带“户口”。拆迁不干净，下游全喝西北风。

## 13.1　组档案

**表 13.1**　第 13 组档案

| | |
|---|---|
| 项目 | 数据基座治理（项目三，工程资料结构化与本地数据查询系统，详见第 三 章） |
| 组号与组名 | 第 13 组 • 数据结构化组 |
| 人数与投入 | 5 人；每周合计 6–8 小时（组内自行排班，建议每周固定 2 次） |
| 负责的能力 | B 类需求（文档结构恢复）的教学版——文档结构化小流水线 |
| 上游依赖 | 项目提供的样例包：3–5 份含表格与公式的工程文档 PDF 片段 |
| 下游依赖 | 第 14 章（第 14 组）：一句话问数 demo 吃你的结构化产物——中期前他们用演示数据兜底，你交付后无缝切换 |
| 工作目录 | `tasks/d01_struct/`（样例 PDF 放 `data/`，产物放 `out/`，脚本放 `src/`） |
| 硬件要求 | 普通笔记本 CPU 即可：解析库吃 CPU，实验三的小模型只有几百个参数，无 GPU 毫无压力 |

先解释一个词：**教学版**。真实系统的 B 类能力是工业级的——跨页表格合并、公式 TeX 源码、19,893 个锚点（第 三 章的数字）；你们做的是从它派生的缩小版：样例只有几份 PDF，流水线只跑几十页，但**每一个设计决策都和真实系统同构**——抽不出来怎么办？登记回退。抽出来没把握怎么办？如实标注。这套纪律练熟了，换到任何一个真实项目都能直接上手。另外把求助通道抄在工位上：组内互助 → 项目助教 → 导师，逐级上升；求助必须附完整命令与完整输出，只发一句“跑不通”等于没发。

## 13.2　项目背景与你的位置

第 三 章讲过来龙去脉：甲方的资料分两座山——126 份 CSV 共 3.1 亿行的运行数据是“洪水”，6 份共 1,001 个物理页的工程资料是“迷宫”。洪水难在量，迷宫难在结构：表格动辄跨页、公式不是图片就是扫描件、PDF 的逻辑页码和物理页还对不上。系统对这座迷宫下的第一步功夫，就是你们对口的 **B 类需求：文档结构恢复**——把 1,001 页还原成“表格是表格、公式是公式、每一段都能指回原文”的结构化数据，最终交出 484 张表、272 段公式源码、19,893 个锚点，并且大大方方登记了 39 处回退区域、138 张列宽证据不足的表、“人工确认计数 0”。

你在流水线的最上游。看下图，四条链里最左边那条“结构化”就是你们的工地：文档从你们手里变成带锚点的结构化数据，问数链才有 SQL 可查，门控链才有知识可抽，下游的每个数字才能“点回原件”。第 三 章缺口表上属于你们的两行——39 个局部回退区域、138 张列宽证据不足的表——就是这学期要正面强攻的城墙。

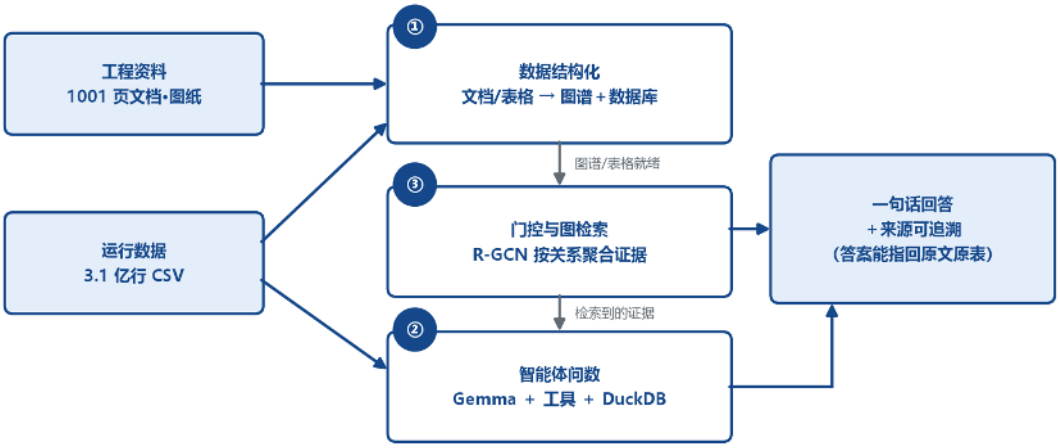



**图 13.1**　数据基座三大能力流水线上的第 13 组：你们站在最上游的“结构化”一环，把文档与表格变成带锚点的结构化数据——下游的智能体问数、门控与图检索、攻防，全都踩在你们打的地基上

## 13.3　核心任务

四件事一张表看明白，随后逐件说人话。

**表 13.2**　第 13 组四项任务：干什么与产出什么

| 任务 | 干什么 | 产出 |
|---|---|---|
| 表格抽取 | 用解析库把每页表格抽成行列，跨页表格登记“可拼接线索”，合并单元格关系尽量保留 | 表格 CSV/JSON |
| 公式转 TeX | 找出公式区域，转写成 LaTeX 片段（数学式子可复制、可检索的那套记号） | 公式片段 |
| 锚点登记 | 每个元素登记“哪份文件、第几个物理页、页面哪个区域”的出处坐标 | 锚点表 |
| 质量报告 | 抽了多少表、多少公式、多少回退区域、多少存疑，**如实登记**，学真实系统的“如实条款” | 质量报告 PDF |

逐件展开。**表格抽取**是主力活：`pdfplumber` 靠文字坐标与线框认表，`PyMuPDF` 快而稳，`camelot` 对带线框的表更准——三选二装通即可（实验二细说）。抽完不是终点：跨页的表会被拆成两截，你至少要登记“本表在第 37 页结束、第 38 页有同列数的疑似延续”，能不能拼回来如实说明。**公式转 TeX**：样例里的公式先框出区域，简单式子可用规则或在线大模型转写，转写结果必须人工抽核——大模型最爱在上下标上幻觉。**锚点登记**是全组的地基，格式看下面这段产物示例。**质量报告**是本组的灵魂：真实系统“人工确认计数 0”都敢写，你们抽错几张、哪几页抽不出来，也要一张不落地说清楚。

```
{
  "element_id": "T-0037",
  "type": "table",
  "source_file": "sample_calc_book.pdf",
  "physical_page": 37,
  "bbox": [84.2, 120.5, 511.6, 702.9],
  "caption": "表 3-2 系泊缆索设计参数",
  "n_rows": 12, "n_cols": 5,
  "merged_cells": [[2, 0, 2, 1]],
  "cross_page_hint": "same_cols_continue_p38",
  "confidence": 0.86,
  "flag": "column_width_uncertain"
}
```

逐字段念一遍：`physical_page` 是物理页（不是目录里的逻辑页码，封面目录造成的错位在这里不起作用）；`bbox` 是页面坐标，四个数圈出这张表在页面上的位置——锚点的“户口地址”；`merged_cells` 记合并单元格；`confidence` 与 `flag` 就是“如实条款”的落点：机器没把握的，如实写出来，不许悄悄装没问题。下游看到 `flag` 就知道这张表用之前该查什么——**敢标存疑，别人才敢用**。

**项目连线：拆房子的活与第一篇哪几讲有关**

第 4 讲的“表示”思想：版面上的表格与公式，最终都要变成机器好算的格式——JSON 与向量一样，都是“表示”；第 6 讲的训练循环：实验三你训元素分类小模型时，取批、前向、算损失、反向、更新，一步不少；第 11 讲的注意力与在线模型 API：实验四的疑难表格二次抽取，本质是提示词工程 +few-shot，正是大模型 API 的标准用法。

## 13.4　第 19 周周五提交清单

提交线：2027年 1 月 22 日（第 19 周周五）完全提交离场。你的组交齐下表七样，每样都与第 三 章的总清单模板对应。

表 13.3　第 13 组提交清单（2027 年 1 月 22 日）

| 交付物 | 格式 | 说明 |
|---|---|---|
| 结构化流水线代码包 | zip，含 README | 环境说明 + 一键脚本，别人拿到能跑；三选二解析库的选型理由写进 README |
| 结构化产物与锚点表 | JSON/CSV | 样例包全部 PDF 的产物：表格、公式片段、锚点表，字段按上一节示例齐全 |
| 质量报告 | PDF，2–4 页 | 含如实条款：抽了多少表/公式、多少回退区域、多少存疑、人工核对计数，如实登记 |
| 分类小模型与曲线 | 模型文件 + 图 | 实验三的“表格/公式/正文”分类器、loss 曲线、混淆矩阵 |
| 小组项目报告 | PDF，8–15 页 | 按第 三 章模板：摘要/背景/方法/实验/结论/分工声明 |
| 证据包 | zip | 四次实验各一包，命令 + 日志 + 产物三件套，日志为终端完整输出 |
| AI 使用声明 | 一页 | 哪些事 AI 做了、人做了什么核验，逐条列清 |

## 13.5　AI 分工与人工必做

分工原则一句话：AI出草稿，人出签字——质量报告里每一个数字，签字的人负全责。

表 13.4　第 13 组 AI 分工与人工必做

| AI可以自动做 | 人必须亲自做 |
|---|---|
| 写解析与批处理脚本草稿 | **逐页核对 ≥20 页（AI 不可代看）** |
| 生成标注建议与预标结果 | **标注 ≥200 条终审**：AI 可预标，人逐条过目签字 |
| 解释报错信息、给排查建议 | 本机实跑每一条命令 |
| 画统计图、起草报告与质量报告文字 | 逐条核对 AI 结论（在代码里找到对应行才算数） |
| 建议提示词与 few-shot 样例写法 | 在产物与质量报告上签字，对数字担责 |

## 13.6　防糊弄指南与满分标准

**防人工糊弄指南（第 13 组版）**

**(1) 核对记录不许复制粘贴凑数**——“逐页核对 20 页”就是 20 个不同的物理页，同一页抄十遍不行。**(2) “抽查合格”必须写清抽查了哪几页**——页码区间、核对结论、发现的问题，三样缺一不可；只写“抽查合格”四个字等于没查。**(3) 编造抽取数量 = 该节点成绩作废**——质量报告里的每个数字都要能与 JSON 产物对上账，对不上就是作废，没有商量。**(4) 代跑必须留运行者姓名与时间**，谁跑的谁负责。

**防 AI 糊弄指南（第 13 组版）**

**(1) AI 说“抽取成功”不算数**——成功不成功只看 JSON 内容：行数、列名、数值，打开文件逐项对。**(2) 大模型幻觉的表头必须人工纠错**——它会给缺列名的表“好心”补一个，补得头头是道，原文里根本没有。**(3) AI 报的页码与坐标要抽验**——它没真读过你的 PDF，位置信息十有八九是编的。**(4) 禁止把 AI 的回答直接粘进质量报告**——报告里只许出现你亲手核对过的内

容。

按课程建议方案，实验一不计分，实验二至四各 10 分共 30 分，平时作业 20 分（十二讲章后习题十二选二）；期末成果满分 50 分（教师可调）。成果怎么拿满，看下面这张 50 分制细则。

**表 13.5**　第 13 组成果满分标准（50 分制）

| 分值 | 项目 | 怎么拿满 |
|---|---|---|
| 15 | 流水线功能 | 实验二 10 页流水线跑通且产物字段齐全（8 分）；实验四疑难表格二次抽取并出对比报告（7 分） |
| 10 | 质量报告与如实度 | 每个数字与产物对得上账；回退、存疑、人工核对计数如实登记，无一件虚报 |
| 8 | 小模型训练 | 标注 ≥200 条且终审留痕；loss 曲线齐全；混淆矩阵与宏平均有分析 |
| 7 | 报告质量 | 组报告按模板 8–15 页，方法讲得清、图表齐全、结论有数据支撑 |
| 5 | 协作与台账 | 排班执行有记录；标注分工与终审留痕；求助记录规范 |
| 5 | 加分 | 如：跨页表格自动合并逻辑并用数字证明有效；为扫描件接 OCR 兜底并对比效果；把流水线做成一键复跑工具 |

## 13.7　四次实验安排

四个节点与全班同频（实验一 2026 年 10 月 12 日、实验二 11 月 2 日、实验三 11 月 23 日、实验四 12 月 21 日），但每场都有本组的命令级动作。先看总览表，再逐场展开；每场按“目标—步骤—验收证据—常见坑”执行：步骤照抄就能跑，证据一栏就是成绩。

**表 13.6**　第 13 组四次实验总览

| 节点 | 日期 | 本组动作 | 核心验收证据 |
|---|---|---|---|
| 实验一 | 10月 12 日 | agent 解读表格抽取原理，3 份样例各抽一页人工核对 | 解读 + 核对记录一页 |
| 实验二 | 11 月 2 日 | 解析环境三选二装通，10 页流水线跑通 | 版本输出 + 日志 +JSON |
| 实验三 | 11 月 23 日 | 自造标注 ≥200 条，训元素分类小模型 | 训练日志 +loss 曲线 |
| 实验四 | 12 月 21 日 | 在线 API 疑难表格二次抽取，对比规则法 | 提示词 + 对比表 + 账单 |

### 13.7.1　实验一（10 月 12 日）：让大模型讲原理，你拿样例当裁判

**目标：**配好大模型编程助手的 agent 模式，让它讲清 `pdfplumber` 表格抽取的原理，你用 3 份样例 PDF 各抽一页，人工核对它说得对不对、抽得准不准。

**步骤：**(1) 按课堂演示装好编程助手并切换到 agent 模式；(2) 安装并验证：`pip install pdfplumber`，再跑 `python -c "import pdfplumber; print(pdfplumber.__version__)"`；(3) 让 agent 回答三问：`extract_tables` 是靠什么把一张表从页面里“认”出来的（线框？文字对齐？）、跨页表格为什么会被拆开、什么情况下会抽错；(4) 写一个十行小脚本对 3 份样例各抽一页，结果存 JSON；(5) 打开 PDF 逐格对

照，标注“对/错/漏”，并把 agent 的解释逐条与源码或官方文档对照，至少纠正它一处错误。

**验收证据：**一页“AI解读 + 人工核对”记录（含 AI 原话、你的核对结论、至少一处纠正），加 3 页抽取结果与逐格核对记录。

**常见坑：扫描件让 pdfplumber 全军覆没**——页面上没有文字层，`extract_tables` 返回空列表，先确认样例是文字版，扫描件要留到 OCR 兜底（可做加分项）；agent 会“脑补”原理——它讲的算法细节可能来自教科书而不是这个库，一切以源码为准；只数抽了几行不看列名对不对——列名错了，整张表都是错的。

### 13.7.2　实验二（11 月 2 日）：解析环境装通，10 页流水线跑通

**目标：**本地部署 Python 解析环境（PyMuPDF / pdfplumber / camelot 三选二装通），批量跑通 10 页的“抽取—锚点—登记”流水线，产出第一批结构化 JSON。

**步骤：**(1) 建独立虚拟环境，避免污染；(2) 三选二安装：`pip install pymupdf pdfplumber`，想上 camelot 再装 `pip install "camelot-py[cv]"`——它依赖 Ghostscript，装不上就果断放弃，三选二达标即可；(3) 写 `extract_pipeline.py`：输入一个 PDF，逐页跑“表格抽取 → 公式区域记录 → 锚点登记”，输出上一节格式的 JSON；(4) 对含 10 页的样例跑通全流程；(5) 打印统计：几张表、几段疑似公式、几条锚点、几个回退区域。

**验收证据：**两个库的版本号输出、10 页流水线的完整终端日志、JSON 产物路径。

**常见坑：中文路径报错**——部分库对非 ASCII 路径支持不佳，样例统一放英文相对路径（如 `data/sample1.pdf`）最省事；camelot 卡安装一整晚——三选二就是给你留的后路，别跟依赖较劲；跨页表格被拆成两张——先如实登记“可拼接线索”，不强行合并，自动合并是加分项不是门槛。

### 13.7.3　实验三（11 月 23 日）：自己造数据，训一个“认版面”的小模型

**目标：**用你自己造的标注数据（≥200条）训练一个“这块是表格/公式/正文”的三分类小模型，亲眼看 loss 下降——这是“深度网络的模型训练实践”在本组的落点。

**步骤：**(1) 造数据：把样例 PDF 切成块（如按行块），给每块打标签，写一页标注规范（什么算公式块、跨页怎么算）再动手；AI 可预标，人逐条终审，凑够 200 条且三类都有像样的样本；(2) 造特征：块高、字号、数字密度、线框数等十几维手工特征；(3) 训一个两层小网络（第 2 讲的反向传播就够了），训练/验证**按页切分**——同一页的块不许一边进训练一边进验证；(4) 记录每个 epoch 的 loss，画曲线；(5) 算验证集总体准确率之外，再算宏平均与混淆矩阵，看哪两类最容易混。

**验收证据：**标注集 + 标注规范一页、训练日志、loss 曲线、混淆矩阵与宏平均。

**常见坑：类别不平衡**——正文块远多于公式块，只看总体准确率会被“全判正文”骗过，宏平均才见真章；特征量纲不一忘了归一化，loss 降得莫名其妙地慢；两人标注打架——先写规范、抽 20 条交叉核对一致性，再批量标。

### 13.7.4　实验四（12 月 21 日）：在线大模型救疑难表格，规则法当对照组

**目标：**接入在线大模型 API，对规则法抽不动的疑难表格做“二次抽取”（提示词工程 +few-shot 样例），与规则法同题对比，输出对比报告与结论。

**步骤：**(1) 从实验二的产物里挑出抽错/抽不出的疑难表格 ≥5 张，登记“难在哪”（无线框？合并单元格？跨页？）；(2) 设计提示词：描述表格结构，配 1–2 个人工核对过的正例做 few-shot；(3) 调 API 抽取，结果逐格人工核对；(4) 填对比表：规则法 vs 在线增强法，各列“列名对/数值对/合并

关系对”的格数、耗时、花费；(5) 写不少于 100 字的结论：什么时候用规则、什么时候值得上在线 API。

**验收证据：**提示词全文、API调用记录（次数与花费账单）、对比表、结论段。

**常见坑：API key 硬编码进代码包**——用环境变量，交包前全文搜一遍 key；把整份文档发给在线 API——只发疑难页的文本或截图，能少发不多发；只报成功案例不报失败——在线法也有翻车的题，如实写进对比表，这份诚实正是你们组的看家本领。

# 第 14 组　数据基座 · 智能问答与图检索组

**本章地图**

**讲什么**：第 14 组任务书——两件事：搭一个“一句话问数”小 demo（本地小模型 + 演示库 + 白名单工具 +SQL 安全检查），再在一个 50–100 节点的小知识图谱上训两层 R-GCN 做链路预测。**为什么值得读**：真实系统 500 题回归考卷拿下 497 分的问数能力、缺口表上 G8/G9 两条（正式图谱未发布、R-GCN 未联合验收）都和你们对口——demo 虽小，“模型只提出 SQL，程序执行 SQL”的军规一条不少。**学完你能**：用 Ollama 跑通本地问数并录屏留证；说得清白名单与只读检查怎么防住 SQL 注入；独立完成一次小规模图神经网络训练并报出 AUC。

一句话概括你们组：数据基座流水线的“窗口柜员”兼“关系侦探”。柜员这半句：用户说人话，你们把人话变成 SQL，查出的每个数字都带出处；侦探这半句：把设备、部件、故障、文档连成一张网，让“A 和 B 什么关系”这种关键词检索答不了的题有据可查。

## 14.1　组档案

**表 14.1**　第 14 组档案

| | |
|---|---|
| 项目 | 数据基座治理（项目三，工程资料结构化与本地数据查询系统，详见第 三 章） |
| 组号与组名 | 第 14 组 • 智能问答与图检索组 |
| 人数与投入 | 5 人；每周合计 6–8 小时（组内自行排班，建议每周固定 2 次） |
| 负责的能力 | A 类需求（一句话问数）与图检索的教学版——问数小 demo + R-GCN 链路预测小实验 |
| 上游依赖 | 第 13 章（第 13 组）的结构化产物；中期前用项目提供的演示数据兜底（四个查询模型的精简版 CSV，各几百行） |
| 联动依赖 | 第 15 章（第 15 组）：其造假样本可进你们的测试题库（对抗联动，见核心任务） |
| 工作目录 | `tasks/d02_agent_kg/`（`demo/` 放问数服务，`kg/` 放图谱实验，`tests/` 放 30 题） |
| 硬件要求 | 8GB 内存笔记本即可：1.5B–3B 的量化模型 CPU 能跑、有 GPU 更快；R-GCN 小图 CPU 可训 |

先解释“演示数据兜底”：第 13 组的结构化产物要到期中才交付，但你们的流水线一天都不能等——项目提供 `fpso_navigation` 等四个查询模型的精简版 CSV（各几百行，真数据的缩样）供你们先建库跑通；第 13 组产物到位后，同一套代码换个数据源就能切换。这套“先兜底、后切换”的打法，与真实系统“先沙箱、后生产”的节奏同源。求助通道照旧抄在工位上：组内互助 → 项目助教 → 导师，求助附完整命令与完整输出。

## 14.2　项目背景与你的位置

第 三 章讲过真实系统怎么答“2023 年 11 月 18 日 20:27 的尾吃水是多少”：大脑是本地部署的 Gemma 12B（上下文 8,192、温度 0、并发 1），手里只有 8 个白名单工具；它提出的每条 SQL 都要过 SQLGlot 检查、Wren 规划，最后由 DuckDB 执行——一句话军规：**模型选择工具，程序检查边界，数据库计算数值**。凭这套镣铐，500 题独立标准答案的回归考卷拿了 497 分。图检索那边，系统把知识连成图谱，用 R-GCN 按关系类型聚合邻居、给证据排序；但缺口表也如实写着：**G8**——正式知识图谱尚未发布；**G9**——R-GCN 尚未与业务数据联合验收。这两条就是你们的进场口。

你在流水线的中段，管两条链：**问数链**——人话进来、SQL起草、检查放行、数据库执行、答案带引用出去；**图检索链**——实体进来、沿关系扩邻居、模型排序、证据出去。看下图，真实系统把“理解问题 → 调用工具 → 核查依据 → 回答与来源”画成一个八步闭环，证据不足就回头重排步骤；你们的 demo 就是把这个闭环缩小装进笔记本——**环节一个不砍，规模全线下调**。

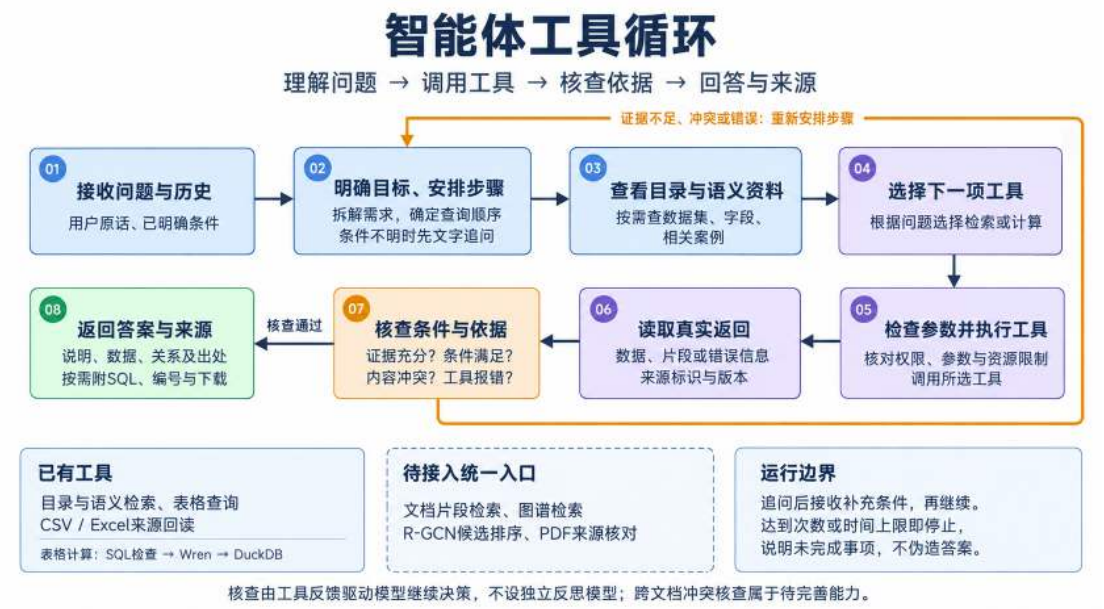



**图 14.1**　真实系统的智能体工具循环：理解问题 → 调用工具 → 核查依据 → 回答与来源的八步闭环，证据不足、冲突或报错就重新安排步骤；第 14 组的问数 demo 就是它的教学版——环节一个不砍，规模全线下调

图片来源：本项目《总体方案与验收计划》文档配图

## 14.3　核心任务

两块任务一张表看明白，随后逐块说人话。

**表 14.2**　第 14 组两块任务：干什么与产出什么

| 任务 | 干什么 | 产出 |
|---|---|---|
| 问数小 demo | 本地小模型 + 演示库 +3 个白名单工具函数 +SQL 安全检查，一句话问数、答案带引用 | 可跑的 demo |
| R-GCN 实验 | networkx 建 50–100 节点小图谱（设备-部件-故障-文档四类实体），训两层 R-GCN 做链路预测 | notebook+ 指标 |

**问数小 demo** 四件套。第一件，**本地小模型**：用 Ollama 跑 Qwen2.5-1.5B（内存紧张就 1.5B，

宽裕可上 3B）或 Gemma2-2B，**温度设 0**——同一问题两次回答必须一致，这是第 三 章 A7 条款的教学版。第二件，**演示库**：SQLite 或 DuckDB 装下四个精简查询模型。第三件，**三个白名单工具函数**：`query_rows`（按条件查明细）、`agg_stats`（求和/平均/计数/最值）、`describe_columns`（列名与含义说明）——模型手里只有这三件，没有任意执行。第四件，**SQL 安全检查**：只读（拦下 INSERT/UPDATE/DELETE/DROP）、列名白名单（库里有这列才放行）；铁律再次上墙：**模型只提出 SQL，程序执行 SQL**——模型碰不到数据库，碰数据库的只有被检查过的语句。

**R-GCN 实验**。先用 networkx 造一张 50–100 个节点的小图：设备、部件、故障、文档四类实体，“设备–装有–部件”“部件–发生–故障”“故障–记载于–文档”几类关系边；再转成 PyTorch Geometric 的图数据，训一个**两层 R-GCN**（关系图卷积网络：按关系类型分别聚合邻居信息的图神经网络），做链路预测——遮住图上一部分边让模型猜“这两点之间该不该有条边”，用 AUC 或 Hit@10 报成绩。真实系统用三层 R-GCN 给检索证据排序（第 三 章），你们这两层小网练的就是同一门手艺，看下图：

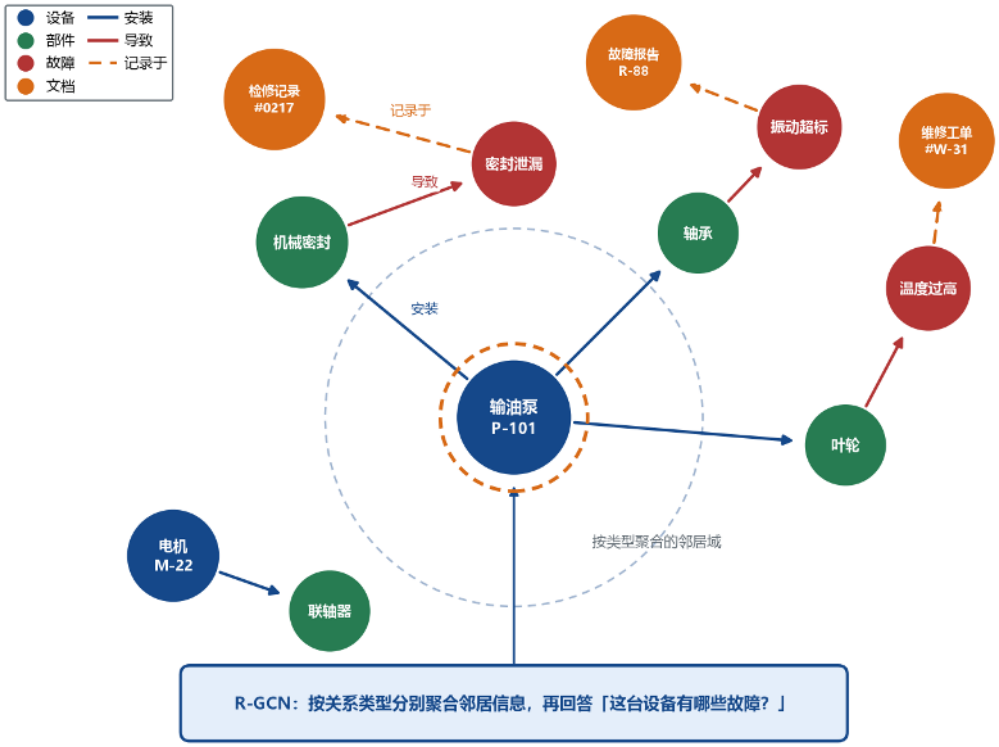



**图 14.2**　R-GCN 按关系类型聚合邻居：不同颜色的边是不同类型的关系，节点“信谁”取决于它沿各类关系连着谁；第 14 组在 50–100 节点的小图上训两层 R-GCN 做链路预测，练的就是这套聚合的本事

**项目连线：问数与图谱，第一篇都有老巢**

第 2 讲：反向传播——R-GCN 再“图”，它也是一层层前向、把误差一层层反传的网络；第 8 讲：本地部署与量化——Ollama 就是“小模型搬回家”的现成工具；第 11 讲：注意力与 agent——“查询-回答-引用”的循环正是 agent 范式；第 3 讲：评估——30 题怎么算对错、AUC 怎么读，全按它的方法论来。

另外记住你们的**对抗联动**：第 15 组会拿造假样本“污染”你们的测试题库——在 30 题里混进几条似是而非的假记录，看你们的安全检查与引用核对能不能接住。这个联动不是找茬，是两边都涨本事的合练。

## 14.4　第 19 周周五提交清单

提交线：2027年 1 月 22 日（第 19 周周五）完全提交离场。你的组交齐下表七样，每样都与第 三 章的总清单模板对应。

表 14.3　第 14 组提交清单（2027 年 1 月 22 日）

| 交付物 | 格式 | 说明 |
|---|---|---|
| 问数 demo 代码包 | zip，含 README 与启动脚本 | 一键启动脚本 + 建库脚本 + 演示库；README 写清模型下载与运行步骤，别人拿到能跑 |
| 30 题测试集 + 标准答案 | 表格 | **标准答案独立编写、先行复审**——系统生成的 SQL 不得反过来充当标准答案；含 ≥10 题原创边界题 |
| R-GCN 实验 notebook | ipynb+ 曲线 | 建图、训练、评估全流程可复跑，loss 曲线与 AUC/Hit@10 齐全 |
| 本地 vs 在线对比报告 | PDF，2–4 页 | 30 题上准确率/延迟/成本三列对比，附“何时用本地、何时用在线”建议 |
| 小组项目报告 | PDF，8–15 页 | 按第 三 章模板：摘要/背景/方法/实验/结论/分工声明 |
| 证据包 | zip | 四次实验各一包，命令 + 日志 + 产物三件套，日志为终端完整输出 |
| AI 使用声明 | 一页 | 哪些事 AI 做了、人做了什么核验，逐条列清 |

## 14.5　AI 分工与人工必做

分工原则一句话：AI出草稿，人出签字——30 题的每份标准答案，编写的人负全责。

表 14.4　第 14 组 AI 分工与人工必做

| AI可以自动做 | 人必须亲自做 |
|---|---|
| 写三个工具函数与建库脚本草稿 | **独立编写 30 题标准答案（先于测试，AI 不得反推）** |
| 生成测试问题草稿与边界题点子 | **逐题核对回答与引用**：数字点回数据库原文 |
| 解释报错信息、给排查建议 | 运行 demo 并录屏留证 |
| 起草对比报告与 README 文字 | 自己出一页 SQL 注入测试题打自己的防线 |
| 建议提示词与输出格式写法 | 在测试集与指标上签字，对每个数字担责 |

## 14.6　防糊弄指南与满分标准

**防人工糊弄指南（第 14 组版）**

**(1) 标准答案不许让 AI 反推**——先独立写答案、后跑系统，顺序反了就是“考生批卷”；系统生成的 SQL 永远不能反过来充当标准答案。**(2) 30 题里至少 10 题是原创边界题**——缺条件该追问的、说“前五条”就该只给五条的、查无结果该老实报空的；全挑软柿子捏不算测试。**(3) 编造成功率 = 该节点成绩作废**——每题的对错明细都要能对账，报“成功率 90%”就得拿得出 30 行明细。**(4) 代跑必须留运行者姓名与时间**，谁跑的谁负责。

**防 AI 糊弄指南（第 14 组版）**

**(1) 本地模型温度必须设 0**——不设 0，同一个问题两次回答俩样，可追溯性无从谈起。**(2) “模型说查到了”不算数**——必须点回数据库原文：数字、时间戳、行数，逐项对上才算对。**(3) 别把 API key 写进代码包**——用环境变量，交包前全文搜一遍。**(4) AI 写的 SQL 安全检查可能有漏**——比如没拦住 PRAGMA、ATTACH 这类旁门语句，注入测试要自己出题自己防，别迷信 AI 写的“万能拦截”。

按课程建议方案，实验一不计分，实验二至四各 10 分共 30 分，平时作业 20 分（十二讲章后习题十二选二）；期末成果满分 50 分（教师可调）。成果怎么拿满，看下面这张 50 分制细则。

**表 14.5**　第 14 组成果满分标准（50 分制）

| 分值 | 项目 | 怎么拿满 |
|---|---|---|
| 15 | 问数 demo | 实验二建库 + 一句话问数跑通并录屏（8 分）；白名单工具与只读检查齐备、通过自设注入测试（7 分） |
| 10 | R-GCN 实验 | 建图规范、训练真实完成、loss 曲线齐全，报出 AUC 或 Hit@10 并有分析 |
| 8 | 测试集与评估 | 30 题独立标准答案 + 至少 10 题原创边界题；每题对错明细可对账 |
| 7 | 报告质量 | 组报告按模板 8–15 页，方法讲得清、图表齐全、结论有数据支撑 |
| 5 | 协作与台账 | 排班执行有记录；录屏与明细归档完整；求助记录规范 |
| 5 | 加分 | 如：给 demo 加“缺条件追问”并演示；加语义检索提升表名/列名命中率；对 R-GCN 做关系类型消融实验 |

## 14.7　四次实验安排

四个节点与全班同频（实验一 2026 年 10 月 12 日、实验二 11 月 2 日、实验三 11 月 23 日、实验四 12 月 21 日），但每场都有本组的命令级动作。先看总览表，再逐场展开；每场按“目标—步骤—验收证据—常见坑”执行：步骤照抄就能跑，证据一栏就是成绩。

**表 14.6**　第 14 组四次实验总览

| 节点 | 日期 | 本组动作 | 核心验收证据 |
|---|---|---|---|
| 实验一 | 10月 12 日 | agent 调用你的工具函数，走通“查询-回答-引用” | 完整循环日志/录屏 |
| 实验二 | 11 月 2 日 | Ollama 部署 + 建库 + 一句话问数跑通 | 版本输出 + 问答录屏 |
| 实验三 | 11 月 23 日 | R-GCN 链路预测真训练 | 训练日志 +loss 曲线 +AUC |
| 实验四 | 12 月 21 日 | 30 题本地 vs 在线对比 | 明细表 + 对比表 + 账单 |

### 14.7.1　实验一（10 月 12 日）：当一回 agent，走通“查询-回答-引用”

**目标：**配好大模型编程助手的 agent 模式，让它调用你们写好的工具函数完成一次“查询-回答-引用”——亲身体验 agent 循环，理解“模型提要求、程序干活、结果回填”怎么转起来。

**步骤**：(1) 按课堂演示装好编程助手并切换到 agent 模式；(2) 先把三个工具函数写出草稿（AI 起草、人核对函数签名与 docstring——每个参数什么含义、返回什么 JSON，写不清 agent 就不会用）；(3) 把工具挂给 agent（或手写一个最简循环：模型输出工具名与参数 → 程序执行 → 结果回填对话）；(4) 提一个“2023 年 11 月 18 日 20:27 的轴向振动”式问题，要求 agent 给出“答案 + 所用 SQL+ 数据来源行号”三件套；(5) 把完整循环记录归档。

**验收证据**：一次完整“查询-回答-引用”的日志或录屏，含工具调用参数与返回原文。

**常见坑**：agent不调工具直接“背答案”——提示词里写明“必须调用工具函数作答，禁止凭记忆回答”；工具函数没有 docstring，agent 瞎猜参数含义，调用连连报错；历史对话不回填，agent 第二轮就忘了第一轮查到了什么。

### 14.7.2　实验二（11 月 2 日）：本地小模型上岗，一句话问数跑通

**目标**：本地部署 Ollama 小模型，建好演示库，白名单工具与 SQL 安全检查齐备，一句话问数端到端跑通。

**步骤**：(1) 装 Ollama，`ollama pull qwen2.5:1.5b`，**下载完再调用**（`ollama list` 能看到模型才算完）；(2) 把四个精简 CSV 导入 SQLite/DuckDB，记录各表行数；(3) 接模型：提示词限定“只输出一条 SQL，不输出别的”，温度 0；(4) 写 `check_sql()`：只读 + 列名白名单两道闸，先拦后跑；(5) 跑通三个样例问题（明细/聚合/列名说明各一），全程录屏。

**验收证据**：`ollama list` 输出、建库行数统计、三个样例问答的录屏（含 SQL 与查询结果）。

**常见坑**：Ollama 模型没下载完就调用，报错还以为代码有问题；模型输出带解释文字不是纯 SQL，解析当场崩——输出格式要在提示词里钉死；**SQL 注入测试题自己没防住**——自己写一页攻击用例（改数据的、查系统表的、多语句的）逐条打自己的闸门；温度忘设 0，两次回答不一致被同学当场抓包。

### 14.7.3　实验三（11 月 23 日）：小图上真训练——两层 R-GCN 做链路预测

**目标**：在自建的小知识图谱上真实训练两层 R-GCN 做链路预测，画 loss 曲线、报出 AUC 或 Hit@10——这是“深度网络的模型训练实践”在本组的落点。

**步骤**：(1) networkx 建图：50–100 个节点、四类实体、至少三类关系，登记节点数/边数/各类关系条数；(2) 转成 PyTorch Geometric 的图数据，节点特征做简单编码并**归一化**；(3) 切边：训练边与验证边按比例分开，口径写清楚；(4) 训两层 R-GCN：正样本是被遮住的真边，负采样配对假边，二分类训练；(5) 每个 epoch 记 loss，画曲线；(6) 验证集上报 AUC 或 Hit@10。

**验收证据**：建图统计、训练日志、loss 曲线、AUC/Hit@10 数字及其含义解释。

**常见坑**：**节点特征忘了归一化**，训练曲线抖成锯齿；图太小（不足 50 节点）指标虚高到不好意思；负采样全用“隔着十万八千里”的点对，模型学成“看距离猜边”；torch-geometric 与 torch 版本不匹配装不上——按官方版本对应表锁定版本再装。

### 14.7.4　实验四（12 月 21 日）：本地 vs 在线——30 题上一张对比表

**目标**：用在线大模型 API（或在线 embedding 检索）对比本地方案，在固定的 30 题上量出准确率、延迟、成本三笔账，形成“什么时候用本地、什么时候用在线”的结论。

**步骤**：(1) 固定 30 题测试集（样例题 + 原创边界题，标准答案早已独立编写）；(2) 本地方案全量跑一遍，记每题对错与耗时；(3) 在线方案同样提示词再跑一遍，记对错、耗时与花费；(4) 填三列

对比表：准确率/中位延迟/API 花费；(5) 写不少于 100 字的建议：什么问题本地够用、什么问题值得上在线、断网时怎么兜底。

**验收证据：**30题逐题明细表（两方案各一份）、三列对比表、建议段、花费账单。

**常见坑：成功率统计基数写错**——分母是 30，不是“跑完的那些题”，崩掉的题记失败不算弃权；对比时本地与在线用了不同提示词，变量没控住比了白比；只算准确率不记延迟与成本，“在线更好”的结论说不清代价；在线 API 当天抽风没有兜底——留一次重跑时间。

# 第 15 组　数据基座 · 虚假内容识别与对抗训练组

**本章地图**

**讲什么**：第 15 组任务书——办一场缩小版“造假者 vs 监考 vs 判官”对抗：让在线大模型按配额造假，规则 + 小模型当监考，攒出 ≥500 条真假混装数据集，训出真假分类器，报出宏观 F1，并交一份带重叠率数字的泄漏自查报告。**为什么值得读**：真实系统那次把 0.88 改判成 0.74 的审计（第 三 章）证明：不能证明没泄漏的成绩不算成绩。你们组的每一样交付物，都在练“诚实”这门手艺。**学完你能**：按配方指挥 AI 造假并记好账；独立完成规则基线 + 小模型两版分类器；给出带数字的泄漏自查，看得穿“指标虚高”的把戏。

一句话概括你们组：全班唯一“官方允许造假”的小组——你们的业务是让 AI 扮演造假者日夜出题，把你们自己的分类器练成测谎仪；而且测谎仪几斤几两，要按最苛刻的口径如实招报。铁律先立：**大模型只能改说法，永远改不了事实——目标假数值由程序给定**。

## 15.1　组档案

**表 15.1**　第 15 组档案

| | |
|---|---|
| 项目 | 数据基座治理（项目三，工程资料结构化与本地数据查询系统，详见第 三 章） |
| 组号与组名 | 第 15 组 • 虚假内容识别与对抗训练组 |
| 人数与投入 | 5 人；每周合计 6–8 小时（组内自行排班，建议每周固定 2 次） |
| 负责的能力 | 对抗训练 demo 的教学版——缩小版“造假者 vs 监考 vs 判官”攻防闭环 |
| 上游依赖 | 真实系统现成材料：26 个造假家族清单与 2,000 条交付数据集的“配方”（假 1,000 条按易 300/中 400/难 300 分层）——你们做缩小版 |
| 联动依赖 | 第 14 章（第 14 组）：你们的假样本可进它的测试题库（对抗联动） |
| 工作目录 | `tasks/d03_adversarial/`（`fake/` 放造假产物，`dataset/` 放混装数据集，`model/` 放分类器） |
| 硬件要求 | TF-IDF+ 逻辑回归 CPU 秒级出结果；embedding 小网络笔记本可训；API 花费是主要成本，须记账 |

先说清楚你们和真实系统的关系：真实系统那场攻防是“重装部队”——四角色闭环、3,300 道封存考卷、蒸馏库 2,654 条、每轮配额 30 条直接造假 +10 条语言造假 +10 条纯文字造假、最多 5 条润色。你们是“轻装分队”：同一套打法，规模全线缩小——4 轮对抗、≥500 条数据集、≥10 个造假家族。**配额可以缩，但配额制度不能省**：每轮造多少条，开学第一周写进配方文档，之后一轮一记账。求助通道照旧：组内互助 → 项目助教 → 导师，求助附完整命令与完整输出。

## 15.2　项目背景与你的位置

第 三 章的高潮就是你们的原型：门控规则抓得住“没出处、证据打架、对象不清”，却抓不住**语义词造假**——无数值格式错误、无逻辑硬伤、纯粹在事实层面撒谎。系统的回应是四角色闭环：**造假者**（大模型 + 程序给定假数值）批量生产以假乱真的条目，骗过的题目回流给它下一轮“进化”；**监考**不确定必问、放行抽检 10%；**判官**只认数值与单位等硬证据，判“真”须置信度 $\geq 0.75$；**库存管家**管蒸馏库与先例库，蒸馏样本以 3 倍权重回流训练。26 个造假家族按易中难分层，考卷 3,300 道封存待考。

然后是那个你们必须烂熟于心的故事：2026 年 9 月 15 日的审计发现考卷与训练数据事实级重叠 78%、逐字重复 15.7%，旧口径的 0.84–0.88 全部作废，重建考卷后真实水位落在 0.72–0.74——**“不是性能回退，是挤掉水分”**。审计还留下一条尾巴：300 道 OOD 纯文字造假考题 F1 只有 0.071–0.233，系统对语义造假近乎失明；缺口表 **G10**——“难负样本与对抗训练尚未形成成果”——就是你们的进场口。看下图，四角色闭环的每一环，这学期都要在你们手里缩样跑通：

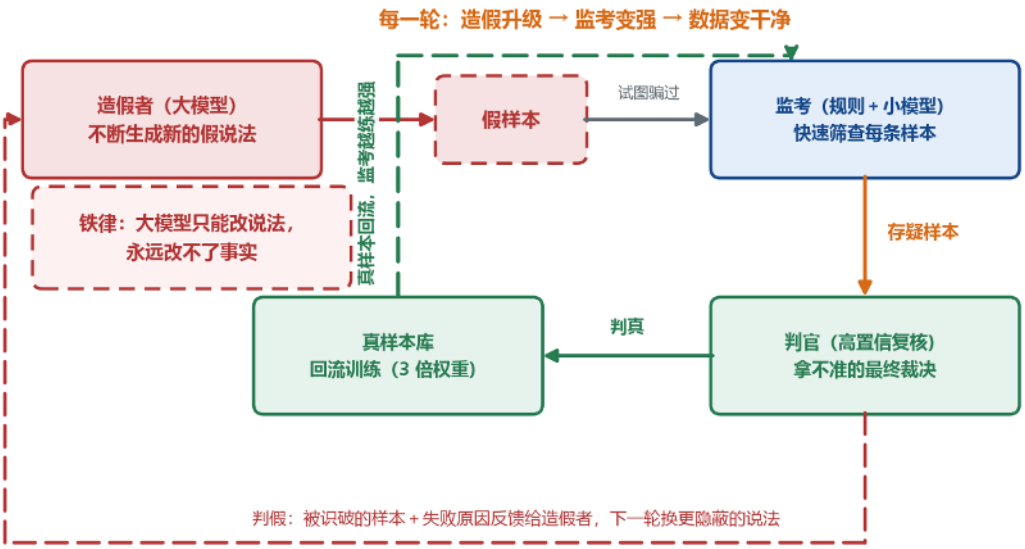



**图 15.1**　真假内容攻防闭环：造假者（大模型 + 程序给定假数值）出题，监考判定与追问，判官凭硬证据复核，蒸馏样本回流再训——第 15 组要把这个闭环缩样跑通四轮，并诚实报告成绩的水分检查

## 15.3　核心任务

六步任务一张表看明白，随后逐步说人话。

**表 15.2**　第 15 组六步任务：干什么与产出什么

| 任务 | 干什么 | 产出 |
|---|---|---|
| 造假者上线 | 在线大模型 API 按配额造假，每周一轮共 4 轮：直接造假/语言造假/纯文字造假三类，假数值由程序给定 | 造假样本流 |
| 监考上线 | 规则（数值越界/单位错/时间戳矛盾）+ 小模型判定，不确定必问、放行抽检 | 判定记录 |
| 数据集 v1 | ≥500 条真假混装：假 250（易 75/中 100/难 75）+ 真 250，洗牌编号，双金标 | CSV |
| 分类器 | TF-IDF+ 逻辑回归基线，加一个 embedding 小网络对比 | 指标表 |
| 泄漏自查 | 算考卷-训练重叠率，**必须给出数字** | 自查报告 |
| 蒸馏回流 | 判对的难样本 3 倍权重回流再训一轮，看宏观 F1 变化 | 前后对比 |

逐步展开。**造假者**的提示词框架是“程序给定目标数值，大模型只负责把说法包装得像真的”——比如程序把“排量 120”改成“135”，模型的任务是给 135 编一圈合情合理的话术；它无权自己发明数字。**监考**先上三条规则（数值越界/单位错/时间戳矛盾），再补一个小模型；拿不准的必须发问，不许硬判。**数据集**的公平性全在细节里：真假各半、难度分层、**随机洗牌**后按 `MIX_0001` 起编号——任何人不能从编号顺序猜出真假。每条样本挂**双金标**，看下面的知识盒。**分类器**两版：TF-IDF+ 逻辑回归是基线（词袋特征 + 线性分类器，第 1–3 讲的老朋友），embedding 小网络是把文字变向量再分类的小模型，两版同题比武。**泄漏自查**是本组的灵魂，呼应 0.88 → 0.74 的审计故事。

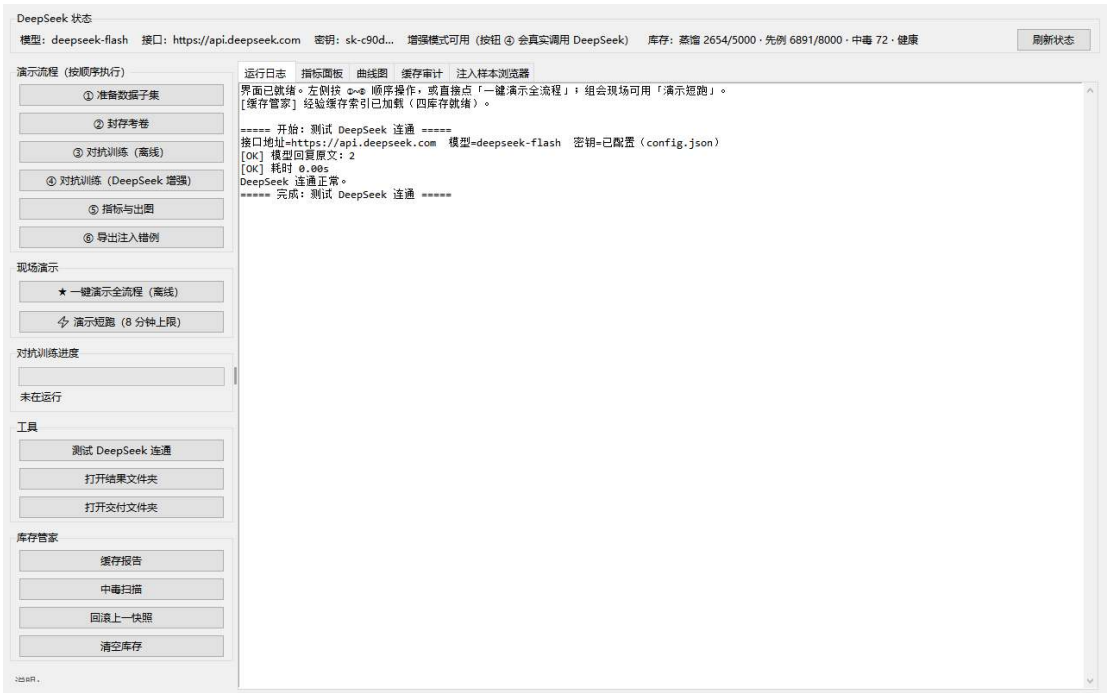



**图 15.2**　真实系统对抗训练 demo 主界面：左侧六步演示流程（准备数据子集 → 封存考卷 → 离线对抗 → DeepSeek 增强 → 指标与出图 → 导出注入错例），右侧运行日志，顶部常驻库存水位——第 15 组的六步任务就是把这流程缩样搬进 `tasks/d03_adversarial/`

图片来源：本项目 demo 截图（E:\中海油数据）

**知识补齐：双金标——客观真假与判定口径分开记**

每条样本挂两个标签：`gold_label` 是**客观真假**——这条内容到底是不是假的，由构造过程保证（程序改的数值，天生为假）；`disc_label` 是**判官当时的判定口径**——监考与判官把它判成了什么。两标分开，才能算出两种错：把假的判成真（漏报，最危险）与把真的判成假（误报，冤枉好人）。只留一个标签，这两种错就糊成一团，复盘时分不清该修监考还是该修判官——真实系统 2,000 条交付数据集正是双金标设计，你们的 v1 数据集照方抓药。

**项目连线：造假的活，第一篇早埋了伏笔**

第 3 讲：评估方法论——宏观 F1 的口径、查准查全的分母，全按它执行；第 6 讲：训练循环——蒸馏回流再训就是“数据回流、模型更新”；第 12 讲：生成对抗思想——造假者与监考就是一对“生成器 vs 判别器”，互相逼强的军备竞赛，赌注从假图像换成了假数据。

## 15.4　第 19 周周五提交清单

提交线：2027年 1 月 22 日（第 19 周周五）完全提交离场。你的组交齐下表七样，每样都与第 三 章的总清单模板对应。

**表 15.3**　第 15 组提交清单（2027 年 1 月 22 日）

| 交付物 | 格式 | 说明 |
|---|---|---|
| 对抗数据集 v1 | CSV，≥500 条 | 假 250（易 75/中 100/难 75）+ 真 250；分层标签 + 双金标（`gold_label/disc_label`）+ 洗牌 `MIX_` 编号 |
| 造假配方文档 | PDF 或 Markdown | ≥10 个家族的缩小版配方：每家族给定义、示例、难度档、提示词模板与配额 |
| 分类器代码与指标表 | 代码包 + 表 | 基线与小模型两版对比：准确率/查准/查全/**宏观 F1**，回流前后各一份 |
| 泄漏自查报告 | PDF，2–3 页 | 重叠率数字 + 检查方法 + 逐条结论——不许“自查通过”一句带过 |
| 小组项目报告 | PDF，8–15 页 | 按第 三 章模板：摘要/背景/方法/实验/结论/分工声明 |
| 证据包 | zip | 四次实验各一包，命令 + 日志 + 产物三件套，含 API 调用与花费记录 |
| AI 使用声明 | 一页 | AI 当造假者的轮次与配额、人工抽检记录索引，逐条列清 |

## 15.5　AI 分工与人工必做

分工原则一句话：AI当反派，人当裁判——每条样本的真假，终审权在人。

表 15.4　第 15 组 AI 分工与人工必做

| AI可以自动做 | 人必须亲自做 |
|---|---|
| **当造假者**（按配额！假数值由程序给定） | **人工抽检 ≥100 条判真假（AI 不可代看）** |
| 起草造假配方与提示词模板 | 核对双金标一致性：`gold_label` 与 `disc_label` 逐条对账 |
| 预标真假标签供监考参考 | 控制 API 花费：一轮一记账，超配额轮次作废 |
| 画指标曲线、起草报告文字 | 封存考卷、隔离回流样本（时间戳为证） |
| 解释报错信息、给排查建议 | 在数据集与自查报告上签字，对每个数字担责 |

## 15.6　防糊弄指南与满分标准

**防人工糊弄指南（第 15 组版）**

**(1) 抽检记录要留痕**——抽了哪 100 条（编号区间）、你判对多少条，两个数字缺一不可；没有数字的“已抽检”等于没抽检。**(2) “全对”要小心**——真有 100% 吗？想想 0.88 是怎么死的：漂亮得反常的指标，第一反应该是怀疑测量，不是庆祝。**(3) 真假失衡或忘了分层洗牌 = 数据集重做**——编号能猜出真假的数据集没有考试资格。**(4) 编造 F1= 该节点成绩作废**——指标表每个数字都要能与训练日志对上账。

**防 AI 糊弄指南（第 15 组版）**

**(1) 铁律：大模型只能改说法，永远改不了事实**——目标假数值由程序给定；造假者“越权自创数值”的样本一律作废，不得入集。**(2) 考卷的 AI 复核结论绝不回流进蒸馏集**——判官看了考卷再出题，等于泄题。**(3) API 用量记账**——每轮调用次数与花费写进台账，超配额的轮次整轮作废。**(4) AI 判真假只有“参考一票”**——它没看过你的数据库，终审永远是人。

按课程建议方案，实验一不计分，实验二至四各 10 分共 30 分，平时作业 20 分（十二讲章后习题十二选二）；期末成果满分 50 分（教师可调）。成果怎么拿满，看下面这张 50 分制细则。

表 15.5　第 15 组成果满分标准（50 分制）

| 分值 | 项目 | 怎么拿满 |
|---|---|---|
| 13 | 数据集与配方 | ≥500 条分层齐全、洗牌可验证、双金标一致（8 分）；配方 ≥10 家族且每家族有可复现示例（5 分） |
| 12 | 分类器与指标 | 规则 +TF-IDF 基线（5 分）；embedding 小网络与回流对比（7 分）；宏观 F1 口径正确、曲线齐全 |
| 8 | 泄漏自查与如实度 | 给出重叠率数字与检查方法；抽检留痕；“人工确认计数”如实登记——虚报一处，本栏清零 |
| 7 | 报告质量 | 组报告按模板 8–15 页，方法讲得清、图表齐全、结论有数据支撑 |
| 5 | 协作与台账 | 排班执行有记录；API 账本无缺失；求助记录规范 |
| 5 | 加分 | 如：自创造假家族并被配方收录；对纯文字造假做专门评测并给出盲区分析；抽检升级为双人独立判 + 仲裁 |

## 15.7　四次实验安排

四个节点与全班同频（实验一 2026 年 10 月 12 日、实验二 11 月 2 日、实验三 11 月 23 日、实验四 12 月 21 日），但每场都有本组的命令级动作。先看总览表，再逐场展开；每场按“目标—步骤—验收证据—常见坑”执行：步骤照抄就能跑，证据一栏就是成绩。

表 15.6　第 15 组四次实验总览

| 节点 | 日期 | 本组动作 | 核心验收证据 |
|---|---|---|---|
| 实验一 | 10月 12 日 | AI 造假 10 条，人工 + 规则各判一遍 | 双份判定 + 失误清单 |
| 实验二 | 11 月 2 日 | 三条规则 +TF-IDF 基线，扩到 100 条 | 两版基线指标表 |
| 实验三 | 11 月 23 日 | ≥500 条数据集训分类器 + 泄漏自查 | 曲线 + 指标 + 重叠率 |
| 实验四 | 12 月 21 日 | 在线判官复核 + 蒸馏回流再训 | 三方案对比 + 回流增益 |

### 15.7.1　实验一（10 月 12 日）：先当一回受害者——AI 造假你来判

**目标**：配好大模型编程助手的 agent 模式并体验角色扮演：让 AI 当造假者生成 10 条假“设备振动记录”，人工与规则各判一遍，记录各自的失误——先挨一刀，再学防身。

**步骤**：(1) 按课堂演示装好编程助手并切换到 agent 模式；(2) 写造假脚本：程序先随机把记录里某个数值乘一个系数（**假数值由程序给定**），再把假数值与原话术一起交给 AI，让它把说法包装得像真的；(3) 生成 10 条假记录，混入 5 条真记录，洗牌编号；(4) 组内每人在不看答案的情况下一条条判，记录判定与把握程度；(5) 写三条规则（数值越界/单位错/时间戳矛盾）让程序再判一遍；(6) 对账：人工错几条、规则错几条、分别错在哪。

**验收证据**：15 条样本 + 人工与规则两份判定记录 + 失误清单。

**常见坑**：造假者太“良心”，假得一望便知——提示词要明确“以假乱真”的口径；人工判题过度自信——先判完再对答案，不许边看边猜；真/假样本不洗牌，顺序可猜，判定记录作废。

### 15.7.2　实验二（11 月 2 日）：规则当第一道闸，词袋小模型当基线

**目标**：把三条规则扩成规则引擎，再把样本扩到 ≥100条，训出 TF-IDF+ 逻辑回归基线分类器——两条基线的成绩就是以后一切改进的对照物。

**步骤**：(1) 用 pandas/sklearn 把数值越界、单位错、时间戳矛盾三条规则写成可配置的检查函数；(2) 按配方扩造假样本至 ≥100 条（AI 按配额造假，开始记账）；(3) `TfidfVectorizer` 提特征，`LogisticRegression` 训练，留出验证集；(4) 规则法与基线各算一份准确率/查准/查全/F1，填进同一张表；(5) 分析：哪类造假规则抓得住、哪类抓不住（纯文字造假大概率漏网——如实记下这片盲区）。

**验收证据**：规则检查代码、≥100 条样本、两版基线的指标表。

**常见坑**：规则设得太严把真样本全判假，查准率崩盘——规则是闸门不是筛子；类别失衡却只看总体准确率；sklearn 版本差异导致指标对不上——README 里锁版本。

### 15.7.3　实验三（11 月 23 日）：主力会战——500 条数据集、两个模型、一份自查

**目标**：完成本组主力实验：构建 ≥500条真假混装数据集，训练 TF-IDF+ 逻辑回归与 embedding 小网络两版分类器，画 loss 与 F1 曲线，并给出带数字的泄漏自查——这是“深度网络的模型训练实

践”在本组的落点。

**步骤**：(1)按配方补齐数据集：假 250（易 75/中 100/难 75）+ 真 250，**分层随机洗牌**，`MIX_` 编号，双金标逐条填好；(2) 训 TF-IDF+ 逻辑回归基线；(3) 训 embedding 小网络：把文本变向量（可用预训练词向量平均或轻量 embedding 层）接两层小网络分类，记录每个 epoch 的 loss；(4) 验证集上报**宏观 F1**——每一类各算 F1 再取平均，少数类的成绩不被多数类淹没；(5) 泄漏自查：算考卷-训练重叠率（近似重复一条、事实级重叠一条——同设备、同数值、同时间窗算同一事实变体），**给出数字**；(6) 对照 0.88 → 0.74 写一段不少于 100 字的心得。

**验收证据**：数据集 v1（CSV）、两版训练日志与 loss/F1 曲线、重叠率数字、心得段。

**常见坑**：**F1 只算二分类正类没算宏观**——易类把总分拉高、难类的糟糕被平均掩盖；泄漏自查流于形式——只写“已检查无泄漏”不给数字，按未做处理；洗牌前忘了按难度分层；同一事实的变体一边进训练一边进考卷——这正是审计抓到的头号水分，别自己再犯一遍。

### 15.7.4　实验四（12 月 21 日）：在线判官复核，蒸馏回流再上一分

**目标**：接入在线大模型 API 当“判官复核”，对比纯本地/纯在线/混合三种判定的宏观 F1 与成本；把判对的难样本以 3 倍权重回流再训一轮，看 F1 能不能诚实地涨。

**步骤**：(1) 先封存考卷：从数据集抽 50 条锁进 `exam/` 目录，**先封存、后训练**，时间戳为证；(2) 三种判定各跑一遍考卷：纯本地（自家分类器）、纯在线（API 逐条判）、混合（本地判，拿不准的交 API 复核，每轮不超过 25 条——“不确定必问”的教学版）；(3) 各算宏观 F1 与花费，填三方案对比表；(4) 把本地判对的难样本以 3 倍权重回流，基线再训一轮，与回流前对比；(5) 写不少于 100 字结论：在线值不值、回流涨没涨、水分检查怎么做的。

**验收证据**：三方案 F1 与成本表、回流前后对比、结论段、API 账单。

**常见坑**：**回流样本混进考卷**——先封存再训练，顺序不能反；**把 API 复核结论写回流蒸馏集**——铁律第 (2) 条，泄题比不训还糟；成本超支不记账，台账缺一页本栏扣分；涨了零点几个百分点就宣布胜利——报数字时带上波动范围，诚实到最后一厘米。

# 附录　参考资料与图片来源

本讲义在编写过程中参考了大量教材、论文、开源项目与公开课程资料；正文插图分为三类：自绘示意图（本书原创，matplotlib绘制，脚本随源码归档）、真实项目素材（三个项目的工程数据与系统截图，仅教学内部使用）、网络公开图片（逐图标注来源）。以下分四节列出。

## 一、教材与主要参考书

- 许庆阳.《深度学习模型与算法基础》. 清华大学出版社, 2023 年 12 月第 1 版.（本课程教材）
- Ian Goodfellow, Yoshua Bengio, Aaron Courville.《深度学习》. 人民邮电出版社, 2017 年 7 月第 1 版.（参考书）
- Christopher M. Bishop. Pattern Recognition and Machine Learning. Springer, 2006.（贝叶斯视角经典）
- Richard S. Sutton, Andrew G. Barto. Reinforcement Learning: An Introduction. 2nd ed. MIT Press, 2018.（扩展阅读）
- Stanford CS231n: Deep Learning for Computer Vision 课程讲义. cs231n.stanford.edu.（卷积网络部分参考）
- Christopher Olah. Understanding LSTM Networks 博客文章. colah.github.io, 2015.（第 9-10 讲插图来源）
- Jay Alammar. The Illustrated Transformer / The Illustrated Stable Diffusion. jalammar.github.io, 2018/2022.（第 11-12 讲插图来源）

## 二、主要论文与开源项目（按讲次）

- 第 1-2 讲：F. Rosenblatt. The Perceptron: A Probabilistic Model for Information Storage and Organization in the Brain. Psychological Review, 1958; D. Rumelhart, G. Hinton, R. Williams. Learning representations by back-propagating errors. Nature, 1986.
- 第 4 讲：D. Erhan et al. Why Does Unsupervised Pre-training Help Deep Learning? JMLR, 2010; K. He, X. Zhang, S. Ren, J. Sun. Deep Residual Learning（MAE 思想同源）.
- 第 5-8 讲：A. Krizhevsky, I. Sutskever, G. Hinton. ImageNet Classification with Deep CNNs. NeurIPS 2012（AlexNet）; K. Simonyan, A. Zisserman. Very Deep Convolutional Networks. ICLR 2015（VGG）; C. Szegedy et al. Going Deeper with Convolutions. CVPR 2015（GoogLeNet）; K. He et al. Deep Residual Learning for Image Recognition. CVPR 2016（ResNet）; M. Zeiler, R. Fergus. Visualizing and Understanding Convolutional Networks. ECCV 2014; A. G. Howard et al. MobileNets. 2017; J. Long, E. Shelhamer, T. Darrell. Fully Convolutional Networks. CVPR 2015; L.-C. Chen et al. Encoder-Decoder with Atrous

Separable Convolution (DeepLab v3+). ECCV 2018; A. Kirillov et al. Segment Anything. ICCV 2023.

- 第 9-10 讲：J. Hopfield. Neural networks and physical systems with emergent collective computational abilities. PNAS, 1982; M. I. Jordan. Serial Order: A Parallel Distributed Processing Approach. 1986; S. Hochreiter, J. Schmidhuber. Long Short-Term Memory. Neural Computation, 1997; K. Cho et al. Learning Phrase Representations using RNN Encoder-Decoder (GRU). EMNLP 2014; A. Gu, T. Dao. Mamba: Linear-Time Sequence Modeling with Selective State Spaces. 2023.
- 第 11 讲：A. Vaswani et al. Attention Is All You Need. NeurIPS 2017; J. Devlin et al. BERT. NAACL 2019; A. Radford et al. Language Models are Few-Shot Learners (GPT-3). NeurIPS 2020; A. Radford et al. Learning Transferable Visual Models From Natural Language Supervision (CLIP). 2021; OpenAI/o1 与 DeepSeek-R1 推理模型技术报告, 2024–2026.
- 第 12 讲：I. Goodfellow et al. Generative Adversarial Nets. NeurIPS 2014; A. Radford et al. Unsupervised Representation Learning with DCGAN. ICLR 2016; T. Karras et al. A Style-Based Generator Architecture for GANs (StyleGAN). CVPR 2019; R. Rombach et al. High-Resolution Image Synthesis with Latent Diffusion Models. CVPR 2022.
- 项目一素材：360Stitcher 项目任务书与项目理解报告（教学内部资料, 2026）; E. Rublee et al. ORB: An Efficient Alternative to SIFT or SURF. ICCV 2011; P. E. Sarlin et al. SuperGlue. CVPR 2020; P. Lindenberger et al. LoFTR. ICCV 2021.
- 项目二素材：中海油水下双目项目任务书与自动打标签调研（教学内部资料, 2026）; NVIDIA FoundationStereo (CVPR 2025) 与 Fast-FoundationStereo (CVPR 2026) 项目页; L. Yang et al. Depth Anything V2. NeurIPS 2024; UWStereo/FLSea/SQUID 公开水下数据集; Video Depth Anything (CVPR 2025); Stereo Anywhere (CVPR 2025); MonSter (CVPR 2025); DEFOM-Stereo (CVPR 2025).
- 项目三素材：工程资料结构化与本地数据查询系统总体方案（教学内部资料, 2026）; DeepSeek 对抗训练 demo 技术文档（教学内部资料, 2026）; Y. Zhang et al. DragonN-graph / R-GCN: M. Schlichtkrull et al. Modeling Relational Data with Graph Convolutional Networks. ESWC 2018; Microsoft GraphRAG: From Local to Global. 2024; Spider/BIRD Text2SQL 评测基准.

## 三、网络图片来源（逐图）

以下为正文 `webfig` 类插图的下载来源，全部为公开网页资料，仅用于课堂教学。

| 图 key | 来源域名 | 说明 |
|---|---|---|
| web_alexnet | cs231n.stanford.edu | AlexNet 原始架构图 (Krizhevsky 2012, 双 GPU 结构) |
| web_anchor_chain | commons.wikimedia.org | 粗重锚链链环特写 (系泊特征) |
| web_attention_qkv | jalammar.github.io | 多头注意力 Q/K/V 投影示意 (Jay Alammar) |
| web_camera_rig | commons.wikimedia.org | 谷歌街景多相机阵列采集设备 |

| 图 key | 来源域名 | 说明 |
|---|---|---|
| web_cline_vscode | github.com | VS Code 中 Cline AI 编程助手编辑代码界面 |
| web_clip_demo | arxiv.org | CLIP 对比预训练与零样本预测 (Radford et al. 2021) |
| web_cnn_typical | arxiv.org | CNN 层级特征可视化:Layer1 边缘→Layer5 物体 (Zeiler-Fergus 反卷积重建) |
| web_data_growth | commons.wikimedia.org | AI 训练数据点指数增长曲线 (Our World in Data) |
| web_deepfake_example | commons.wikimedia.org | 著名 AI 深度伪造图: 教皇白色羽绒服假照 (2023) |
| web_deeplab_seg | arxiv.org | DeepLab 语义分割结果 (输入/CRF 前后对比) |
| web_diffusion_stable | jalammar.github.io | Stable Diffusion 三大组件 (CLIP-Text+UNet+ 图像解码器) |
| web_dl_history | arxiv.org | ILSVRC 历年冠军错误率里程碑 (AlexNet → ResNet vs 人类 5%) |
| web_fcn_paper | arxiv.org | FCN 端到端像素预测示意 (Long et al. 2015 原图) |
| web_fpso_ship | commons.wikimedia.org | FPSO”Mystras” 船形浮式生产储卧油装置照片 (甲板 + 火炬) |
| web_gan_paper | arxiv.org | GAN 训练演化四联图 (Goodfellow et al. 2014 原图) |
| web_gan_progress | github.com | StyleGAN 生成 1024px 逼真人脸 (NVIDIA 官方样张) |
| web_googlenet | arxiv.org | GoogLeNet Inception 模块 (带 1x1 降维) |
| web_gpt_family | arxiv.org | GPT-3 零样本/单样本/少样本 vs 微调 (Brown et al. 2020) |
| web_iris_flower | commons.wikimedia.org | 变色鸢尾 (Iris versicolor) 花朵, 鸢尾花数据集类别之一 |
| web_knowledge_graph | commons.wikimedia.org | 知识图谱节点-边可视化示例 (实体关系) |
| web_labeling_tool | github.com | CVAT 标注工具界面 (图像 + 标签面板) |
| web_llm_agent_loop | lilianweng.github.io | ReAct 思考-行动-观察循环轨迹 (Lilian Weng 博客) |
| web_lstm_colah | colah.github.io | LSTM 门结构链式图 (Colah 经典图) |
| web_machine_vision_camera | en.wikipedia.org | 海康威视工业相机产品照 |

| 图 key | 来源域名 | 说明 |
|---|---|---|
| web_mnist_samples | commons.wikimedia.org | MNIST 手写数字 0-9 样本网格 |
| web_mobilenet | arxiv.org | 标准卷积 vs 深度可分离卷积分解 (MobileNet 论文原图) |
| web_neuron_bio | commons.wikimedia.org | 生物神经元结构示意 (树突/胞体/轴突/髓鞘) |
| web_offshore_platform | commons.wikimedia.org | 海上油气生产平台黄昏照 (燃烧火炬) |
| web_ollama_local | github.com | Ollama 官方标志 (本地大模型运行框架) |
| web_panorama_example | commons.wikimedia.org | 360° 室内全景拼接成果 (教堂内部) |
| web_resnet_paper | arxiv.org | ResNet 残差学习块原图 (He et al. 2015) |
| web_rnn_colah | colah.github.io | RNN 按时间展开图 (Colah 经典图) |
| web_rov_underwater | commons.wikimedia.org | Oceaneering 工作级 ROV 水下结构物作业 (框架/推进器/缆) |
| web_sam_demo | arxiv.org | SAM 可提示分割: 任务/模型/数据引擎 (Kirillov et al. 2023) |
| web_stereo_camera | commons.wikimedia.org | Kodak 双镜头立体相机硬件 |
| web_transformer_arch | commons.wikimedia.org | Transformer 整体结构 (Vaswani et al. 2017 Fig.1) |
| web_turing | commons.wikimedia.org | 图灵 16 岁经典肖像照片 |
| web_underwater_turbid | commons.wikimedia.org | 浑浊水体中的塞氏盘透明度测量照片 |
| web_vgg16 | commons.wikimedia.org | VGG16 结构块图 (13 卷积 +3 全连接) |

## 四、真实项目素材与脚本

正文 `projfig` 类插图来自三个项目的真实工程资料，包括：六目拼接项目现场照片与系统遥测截图（`E:\六目拼接`）；HYSY123 FPSO锚链 ROV 检查视频帧、自建 500 对校正立体图对与分析可视化（`E:\锚链视频`）；数据基座治理系统架构图与对抗训练 demo 界面截图（`E:\中海油数据`）。上述素材仅限本课程教学使用，不得对外传播。全部自绘图的 matplotlib 生成脚本与来源清单随讲义源码归档，便于教师复现与修改。