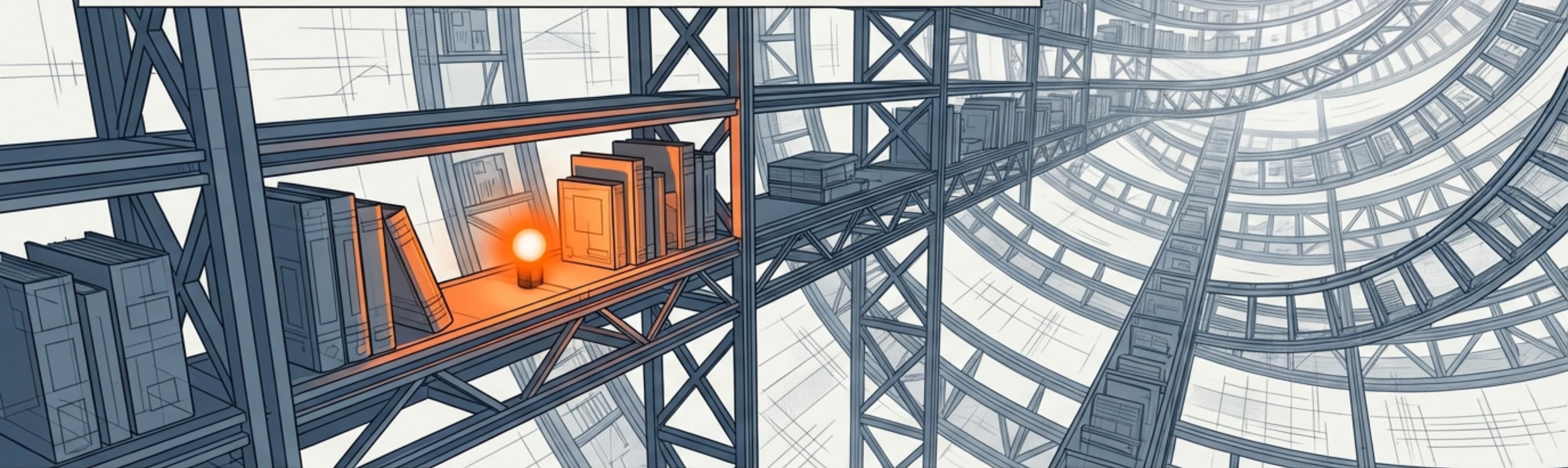


突破极限：递归语言模型 (RLM)

既然记不住，不如学会“查阅”——大模型推理算力扩张的新范式



Dr. Lin

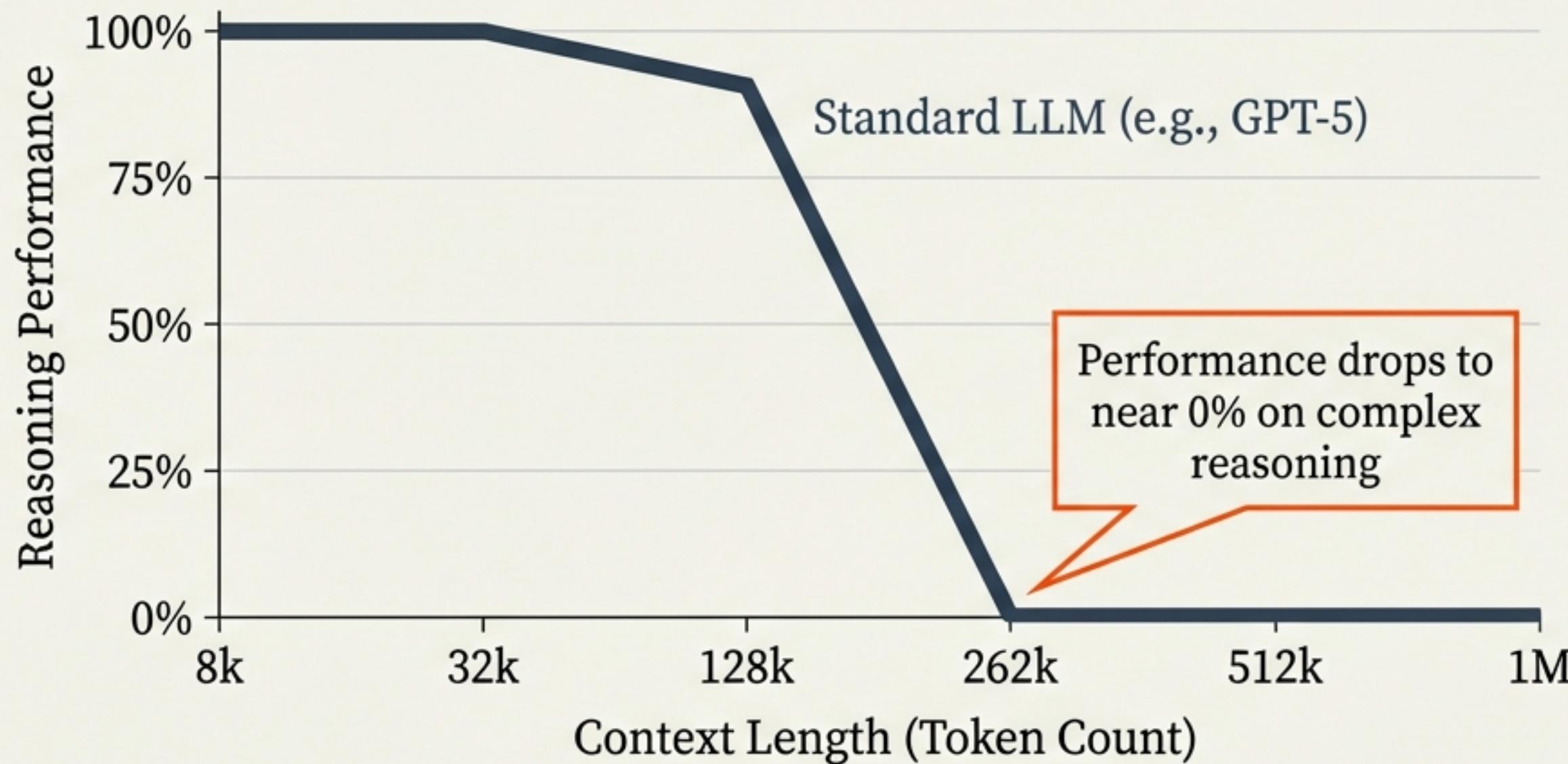
我们一直以为更大的‘上下文窗口’等于更好的记忆。但如果这只是一个错觉呢？



Dr. Chen

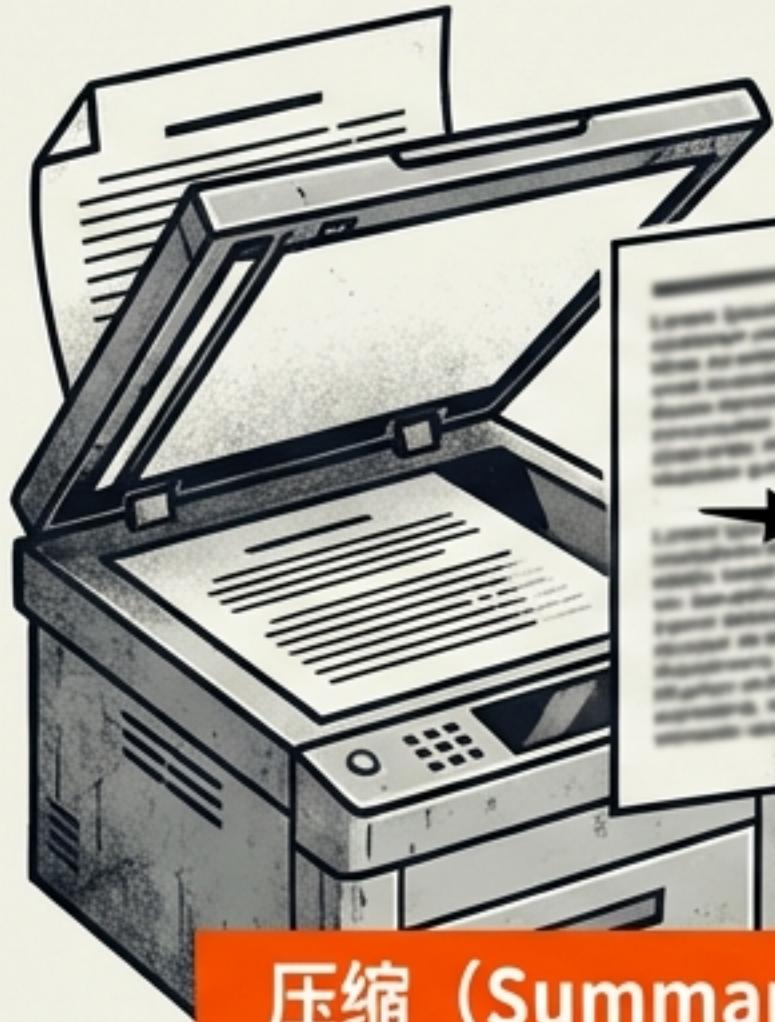
今天，我们不靠训练，而是通过代码打破 1000 万 token 的屏障。

仅仅做大窗口是不够的：上下文腐烂 (Context Rot)



- **现状 (Status Quo):** 模型厂商竞相推出 1M+ token 窗口，试图让模型像学生考前突击一样“背诵”整本书。
- **瓶颈 (The Wall):** 随着输入长度增加，模型在复杂任务（如 Oolong）上的表现迅速下降至 0%。
- **Dr. Lin:** "这就好比让一个人强行背下整本百科全书。即使你能‘读’进去，你也无法有效地‘思考’它。"

现有的创口贴方案：有损压缩与局部视野



压缩 (Summarization) - 细节丢失

压缩 (Compaction/Summarization) : 像“传声筒”游戏，信息在反复摘要中丢失细节。

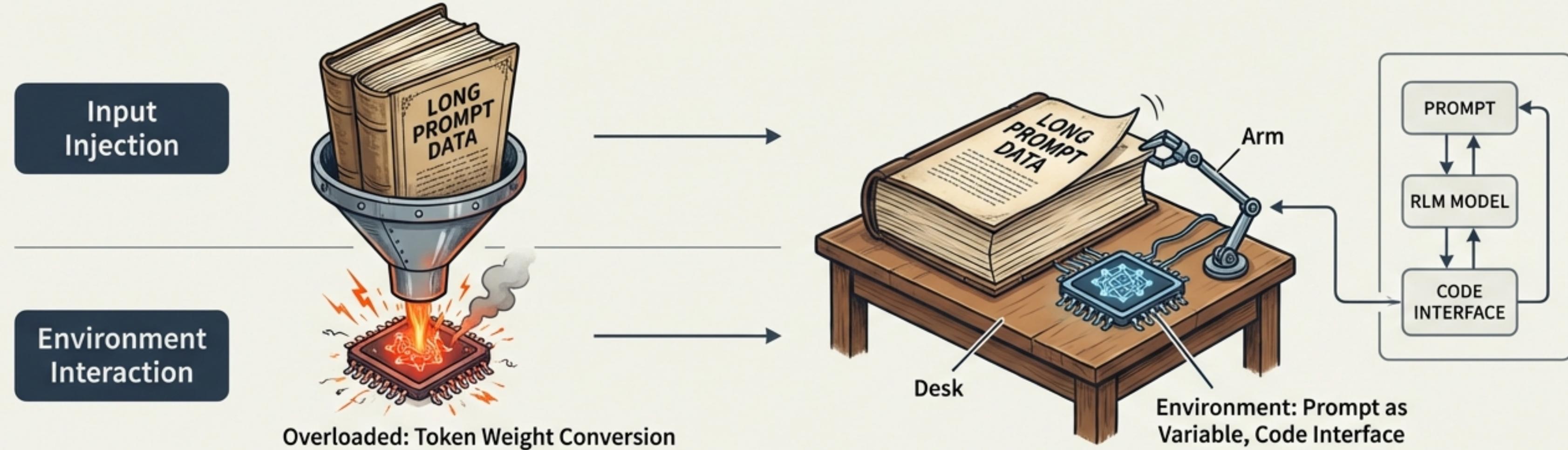


检索增强 (RAG)
- 缺乏全局视野

检索增强 (RAG) : 缺乏全局视野，难以处理需要跨越多个文档的推理 (Global Reasoning)。

Dr. Chen: “摘要丢失了细节，RAG 丢失了连线。我们需要一种既保留细节又能全局思考的方法。”

范式转移：不要把 Prompt 喂给神经网络

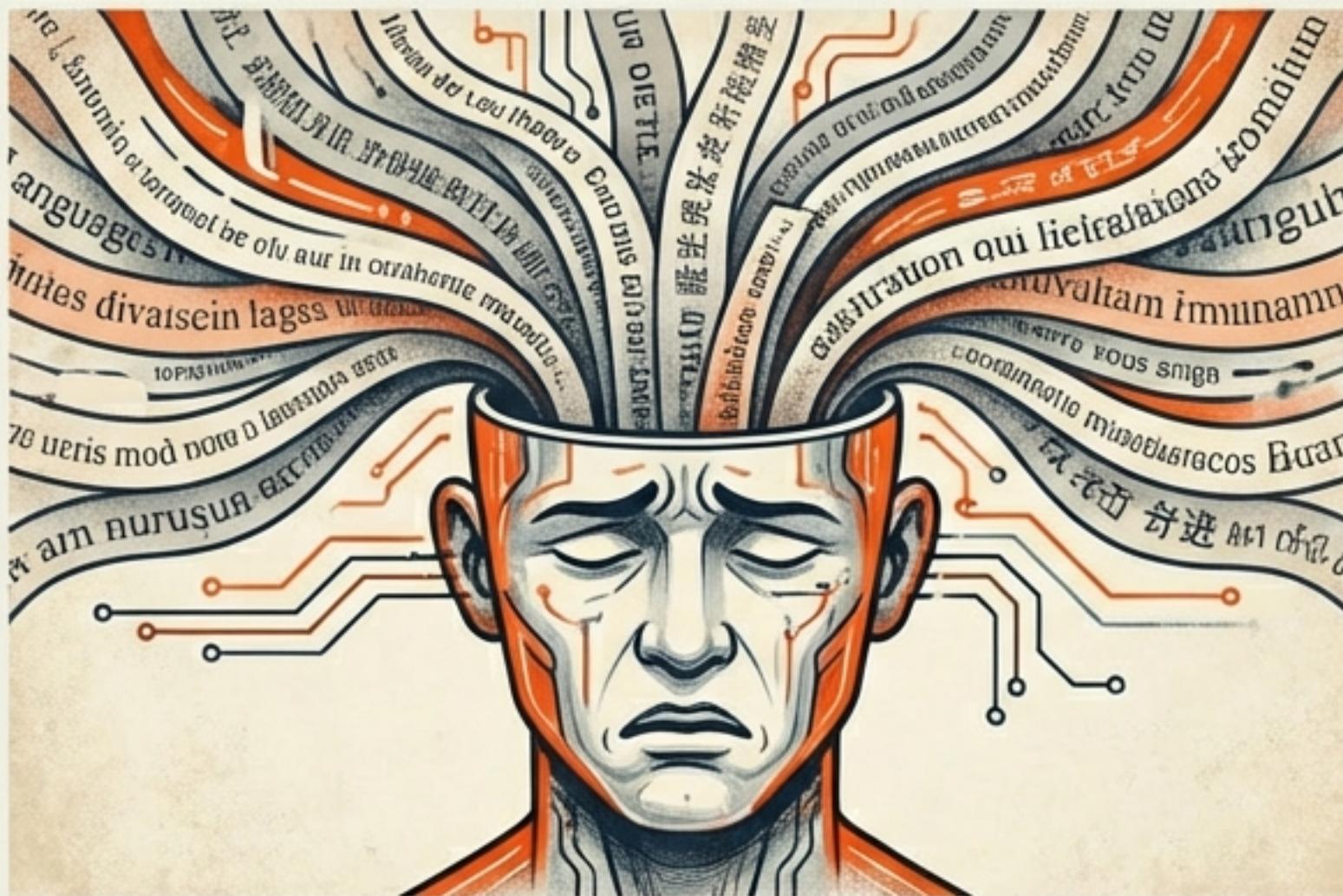


核心洞察 (Core Insight)：长文本不应作为输入 (Input)，而应作为环境 (Environment)。

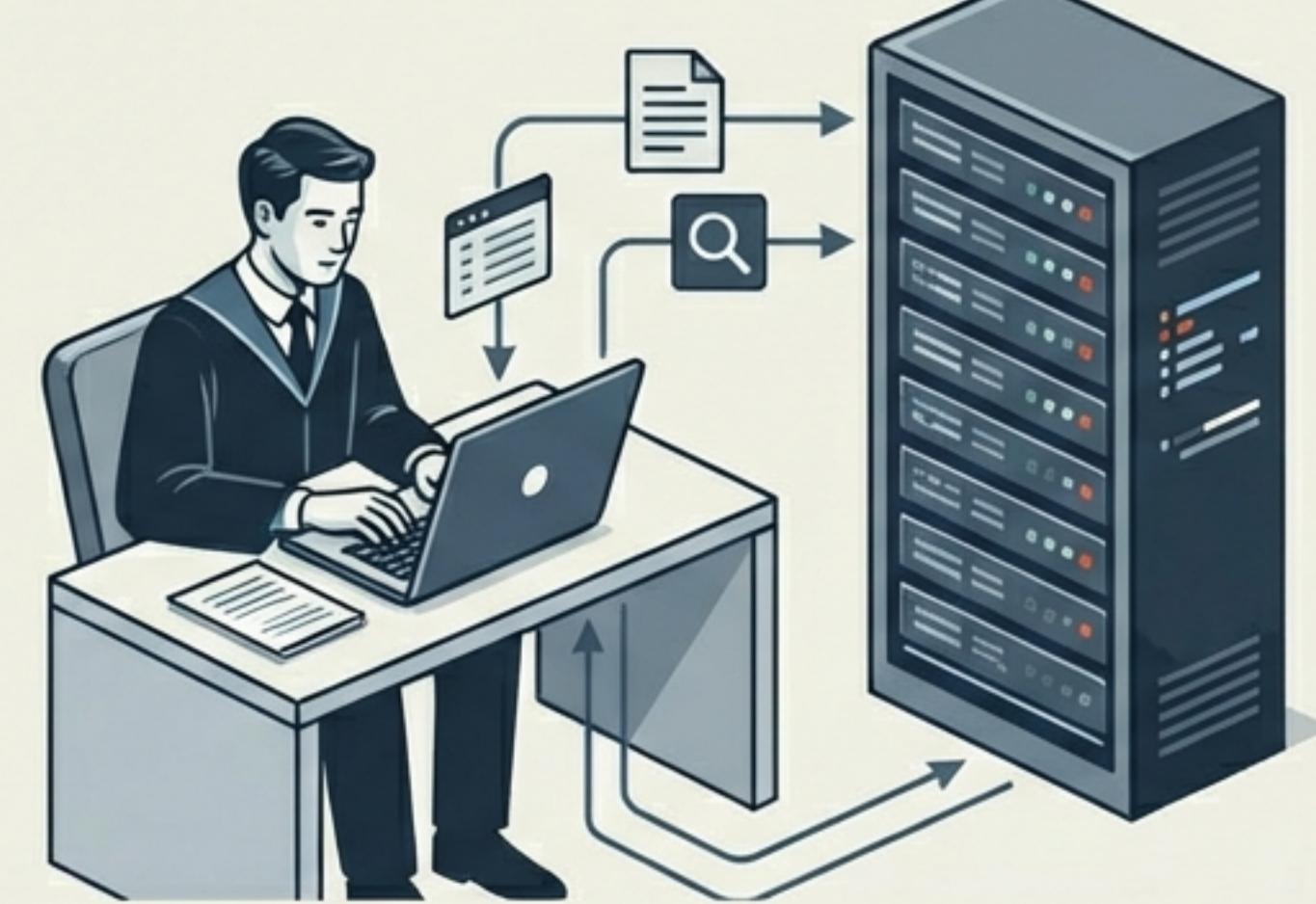
RLM 定义：递归语言模型 (Recursive Language Models) 将 Prompt 存储为外部变量，通过代码与之交互，而非将其转化为 Token 权重。

Dr. Lin: “这是认知卸载 (Cognitive Offloading) 的极致体现——把记忆的负担交给环境，大脑只负责处理逻辑。”

从“背诵者”到“研究员”



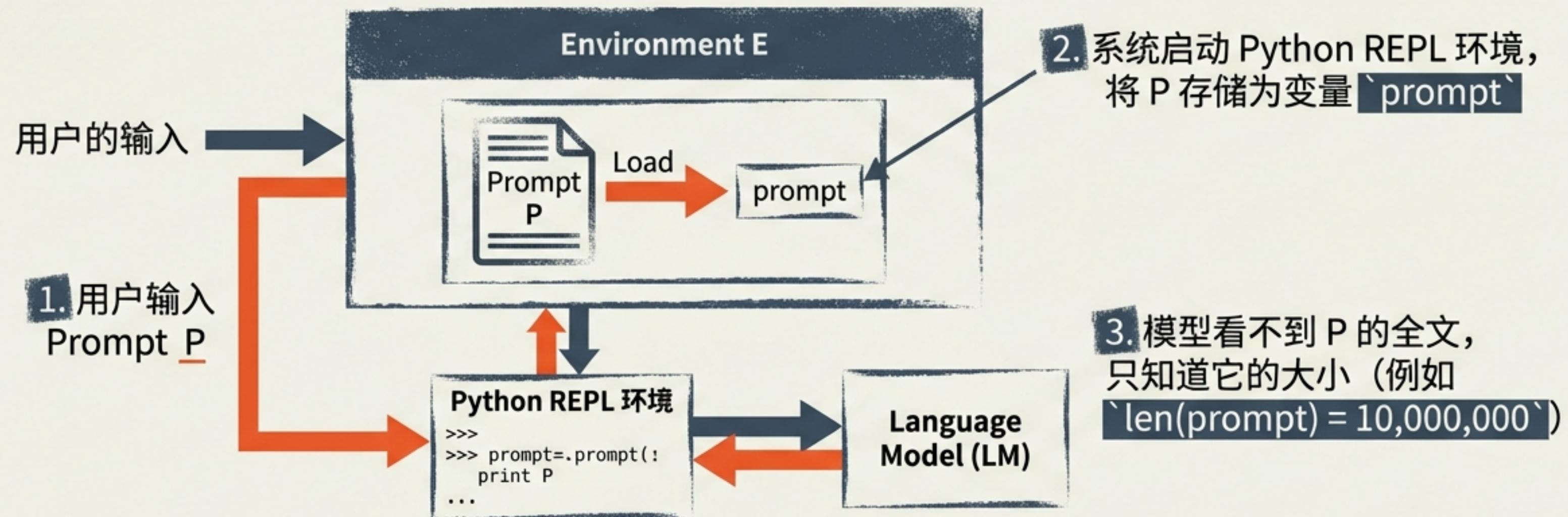
Standard LLM: 试图一次性将所有信息加载到显存中。



RLM: 这是一个 REPL (Read-Eval-Print Loop) 循环。模型编写代码来“查阅”数据，就像人类使用 Ctrl+F 或阅读特定章节。

关键区别：这是一个从 **记忆 (Memory)** 到 **寻址 (Navigation)** 的转变。

架构解析：环境即变量



“Dr. Chen: ‘模型第一眼看到的不是莎士比亚全集，而是文件的大小。它必须写代码去‘偷看’里面的内容。’

机制一：符号化交互 (Symbolic Interaction)

操作：模型编写 Python 代码 (Regex, Slice) 来采样数据。

优势：极低的算力消耗。读取 10M token 的元数据几乎不花钱。

“

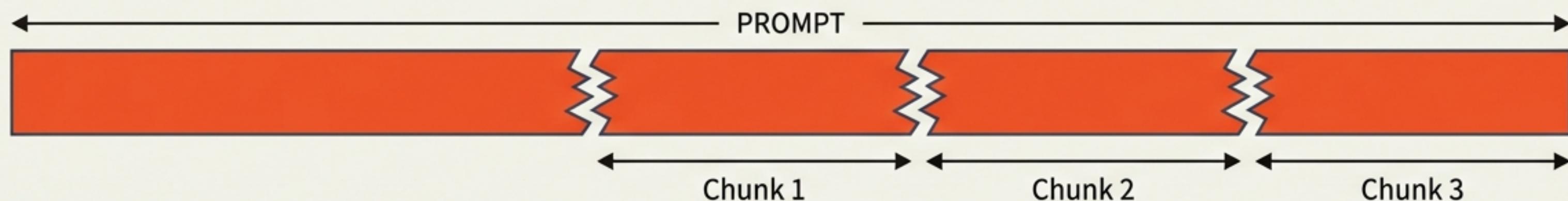
看，模型首先做了
`print(context[:500])`。它像人类一样，
先读‘前言’来搞清楚这是一篇什么文章。

— Dr. Chen

```
# Let's first examine the context to understand its
structure
print(f"Context length: {len(context)} characters")
print("First 500 characters of context:")
print(context[:500])
```

```
Context length: 76803 characters...
```

机制二：分块与任务分解 (Decomposition)



策略：既然读不完，就切分。模型自动编写

循环：`for chunk in chunks:
process(chunk)`。

心理学原理：组块 (Chunking) —— 将巨大的
的认知负荷分解为可管理的单元。

```
# Process in batches of 100 to avoid too many LLM calls
batch_size = 100
for i in range(0, len(lines), batch_size):
    batch = lines[i:i+batch_size]
    process_batch(batch)
```

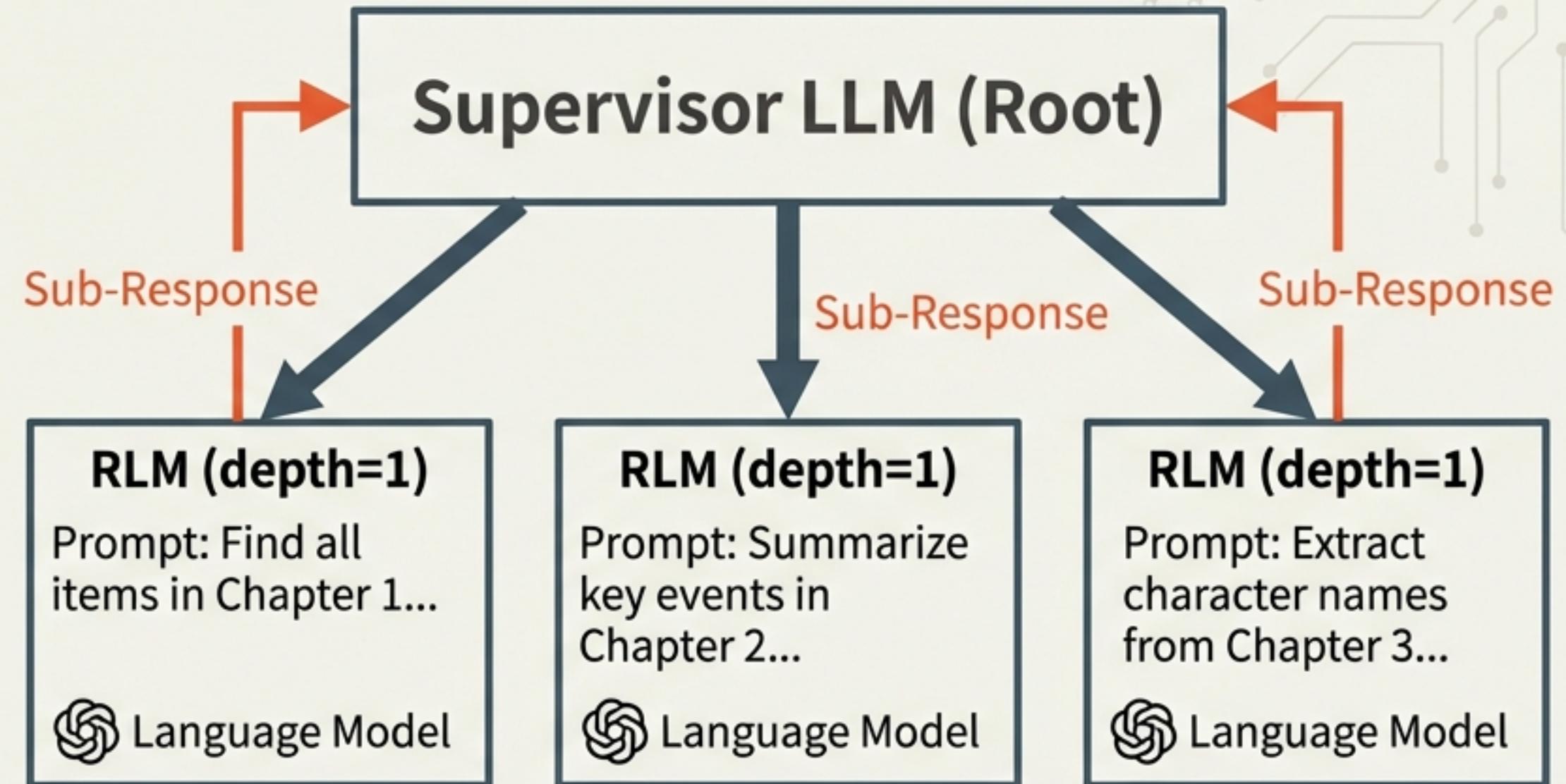
“模型不仅是在执行代码，它是在规划。它意识到‘一口吃不成胖子’，于是
它创造了自己的批处理逻辑。” — Dr. Lin

机制三：递归调用 (Recursive Calls)

Supervisor LLM: 负责规划路径，编写代码。

Worker LLM (Sub-call): 这是一个较小的上下文窗口，专门处理单一的 Chunk。

流程：主模型调用 `llm_query(chunk)` -> 子模型处理 -> 返回结果给主变量。

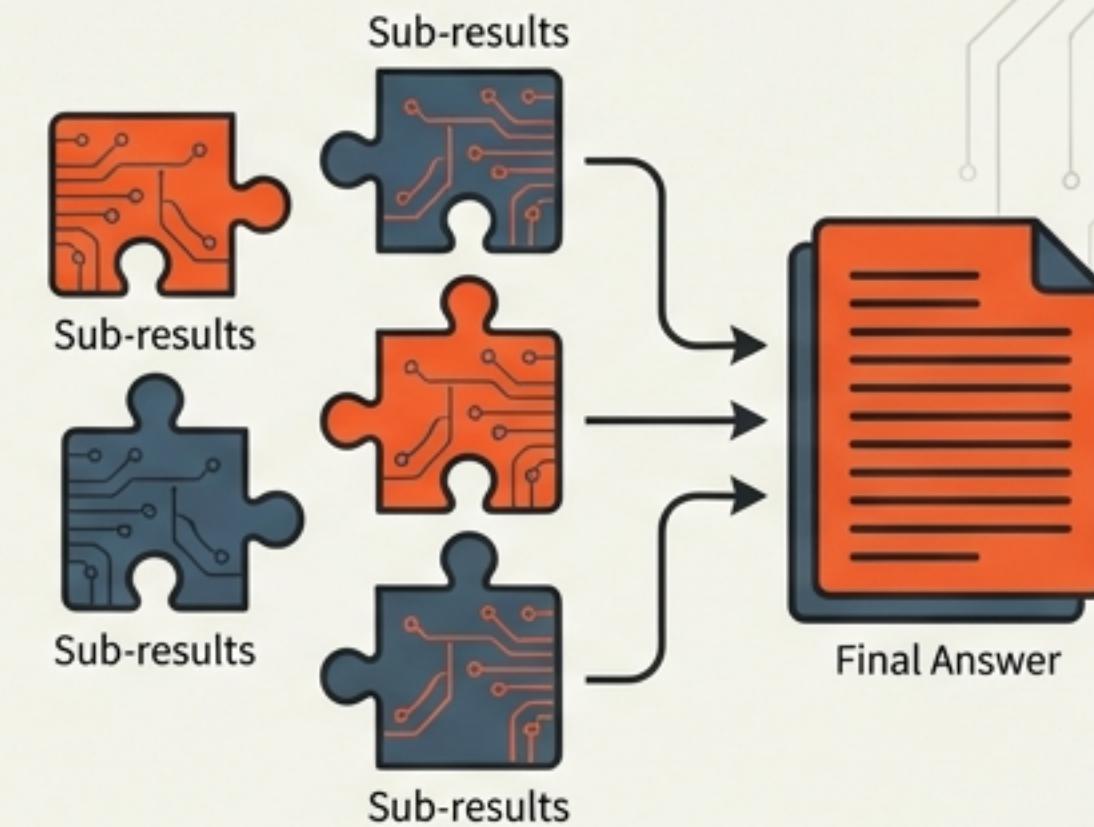


“这是递归深度 depth=1 的演示。主模型变成了‘包工头’，将任务分发给无数个‘工人’。”

— Dr. Chen

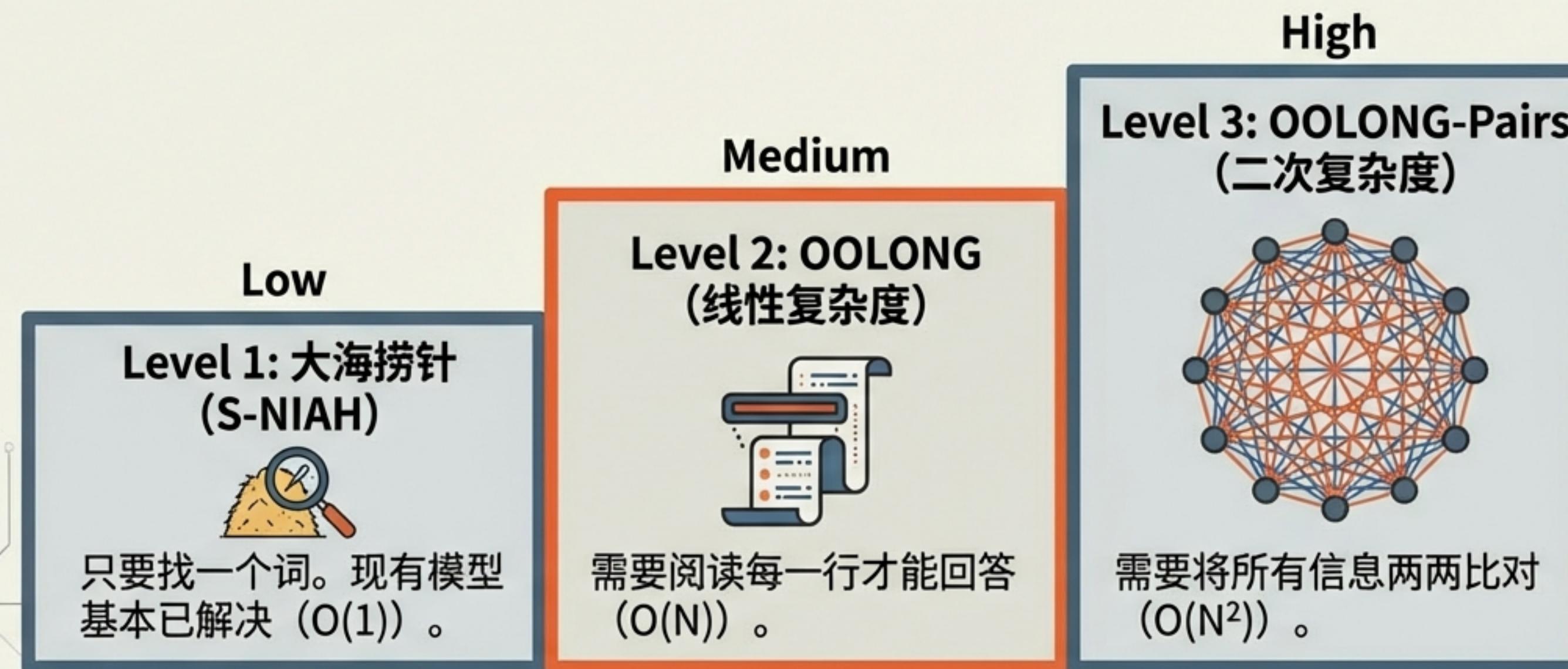
机制四：结果缝合 (Synthesis)

```
# Format the final answer as required
if pairs:
    formatted_pairs = [f"{{pair[0]}}, {pair[1]}}" for pair in pairs]
    final_result = "\n".join(formatted_pairs)
    print("Final result (first 50 lines):")
    FINAL_VAR(final_result)
```



聚合: 子模型的输出被存储在列表或字典中。
输出: 主模型运行 `Final_Answer = aggregate(results)`。
结果: 即使原始文本长达 1000 万 token，最终答案也是经过验证的精确事实，而非模糊的幻觉。

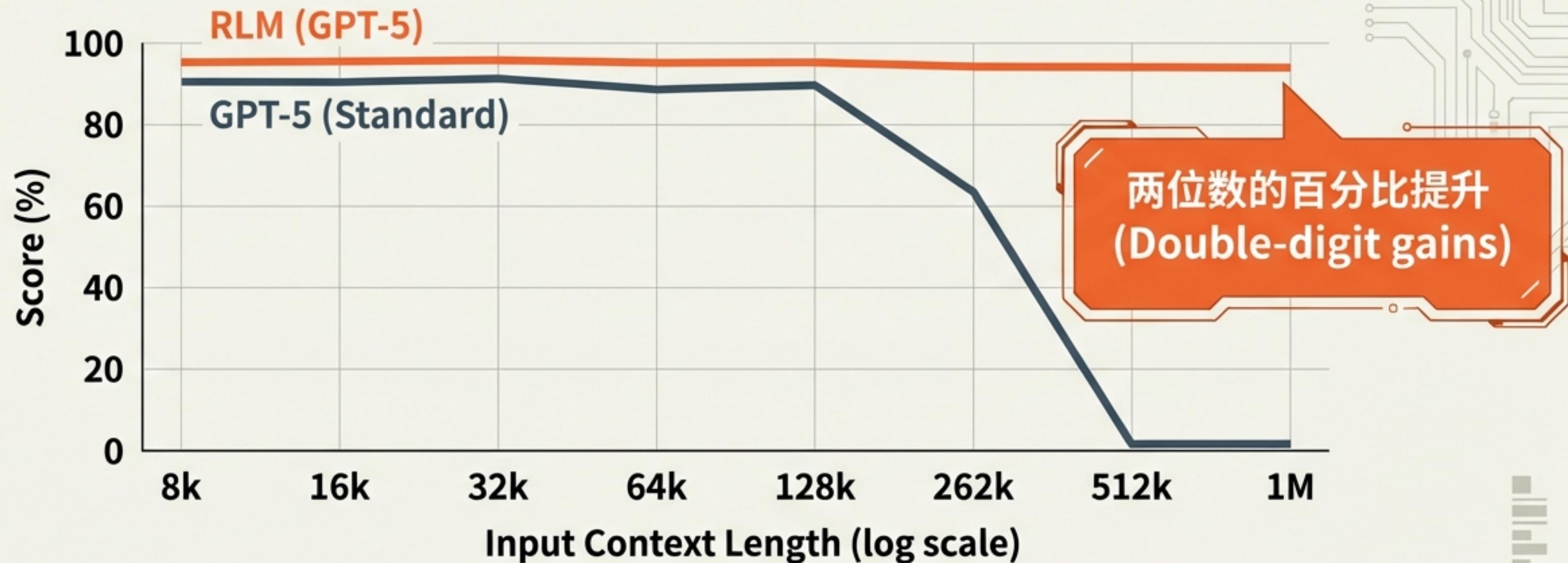
测试场：并不是所有的长文本任务都一样



“大多数模型在 Level 1 表现尚可，但在 Level 3 会彻底崩溃。这才是 RLM 的主场。”

— Dr. Lin

结果：在百万 Token 级别保持“清醒”



GPT-5 Standard (Blue Line): 在 OOLONG 任务中，随着长度增加，准确率迅速跌至 0%。

RLM GPT-5 (Orange Line): 在 100 万 token 甚至更长的情况下，依然保持高准确率。

攻克“二次方复杂度”难题

```
# Now generate all pairs of target users (no duplicates, lower ID first)
target_users_sorted = sorted(list(target_users))
pairs = []

for i in range(len(target_users_sorted)):
    for j in range(i+1, len(target_users_sorted)):
        pairs.append((target_users_sorted[i], target_users_sorted[j]))
print(f"Total pairs: {len(pairs)}")
print("First 20 pairs:")
for i in range(min(20, len(pairs))):
    print(pairs[i])

Total pairs: 8515
```

GPT 5 表现: < 0.1% F1 Score

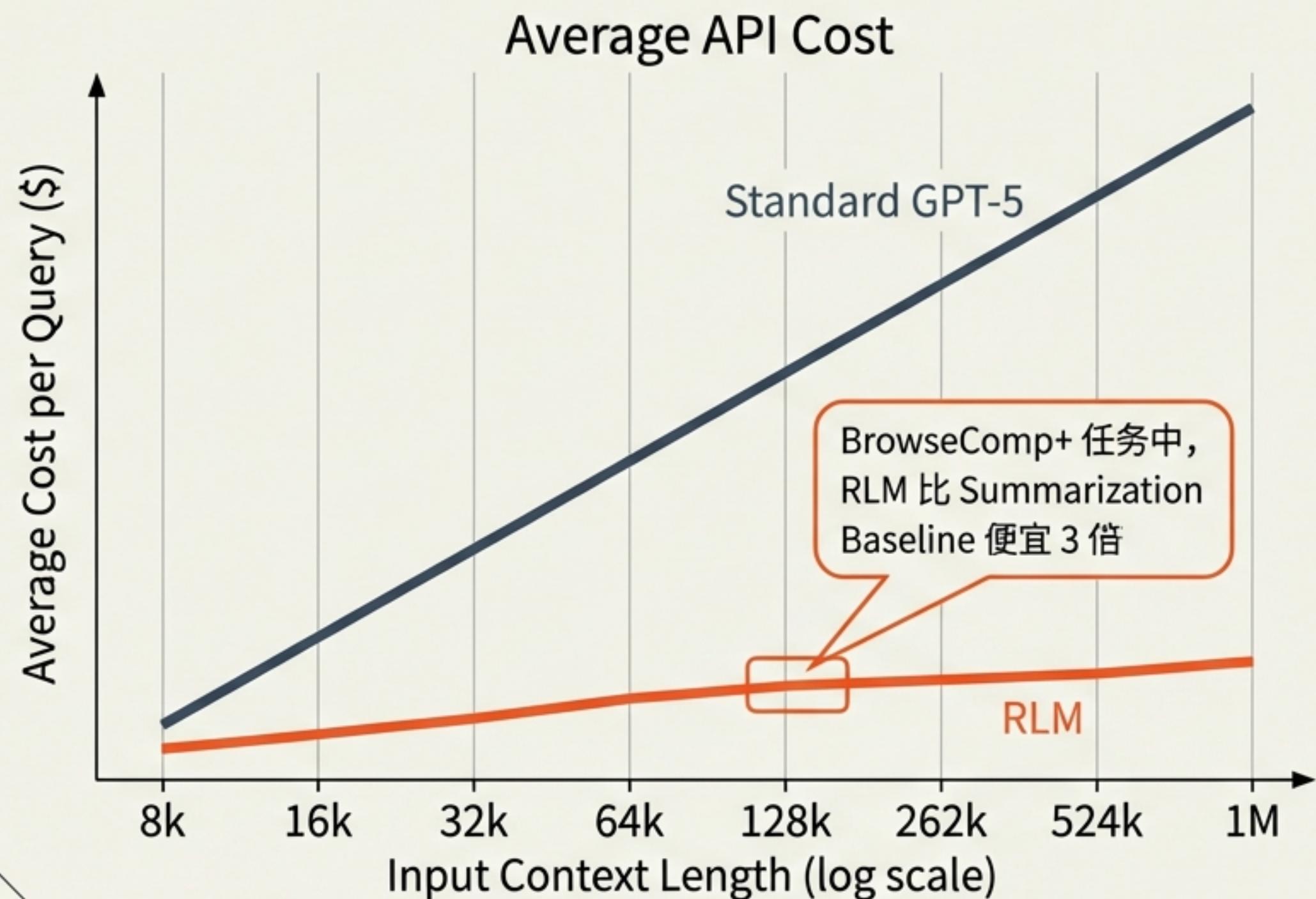
RLM 表现:
~58% F1 Score

任务: 找出列表中所有符合特定条件的用户对 (Pairs)。

原因: RLM 通过代码逻辑强制遍历所有组合, 而不是依赖注意力机制的模糊关联。

Noto Serif SC

成本悖论：更聪明，却更便宜



传统模式：

输入 10M token = 支付
10M token 的费用 (\$\$\$)。

RLM 模式：只读取相关的 1%
内容 = 支付 1% 的费用 (\$)。

“这就好比买书：你不需要买下
整座图书馆，你只需要为你看
的那几页付费。” – Dr. Chen

涌现出的智能行为 (Emergent Behaviors)

自主过滤 (Autonomous Filtering)

```
find_snippets(keyword,  
              window=200)
```

Agent proactively scans context for relevant information, setting window size.

自我修正 (Self-Correction)

```
try:  
    ...  
except Exception as e:  
    print("Error processing...")
```

Agent implements error handling and fallback mechanisms for robustness.

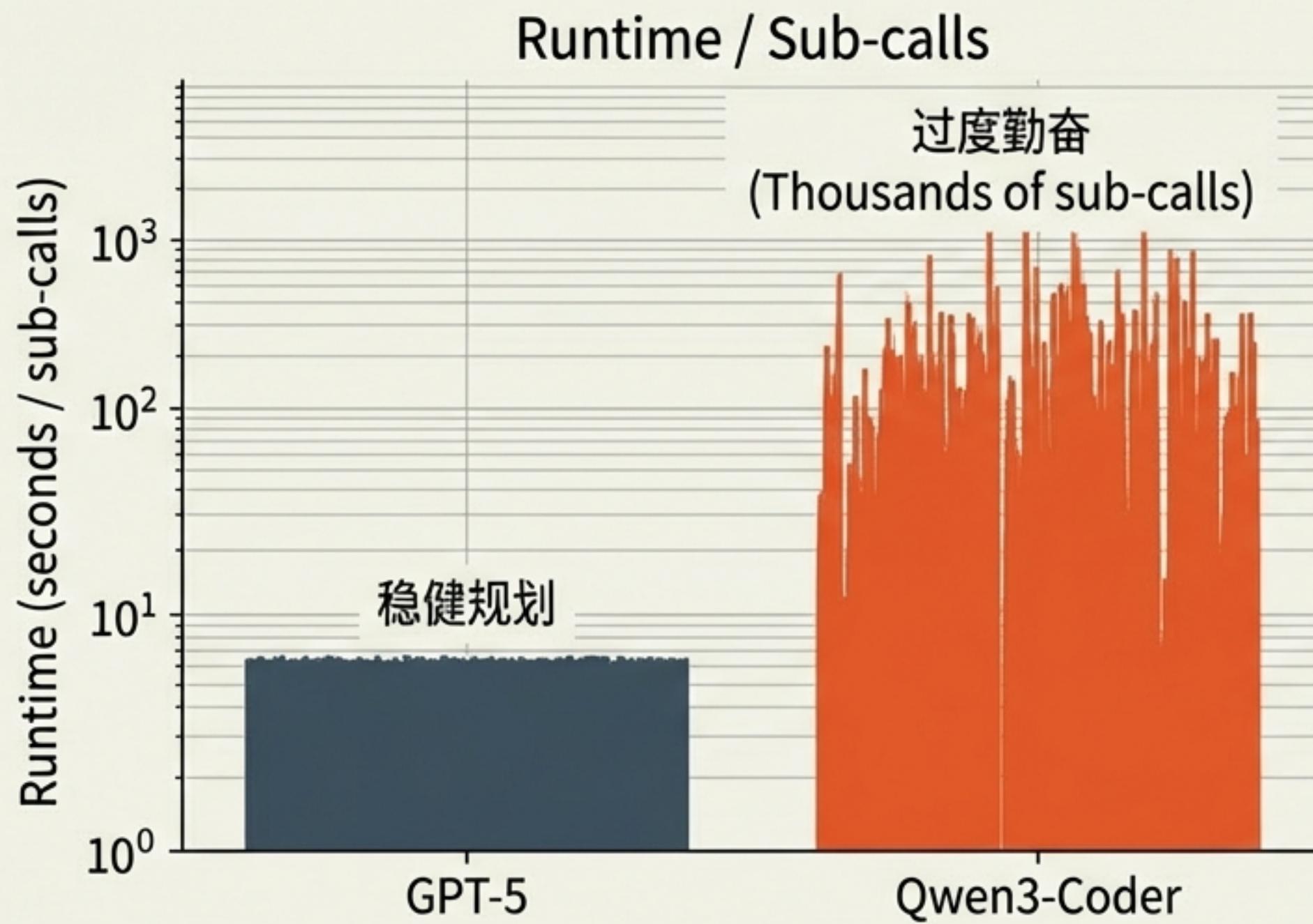
结果验证 (Result Verification)

```
if len(hits) >= max_hits:  
    return hits
```

Agent evaluates output against criteria, terminating when sufficient.

“我们没有教它这些。这是环境强迫它进化出的生存策略。” – Dr. Lin, using Noto Serif SC Regular

并非所有模型都能胜任：编码能力是关键



- **GPT-5:** 表现稳健，懂得“节制”调用子模型。
- **Qwen3-Coder:** 编码能力强，但有时会“过度勤奋”，导致过多的递归调用，增加了成本。
- **结论:** RLM 的性能高度依赖于基础模型的 Coding & Reasoning 能力。

实战案例：超大规模代码库理解

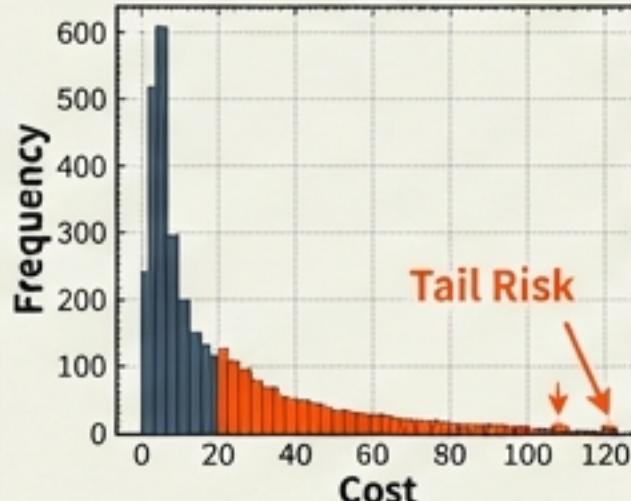


- **挑战：**理解 900k token 的代码库，追踪函数调用。
- **RLM 方法：**像资深程序员一样，先列出文件结构，再 Grep 关键词，最后打开特定文件阅读。
- **结果：**在 LongBench-v2 CodeQA 中显著优于 Baseline。

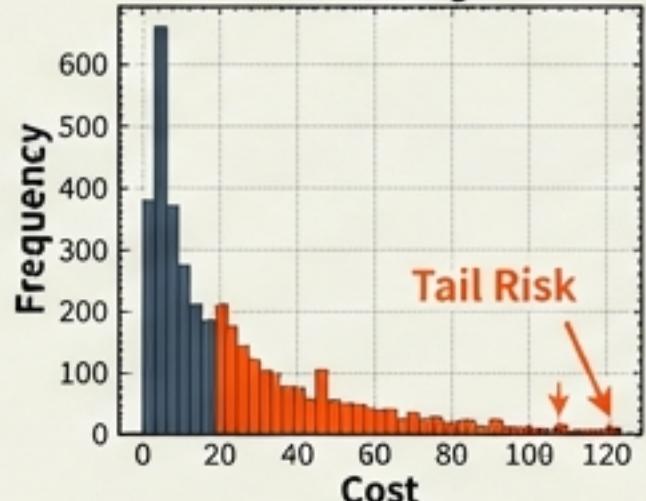
局限性与方差 (Limitations & Variance)

Cost Distributions

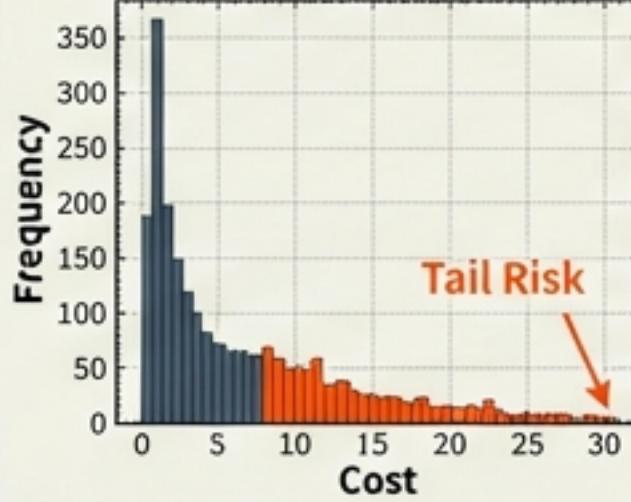
RLM (gpt-5) with REPL for OOLONG



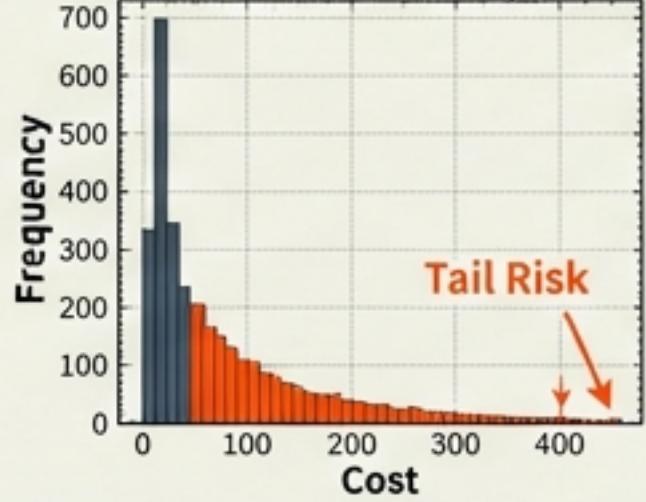
Summary Agent (gpt-5) for CodeQA



Summary Agent (gpt-5) for BrowseComp+



Summary Agent (gpt-5) for BrowseComp+



- **推理时间:** 由于是串行 (Sequential) 处理, RLM 比一次性推理要慢。
- **成本方差:** 虽然平均成本低, 但如果模型陷入“死循环”或过度递归, 成本会飙升 (Tail Risk)。

“这是一个用时间换取空间和精度的策略。我们需要更好的异步并行 (Async) 机制来优化它。” - Dr. Chen

总结：RLM 的四大优势



无限上下文 (Infinite Context)

只要硬盘装得下，模型就能读。



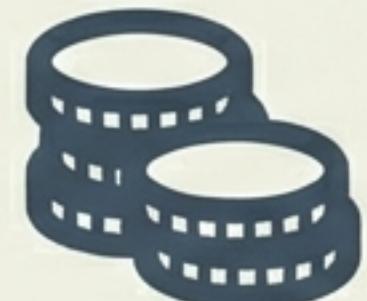
无需重训 (No Retraining)

适用于任何具备强代码能力的现成模型。



白盒推理 (White-Box Reasoning)

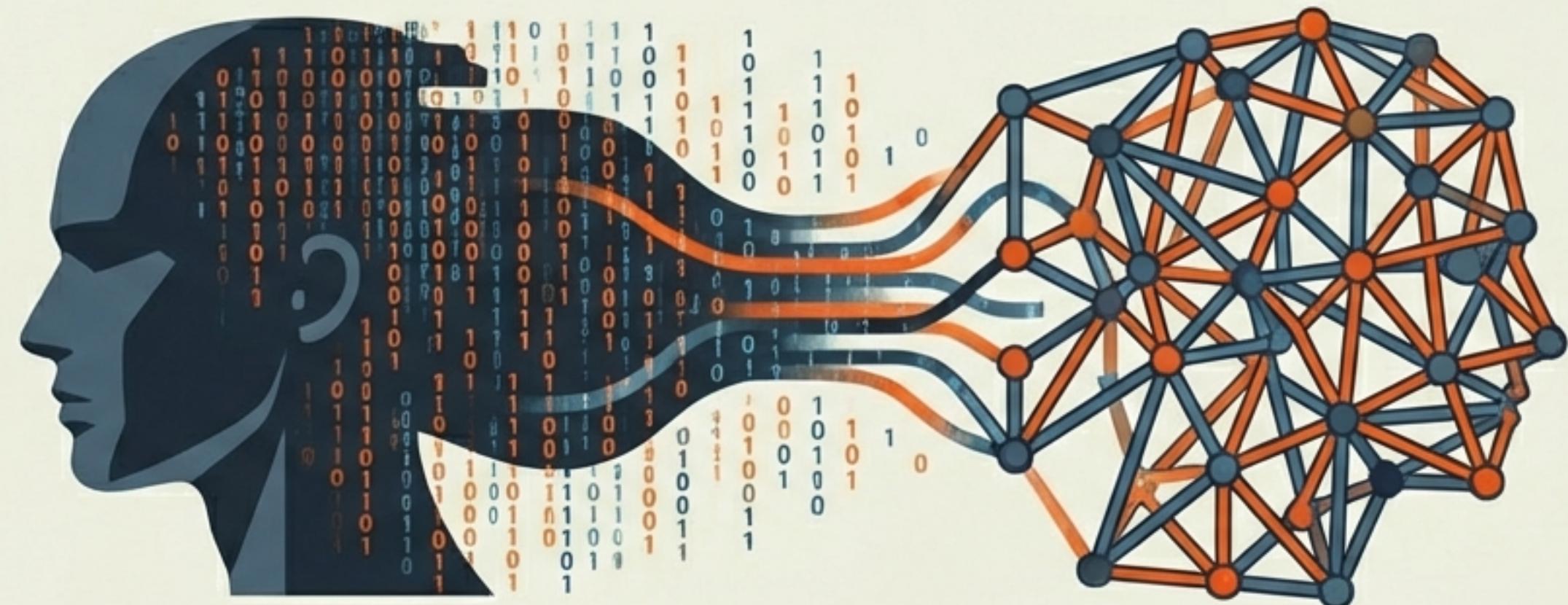
我们可以通过查看代码，完全理解模型的思考过程。



按需付费 (Cost Effective)

避免了长文本的算力浪费。

展望：AI 进化的下一个瓶颈



如果上下文长度不再是限制，那么 AI 能力的下一个天花板是什么？

Dr. Lin: 也许我们正从 ‘**更大的模型**’ (Learning) 走向 ‘**更强的脚手架**’ (Scaffolding)。未来的 AI 不仅需要更大的大脑，更需要更趁手的工具。

拥抱推理时扩展 (Inference-Time Scaling)