今年早些时候，Google 发了一个访谈视频，主持人 Logan 采访 DeepMinnd 首席科学家 Nikolay Savinov。内容是关于长上下文推理的。整个访谈时长为一小时，我把核心内容整理出来了，并附上访谈的完整内容供参考。

【太长不看整理版】

上下文窗口是输入给 LLM 的 “上下文令牌集合”，包含用户提示词（Prompt）、历史交互记录、上传的文件（PDF、视频等）。

在长上下文处理中，LLM 的记忆是重点。

记忆分为**权重内记忆**（in-weight memory），预训练阶段从网络数据中习得，无需外部上下文即可调用，难修改、难更新。**上下文内显式记忆**（explicit in-context memory），用户主动输入的外部信息，易更新、可承载隐私或稀有信息。

短上下文：空间有限，不同知识需 “争夺” 窗口，知识召回率低；长上下文：无需纠结 “输入哪些信息”，可容纳更多内容，提升知识召回率与覆盖率，缓解权重内记忆的 “过时、稀有信息缺失” 问题。

而 RAG 的本质是 “上下文输入前的信息筛选技术”。**RAG 与长上下文工程之间是互补关系而非淘汰**，当前 RAG 还未过时。

处理长上下文的机制与局限性包括，**令牌间存在 “注意力竞争”**。干扰信息会抢占注意力，导致目标信息被忽略；上下文越长，竞争越激烈。受注意力竞争影响，**复杂任务中部分令牌可能被忽略**。

当前从百万级令牌中找单个特定信息，已不再是技术难点。但是当上下文充满与目标**信息相似的内容，模型需精准定位，难度显著提升**；当多信息点检索：从上下文同时提取多个目标信息，需**模型整合多位置信息，是当前核心难点**。

我们需**要充分利用 并构建好“上下文缓存”**，让上下文缓存基于相同前缀可以复用，因此要把问题放在上下文之后（避免每次提问改变缓存前缀，无法复用）。**避免无关信息与知识冲突**，上下文不填充无关信息：既增加推理成本，又干扰多信息点检索效果。**解决知识冲突**：若上下文内记忆与权重内记忆矛盾，用提示词明确优先级引导模型依赖外部上下文。

针对长上下文，用企业知识语料库（数十亿令牌），通过 “语言建模损失” 让模型继续训练，将企业知识融入权重。

对于知识更新频率高：选 RAG（向量数据库成熟，部署效率高）；

对于知识更新频率低：可考虑微调（但需承担高训练成本）。

未来趋势：**3 年左右 1000 万令牌会普及**；长期：1 亿令牌需突破性创新。未来**“长上下文” 将从 “特色功能” 变为 “基础能力”**，用户无需关注窗口规模，默认模型 “能高效处理海量信息”；长上下文与 RAG、智能体、推理能力深度融合，成为 LLM“理解复杂场景、实现自动化” 的核心支柱。

【访谈原始内容翻译】

Logan：今天我们邀请到的嘉宾是谷歌深度思维（Google DeepMind）的资深研究科学家，同时也是长上下文预训练（Long Context Pre-training）项目的联合负责人之一。Nikolay，你好！

Nikolay：你好，感谢邀请。

Logan：我们先从最基础、最有趣的问题开始，然后逐步深入。什么是 “令牌”（token）？大家应该如何理解这个概念？

Nikolay：理解令牌的方式很简单，对于文本而言，它基本上相当于不到一个单词的长度。所以令牌可以是完整的单词、单词的一部分，也可以是标点符号，比如逗号、句号等等。对于图像和音频来说，令牌的定义会略有不同，但针对文本，大家只需将其理解为 “略短于一个单词” 的单位即可。

Logan：好的。那我们为什么需要令牌呢？比如人类通常更熟悉 “字符” 这个概念，为什么人工智能（AI）和大型语言模型（LLMs）会专门有 “令牌” 这样一个特殊概念？它实际上能实现什么功能？

Nikolay：这是个很好的问题，实际上很多研究者也在不断思考这个问题。之前有不少论文尝试摒弃令牌，完全依靠字符级生成来实现模型功能，但这种方式有优势也有明显缺陷。最关键的缺陷在于生成速度会大幅变慢 —— 模型通常一次生成一个令牌，如果你能一次性生成一个完整的单词（以令牌为单位），会比逐个生成字符

快得多。所以那些 “弃用令牌” 的尝试，我认为并没有真正成功，目前我们仍在使用令牌技术。

对于那些没花太多时间研究令牌的人来说，Andrej Karpathy 发布过很多相关视频和推文，他提到 “令牌化（tokenization）是大型语言模型中所有怪异现象和复杂性的根源” —— 比如模型遇到的各种特殊边缘案例，大多是因为模型并非从 “字符级” 理解内容，而是从 “令牌级” 处理信息。一个很典型的例子，现在大家经常讨论的 “数单个单词里的字符数量”，比如 “strawberry（草莓）” 这个单词里有多少个 “r”，模型处理这类问题时总会出错。

Logan：你之前提到令牌会把单词拆分成不同部分，模型并非从单个字符层面看待单词，是不是这个原因导致的？

Nikolay：是的，我认为这很好地描述了问题的核心。有一点需要注意：由于模型的结构设计，它们对世界的 “认知方式” 和人类截然不同。当你看到 “strawberry” 这个词时，你看到的是一串连续的字母；但在模型眼中，这个词可能被拆成了 11 个令牌。这时如果你让模型 “数这个词里有多少个‘r’”，它很难完成这个任务 —— 因为在预训练阶段，模型需要将 “r” 这个字符令牌

（它可能在互联网文本中随处可见）与 “strawberry” 这个单词令牌（可能是一个独立令牌）关联起来，这种关联对模型来说其实并不简单。当然，一旦模型做不到这件事，我们就会抱怨：“连 CGI（计算机生成图像）都能实现的功能，模型怎么连‘数 strawberry 里的 r’ 都做不到？这可是小孩都能完成的事！”

Logan：确实特别奇怪。

Nikolay：还有一个有趣的点：如果你看过 Karpathy 的那些视频，会发现令牌化在 “空格” 处理上也存在很多问题。这一点很关键，因为大多数令牌在生成时会默认前缀一个空格，这可能导致一些怪异的结果 —— 比如在文本边界处，你以为只是添加了一段普通内容，但对模型来说，这种 “带空格的拼接” 是它很少见过的情况，就可能出现异常。

Logan：很有意思，这确实值得深究。这让我想到了 “上下文窗口”（context window）这个话题。现在关于 “长上下文” 的讨论很多，而 “长上下文” 本身就默认大家知道 “上下文窗口” 是什么概念。但你能给大家通俗地解释一下，大家应该如何理解 “上

下文窗口”？无论是作为大型语言模型的用户，还是 AI 模型的开发者，我们为什么需要关注上下文窗口？

Nikolay：上下文窗口，本质上就是我们输入给大型语言模型的“上下文令牌集合”——它可以是当前的提示词（prompt）、与用户之前的交互内容，也可以是用户上传的文件（比如视频或 PDF）。当你向模型提供上下文时，模型的知识来源其实有两个：第一个是我所说的“权重内记忆”（in-weight memory），也就是模型在预训练阶段从“互联网数据切片”中学习到的知识。对于这类知识，模型不需要额外的上下文就能记住相关事实，所以即便没有外部上下文，模型本身也已经具备一定的记忆能力。第二个则是“上下文内显式记忆”（explicit in-context memory），也就是你主动提供给模型的外部信息。

理解这两种记忆的区别非常重要，因为“上下文内记忆”比“权重内记忆”更容易修改和更新。对于某些知识，“权重内记忆”可能已经足够——比如一些简单的事实，“物体只会下落不会上升”，这类像“骑自行车要保持平衡”一样的常识，由预训练赋予模型记忆就足够了。但有些事实在预训练时是正确的，到了模型实际推理（inference）阶段可能已经过时，这时就需要通过某种方式更新这些事实，而“上下文”正是实现这种更新的机制。

当然，上下文的作用不只是更新知识。它还能承载 “隐私信息”——模型本身并不知道你的个人情况，也无法 “读心”，所以如果想让模型为你提供更贴合需求的服务，你需要将个人信息输入到上下文中，模型才能实现个性化响应。没有这些个性化信息，模型给出的只会是 “对所有人都通用” 的回答，而不是针对你定制的内容。

第三种需要通过上下文输入的知识是 “稀有事实”——也就是那些在互联网上出现频率极低的信息。不过我猜测，这类知识的 “必要性” 可能会随着时间逐渐消失：未来的模型或许能 “熟记” 整个互联网的所有数据，到那时我们可能就不需要再为 “稀有事实” 单独输入上下文了。但就目前而言，如果某个信息在整个互联网上只出现过一两次，模型基本不可能记住它，回答时很可能会 “幻觉生成”（hallucinate）错误内容。这种情况下，你就需要将这些稀有事实明确输入到上下文中。

我们面临的核心权衡是：对于短上下文模型，你能提供的额外上下文非常有限，不同知识来源之间会 “争夺” 有限的上下文空间；而如果上下文窗口足够大，你就不用太纠结 “该输入哪些信息”，可以纳入更多内容，从而提高相关知识的召回率（recall）和覆盖率。上下文覆盖率越高，“权重内记忆” 存在的那些问题（比如知识过时、缺失稀有信息）就能得到越多缓解。

Logan：没错，这个话题可以挖掘的角度太多了，你的解释非常清晰。接下来我想追问一个点：我们聊了“权重内记忆”和“上下文内记忆”，那第三种相关的技术是“检索增强生成（RAG，Retrieve Augmented Generation）”——也就是通过检索来引入上下文。你能先大致介绍一下 RAG 是什么吗？之后我还有几个关于“RAG 与长上下文对比”的尖锐问题想请教你。

Nikolay：当然可以。RAG 本质上是一种简单的工程技术，它在“将信息打包输入大型语言模型上下文”之前增加了一个额外步骤。具体流程是这样的：首先，你有一个知识语料库（knowledge corpus），把这个语料库拆分成一个个小型文本块；然后用专门的嵌入模型（embedding model）把每个文本块转换成实数值向量（real-value vector）；到了测试阶段，当你收到用户的查询（query）时，先把查询也转换成实数值向量，再将这个“查询向量”与语料库中所有“文本块向量”进行对比——找到与查询向量最接近的那些文本块，就相当于“找到了相关信息”；最后把这些相关文本块打包输入到模型上下文中，再让大型语言模型基于这个增强后的上下文进行生成。这就是 RAG 的工作原理。

Logan：可能这是个有点基础的问题，但我一直很好奇：RAG 的存在，是不是因为模型的上下文窗口有硬性限制？现在我们有 100 万、200 万令牌的上下文窗口，这已经很厉害了，但如果看互联网的规模 —— 比如维基百科（Wikipedia）就有数十亿甚至更多令牌（具体数量可能不是万亿，但数十亿是有的）。那为什么 RAG 这种 “为模型引入相关上下文” 的机制，没有直接内置到模型本身呢？是因为从研究方向上来说，让模型自己实现这种机制并不现实，还是说我们故意不把这种机制内置进去？我直观感觉，如果模型能自己做 RAG，或者我能给模型输入 10 亿令牌，让它通过某种机制自己筛选出相关令牌，应该会很有用。这是不是因为技术栈其他环节存在问题，所以模型本身不需要负责这件事？

Nikolay：有一点我想先说明：我们发布 Gemini 1.5 Pro 模型后，社交媒体上有很多讨论说 “RAG 要过时了”。但在我看来，事实并非如此 —— 比如企业知识库（enterprise knowledge bases）的规模通常是数十亿令牌，而不是数百万令牌，针对这种规模的场景，你仍然需要 RAG。

我认为未来的实际应用趋势是：RAG 不会马上被淘汰，而是会与长上下文 “协同工作”。长上下文能为 RAG 带来的好处是：通过 RAG 从语料库中检索出的 “相关信息点”（相当于 “针”），

可以更充分地被长上下文容纳，从而提高有用信息的召回率。以前因为上下文窗口小，我们需要设置比较保守的筛选阈值，导致很多潜在相关的文本块被排除；现在有了长上下文，我们可以更“宽松”地筛选——纳入更多可能相关的信息，所以长上下文和 RAG 之间其实是很好的互补关系。

真正的限制因素在于应用的延迟（latency）需求：如果你的应用需要实时交互，那你就只能用较短的上下文；但如果可以接受稍长的等待时间，长上下文会是更好的选择，因为它能提高信息召回率。

Logan：那 100 万令牌的上下文窗口，会不会只是一个“营销数字”？比如从长上下文的技术角度来看，是不是超过 100 万或 200 万令牌后，会有某种内在的技术变化？还是说，我们只是找了一个听起来很厉害的数字，然后让技术去适配它？

Nikolay：我刚开始做长上下文研究时，当时行业内的竞争水平大概是 12.8 万（128k）或最多 20 万令牌的上下文窗口。那时我在思考长上下文项目的目标——当时这个项目还是 Gemini 的一个小分支，我一开始想“只要追上竞争对手就行”，但这听起来不够有吸引力。所以我想设定一个更有挑战性的目标，然后就想到

“100 万令牌”——这个目标足够有突破性，相比当时的主流水平（比如 20 万令牌），相当于 5 倍的提升。而且在发布 100 万令牌版本后不久，我们很快又推出了 200 万令牌的版本，这相当于比之前的行业最佳水平提升了一个数量级（10 倍）。设定这样的目标很有意义，它能让团队成员更有动力投入工作。

Logan：太赞了。那接下来我再问一个尖锐点的问题：我们很快就从 100 万令牌迭代到了 200 万令牌，那继续突破 200 万令牌的限制，会面临哪些挑战？是因为部署（serving）成本太高，还是说让 100 万、200 万令牌生效的架构，在上下文规模更大时会彻底失效？为什么长上下文的技术前沿没有继续突破 200 万令牌？

Nikolay：我们发布 Gemini 1.5 Pro 模型时，其实做过 1000 万（10M）令牌上下文的推理测试，也拿到了相关的质量数据。

比如在 “单信息点检索”（single needle）或 “三信息点检索”（triple needle）任务中，模型在整个 1000 万令牌的上下文中表现几乎完美。我们其实可以发布这个 1000 万令牌的模型，但问题在于它的推理成本太高了——我们不确定用户是否愿意为这种

高成本服务付费，所以先推出了价格更合理的版本（100 万、200 万令牌）。

从质量角度来说，这也是个值得探讨的问题：因为 1000 万令牌模型的运行成本太高，我们没有做太多测试。而且重新启动相关服务器的成本也很高，所以除非现在有大量客户明确需要这个功能，否则我们暂时不会投入资源去部署它。

Logan：好的。那你觉得这种 “成本限制” 会一直存在吗？比如随着长上下文规模扩大，所需的计算资源会不会呈指数级增长？你有没有一种直觉 —— 要改变这种状况，是否需要研究层面的根本性突破？还是说 200 万令牌会成为主流，之后如果需要更大的上下文，就只能靠 RAG，通过智能筛选上下文来弥补窗口限制？

Nikolay：我的感觉是，我们确实需要更多技术创新。要实现 “接近完美的 1000 万令牌上下文”，光靠现有技术的缩放（scaling）是不够的，还需要更多创新。但至于未来 RAG 和长上下文哪种范式更强大，我认为模型的成本会逐渐下降，而且我们会尝试将越来越多的 “检索到的上下文” 直接输入模型 —— 因为随着模型质量提升，这么做能带来的收益会越来越大。

Logan：这很有道理。能不能跟我们聊聊最初实现长上下文功能的过程？我了解到的情况是，Gemini 1.5 Pro 一开始并不是为 “长上下文” 设计的 —— 你当时和深度思维的其他同事一起启动了这个工作流，但后来研究进展速度远超预期。我模糊听到的说法是：我们取得了技术突破，发现长上下文功能可行，之后很快就把它集成到了模型中。从项目启动到最终将长上下文功能集成到对外发布的模型中，整个时间线是怎样的？

Nikolay：确实，整个过程非常快。这里我想澄清一点：我们一开始就希望实现长上下文，比如 100 万或 200 万令牌的目标，但我们完全没料到会这么快达成。当突破真的出现时，我们都觉得 “这太不可思议了 —— 我们在这个任务上真的取得了重大进展”，所以当时就决定 “必须把这个功能落地”。之后我们迅速组建了一支非常出色的团队，团队成员都付出了极大的努力 —— 说实话，我这辈子从没见过有人工作这么拼命，真的让我印象深刻。

Logan：太棒了，这真的很鼓舞人心。长上下文功能最初在 Gemini 1.5 Pro 系列中落地，后来又应用到了 Gemini 1.5 Flash，现在

Gemini 2.0 Flash、2.5 Pro 也都支持了。现在很多人觉得 Gemini 2.5 Pro 的强大之处，很大程度上在于它的长上下文能力 —— 这对代码生成（coding）等场景特别有用。能不能给我们梳理一下，从最初发布长上下文功能到现在，整个长上下文领域发生了哪些变化？比如我们当时发布 Gemini 1.5 Pro 的技术报告，展示了 “大海捞针”（needle in haystack）任务的结果等，到现在这个时间点，行业和我们自身的技术都有哪些演进？从最初发布到现在，长上下文领域有哪些关键进展？

Nikolay：我认为最大的进步其实是 “质量提升” —— 无论是 12.8 万（128k）令牌还是 100 万令牌的上下文，模型质量都有显著改善。如果看 Gemini 2.5 Pro 的基准测试（benchmark）结果，会发现它优于很多强大的基线模型（baseline），比如 GPT-4.5、Claude 3.7，还有 Anthropic 的 Claude 3 Opus、DeepSeek 的部分模型等。

为了让对比更公平，我们在 12.8 万令牌的上下文下运行了所有模型的测试，结果显示 Gemini 2.5 Pro 的提升非常明显。而在 100 万令牌的上下文下，对比 Gemini 1.5 Pro，Gemini 2.5 Pro 也展现出了显著优势。

Logan：可能这个问题有点奇怪，但模型质量在不同上下文规模下会有波动吗？比如输入 10 万令牌和输入 12.8 万、5 万令牌时，质量是呈线性变化的，还是会有一些异常情况？我直觉上觉得，模型最终训练完成后应该能很好地泛化，不同上下文规模下质量不会有差异，但实际情况是否有一些细微差别？我们有没有做过相关测试来验证这一点？

Nikolay：我们内部做过一些测试。我猜你问这个问题，可能是因为之前行业内观察到的一种现象 ——“中间丢失效应”（lost in the middle effect）。关于你的问题，首先要说明的是：在我们的模型中，并没有观察到 “中间丢失效应”（即信息放在上下文中间时模型无法识别的情况）。但我们发现，如果任务难度较高 —— 不是 “单信息点检索” 这种简单任务，而是需要复杂指令的任务 —— 那么随着上下文规模增大，模型质量会有轻微下降。这是我们未来需要改进的方向。

Logan：好的。为了理清我的认知：如果我给模型输入 10 万令牌的上下文，从开发者或长上下文功能使用者的角度来看，我是否可以认为模型会 “关注到所有上下文内容”？我知道模型肯定能完成 “单信息点检索”，比如从大量上下文中找出某个特定信息，但它

是否真的会对所有令牌进行推理？我对模型处理海量上下文时的内部机制，一直没有一个清晰的认知。

Nikolay：这是个很好的问题。首先需要记住的是，注意力机制（attention）本身存在一个缺陷：令牌之间会存在 “注意力竞争” —— 如果一个令牌获得更多注意力，其他令牌获得的注意力就会减少。

具体来说，如果任务中存在 “强干扰信息”（hard detractors）—— 比如某个干扰信息和你要找的目标信息非常相似，它就会吸引大量注意力；

而你真正想找的信息，获得的注意力就会减少。上下文令牌越多，这种竞争就越激烈 —— 最终效果取决于任务中干扰信息的强度，以及上下文的整体规模。

Logan：再追问一个可能有点基础的问题：模型的 “注意力总量” 是固定的吗？有没有可能让模型分配更多注意力？还是说注意力总量本质上是 “1”，只能在所有上下文令牌中分配 —— 所以令牌越多，每个令牌能分到的注意力就越少，这是无法改变的？

Nikolay：通常情况下，注意力的总 “资源池” 是有限的。

Logan：明白了。你之前提到 “强干扰信息会导致模型分散注意力”，你们团队或应用端的其他团队，有没有探索过 “预筛选机制” 这类方法？比如为了让长上下文在生产环境中表现更好，是不是最好让上下文窗口中的数据尽可能不相似？如果上下文中有大量相似数据，而用户的问题又需要关联所有这些数据，模型的表现会不会普遍变差？这种情况下，是需要开发者自己想办法解决，还是你有什么建议可以分享？

Nikolay：站在研究者的角度，我认为 “依赖预筛选” 可能是个错误方向 —— 我们应该更专注于提升模型的质量和鲁棒性（robustness），而不是靠 “筛选” 这种权宜之计。不过有一个实用建议：尽量不要在上下文中加入完全无关的信息。如果你明确知道某些信息没用，为什么还要把它放进上下文呢？至少从成本角度来说，这会让推理成本更高，完全没必要。

Logan：这很有意思，因为这似乎和人们使用长上下文的常见方式相悖 —— 我在网上看到的例子，很多都是 “把一堆随机数据扔进上下文，让模型自己筛选有用信息”。

考虑到 “移除无关信息” 很重要，人们可能会期待模型自己做预筛选 —— 比如只保留相关信息，因为从用户角度来说，“不用自己筛选数据” 本身就是长上下文的一个卖点。

所以你觉得未来会不会出现一种 “多组件系统” —— 比如让某个子模型先根据用户查询，剔除无关数据，再把筛选后的上下文输入主模型？

Nikolay：随着模型质量提升和成本下降，未来你可能不再需要考虑 “筛选” 这件事。我现在说的是 “当前现状”：如果你现在想充分利用长上下文，就需要现实一点 —— 别把无关信息放进上下文。但我也同意你的观点：如果花太多时间手动筛选或定制上下文内容，会非常麻烦。所以关键在于找到一个平衡点。毕竟，上下文的核心价值是简化你的工作、实现自动化，而不是增加你的时间成本或让你手动处理繁琐的内容。

Logan：没错。接下来我们聊聊 “长上下文质量的评估指标”。“大海捞针”（needle in haystack）肯定是最基础的一个 —— 我们在 Gemini 1.5 的技术报告中也提到了这个指标。对于不了解的人来说，“大海捞针” 就是让模型从 100 万、200 万甚至 1000 万令牌的上下文中，找出某一个特定信息。现在模型在这个任务上表现已经非常好了。除了 “大海捞针”，你认为还有哪些 “标准基准测试” 可以用来评估长上下文能力？我感觉行业内对 “大海捞针” 讨论较多，但可能还有其他重要的评估维度。

Nikolay：我们来梳理一下。首先，评估（evaluation）本身是大型语言模型研究的基石 —— 尤其是对于大型团队来说，评估能让整个团队对齐方向、朝着共同目标努力，长上下文研究也不例外。要取得进展，就必须有完善的评估体系。

“单信息点大海捞针” 任务现在基本已经解决了 —— 尤其是在干扰信息较简单的情况下，这种任务对现在的模型来说毫无难度。

目前长上下文能力的 “前沿方向” 主要有两个：一是 “处理强干扰信息” —— 比如如果整个上下文都充满了 “XX 的魔法数字是 Y” 这样的键值对，再让模型找出 “巴塞罗那的魔法数字”，难度就会大得多，因为这些干扰信息和目标信息非常相似；二是 “多

信息点检索”——让模型从上下文中找出多个目标信息，这也是当前的难点。

此外，评估时还需要考虑其他因素。比如有人会说“即使是带强干扰的大海捞针任务，也太人工了，不够贴近真实场景”——这个观点有一定道理，但需要注意的是：一旦增加评估任务的“真实感”，可能就会失去“衡量长上下文核心能力”的有效性。举个例子：如果让模型处理一个大型代码库并回答问题，而这个问题其实只需要参考代码库中的一个文件，甚至需要模型完成复杂的代码实现——这时你考验的其实不是“长上下文能力”，而是“代码生成能力”，最终的评估信号会偏向“代码能力”而非“长上下文能力”，导致团队在优化时偏离方向。

另一个需要考虑的因素是“检索类评估”与“合成类评估”的区别。理论上，“从上下文中检索单个信息点”这种任务，RAG 也能完成；

我们真正应该关注的，是那些需要“整合整个上下文信息”的任务——比如文本摘要（summarization），这类任务 RAG 很难完成。但这类任务虽然方向正确，却很难用于“自动化评估”——比如摘要任务的评估指标（如 ROUGE）本身就不够完善。如果用这类指标来指导模型优化（即“爬山法” hill climbing），效果会不如那些“指标噪声更低”的任务。

Logan：那为什么摘要这类任务的评估指标实用性较低呢？是不是因为 “好摘要” 的标准更主观，没有明确的 “真值”（ground truth），所以这类任务的评估难度更大？

Nikolay：是的，这类评估的 “噪声” 会很大 —— 因为即使是人类撰写摘要，不同人之间的一致性也相对较低。这并不是说我们不应该研究摘要任务，也不是说不需要评估摘要能力 —— 这些任务本身非常重要。我只是从研究者的角度出发，更倾向于用 “信号更强”（即结果更明确、噪声更低）的任务来指导模型优化。

Logan：这很合理。接下来我们聊聊长上下文在 Gemini 生态中的定位：长上下文显然是 Gemini 向外界传递的 “核心能力” 之一，也是 Gemini 的重要差异化优势。但与此同时，长上下文的研发似乎一直是一个 “独立工作流” —— 好像 “长上下文” 是一个单独的领域，和其他领域（比如推理）相对独立。你觉得未来从研究或建模角度来看，长上下文会不会 “融入” 其他所有工作流？还是说它仍然需要作为独立工作流存在，因为 “让模型利用长上下文

完成有用任务” 和 “让模型提升推理能力” 在技术上有本质区别？

Nikolay：我的答案是 “两者皆有”。首先，每个重要能力都需要有明确的 “负责人团队”，这很有必要；其次，长上下文团队也需要为其他团队提供工具，让他们能参与到长上下文能力的优化中——这样才能形成协同效应。

Logan：完全同意。接下来我们聊聊 “推理与长上下文的关联”。之前杰克·雷（Jack Rae）来过我们的访谈，昨晚我还和他一起吃饭，聊了很多关于推理的话题。你有没有觉得 “推理能力的提升，让长上下文变得更有用”？如果是这样，这是正常的预期（比如模型 “思考时间更长” 所以表现更好），还是说推理能力和长上下文之间存在某种深层关联，能让两者协同发挥更大作用？

Nikolay：我认为两者之间存在深层关联。这种关联可以从 “下一个令牌预测任务”（next token prediction task）的特性来看：如果增加上下文长度能提升下一个令牌的预测效果，这可以从两个角度解读：

一是 “输入更多上下文，能让模型更好地预测短答案”；

二是 “输出令牌与输入令牌非常相似 —— 如果让模型将输出反馈到自身输入中，输出就相当于变成了新的输入”。

所以理论上，如果模型的长上下文能力足够强，它的推理能力也应该会得到提升。

另一个角度是：长上下文对推理至关重要。即使是像 “二选一” 这样只需生成一个令牌的简单决策，有时也需要先生成 “思考轨迹”（thinking trace）—— 原因与模型架构有关：如果在预测时需要在上下文中进行多次逻辑跳转，那么模型的 “注意力层数”（attention layers）会成为限制因素（它大致决定了模型能完成的逻辑跳转次数）。但如果让模型将输出反馈到输入中，这种限制就会消失 —— 模型可以 “写入自身记忆”，从而完成比仅依赖网络深度（network depth）更复杂的任务。

Logan：太有意思了。说到 “推理 + 长上下文”，我们俩之前都一直在推动 “长输出功能” 的落地 —— 开发者也确实需要这个功能。我经常看到有人反馈 “想要超过 8000 令牌的输出”，现在我们在一定程度上实现了：推理类模型支持 6.5 万令牌的输出，但有个前提 —— 其中很大一部分令牌是模型用于 “自我思考”

的，而非生成最终反馈给用户的内容。我想知道：“长输入上下文”和 “长输出能力” 之间有什么关联？比如很多用户需要的核心场景是 “输入 100 万令牌，然后对这些令牌进行重构”，未来这两种能力会不会融合成同一种能力？从研究角度来看，你认为它们是本质不同的能力，还是可以归为一类？

Nikolay：我认为它们并非本质不同。需要理解的是：模型在预训练完成后，本身并不存在 “生成大量令牌” 的限制 —— 比如你可以给模型输入 50 万令牌，然后让它 “复制这 50 万令牌”，我们实际测试过，这种操作是可行的。但这种 “长输出能力” 在训练后（post-training）阶段需要非常谨慎的处理，原因在于：训练后阶段会引入 “序列结束令牌”（end-of-sequence token，EOS）—— 如果你的有监督微调（SFT）数据都是短序列，模型就会学到 “序列结束令牌总会在较短的上下文位置出现”，进而形成 “在上下文长度 X 内生成 EOS 令牌并停止生成” 的习惯 —— 这本质上是一个 “对齐（alignment）问题”。

另外我想补充一点：推理只是 “长输出任务” 的一种。比如翻译也是一种长输出任务，而且推理任务有特殊的格式 —— 它会将 “思考轨迹” 封装在特定分隔符中，模型知道需要在这个范围内完成推理；但对于翻译任务，整个输出（而不只是思考轨迹）都需要

是长文本，这是我们希望模型具备的另一种长输出能力。所以，只要能正确对齐模型，长输出能力是可以实现的，我们目前也在推进相关工作。

Logan：太让人期待了，用户真的非常需要这个功能。这就引出了一个更广泛的话题：开发者在使用长上下文和 RAG 时，应该遵循哪些最佳实践？我知道你之前给我们的长上下文开发者文档提过很多反馈，文档中也已经包含了一些建议，但从你的角度来看，开发者在高效使用长上下文时，还有哪些核心建议？

Nikolay：第一个建议是“充分利用上下文缓存（context caching）”。我来解释一下这个概念：你第一次给模型输入长上下文并提问时，响应会比较慢、成本也较高；但如果基于同一上下文提出第二个问题，就可以通过上下文缓存来降低成本、加快响应速度。这是我们目前为部分模型提供的功能，建议大家尽量使用——比如将用户上传的文件缓存到上下文中，这样不仅处理速度更快，输入令牌的平均成本也会降低约 75%（即仅为原来的 1/4）。

Logan：为了让大家更好地理解，我举个例子，你看看我的理解是否正确：最适合使用上下文缓存的场景，应该是 “与文档对话”“与 PDF 对话” 这类 “与个人数据交互” 的应用 —— 因为这类场景中，原始输入上下文（比如文档内容）是固定的。而且我认为，使用上下文缓存的一个前提是 “原始上下文必须保持不变” —— 如果每次请求的输入上下文都不同，上下文缓存的效果就会大打折扣，因为你需要存储的原始上下文会不断变化，无法复用之前的缓存。

Nikolay：是的，你的两点理解都正确。上下文缓存特别适合 “与文档集合对话”“与长视频对话”“与代码库对话” 这类场景。而且你提到的 “上下文不变” 很关键 —— 如果上下文确实需要更新，最好在 “末尾添加新内容”，因为我们在后台会匹配 “与缓存前缀一致的部分”，只丢弃不一致的后缀。

另外有开发者会问：“应该把问题放在上下文之前，还是之后？” 答案是 “放在上下文之后” —— 如果你想利用缓存节省成本，这是最优选择。如果把问题放在上下文前面，每次提问都会导致缓存前缀变化，相当于每次都要重新开始缓存，无法复用之前的资源。

Logan：太棒了，这个建议非常实用。除了上下文缓存，开发者还需要注意哪些方面？

Nikolay：有几点我们之前已经提到过，比如“与 RAG 结合使用”——如果你的场景需要处理数十亿令牌的上下文，就必须结合 RAG；即使是处理较短上下文的场景，如果需要“多信息点检索”，结合 RAG 也会更有优势。

第二点是“不要用无关信息填充上下文”——这会影响多信息点检索的效果。

第三点与“权重内记忆”和“上下文内记忆”的交互有关：如果你想通过上下文更新模型的“权重内知识”，模型会同时接收到两种知识，可能会出现矛盾。这种情况下，建议通过“精心设计提示词”来明确解决矛盾——比如在提问开头加上“基于以上信息，请回答……”，这样可以提示模型“优先依赖上下文内记忆，而非权重内记忆”，帮模型消除歧义。

Logan：这个建议太有用了。你提到了“权重内记忆与上下文内记忆的冲突”，我们之前也聊过相关话题，现在我想追问“微调（fine-tuning）”的角度——行业内比“长上下文是否会淘

汰 RAG” 更有争议的话题，可能就是 “开发者是否应该做微调”。比如 Simon Williamson 就发过很多相关推文，质疑 “是否真的有人通过微调获得了实际收益”。你如何看待 “长上下文与微调的关联”？比如针对某类相似知识语料库，微调是否能让模型在长上下文任务中表现更好？微调是否能带来更通用的优化效果？

Nikolay：好的，我先具体说说 “如何针对知识语料库进行微调”。有时人们会获取额外的知识（比如企业的大型知识语料库，可能包含数十亿令牌），然后像预训练一样，通过 “语言建模损失（language modeling loss）” 让模型继续训练，学习在这个语料库上的 “下一个令牌预测”。这种方式确实能让模型整合语料库知识，但也存在局限性。

局限性之一是 “训练过程复杂”：你需要调整超参数、确定训练停止时机，还要处理过拟合（overfitting）问题。有些尝试过这种方法的人反馈，模型的幻觉生成现象会增加，他们认为这可能不是向模型输入知识的最佳方式。

当然，这种技术也有优势：推理时速度快、成本低 —— 因为知识已经融入模型权重，只需直接采样生成即可。但它也存在隐私隐患：知识被 “固化” 在模型权重中，如果后续需要更新这些知识，又

会回到最初的问题 ——“权重内知识难以修改”，最终还是需要通过上下文来补充更新。

Logan：没错，从开发者角度来看，这确实是个需要权衡的问题 —— 核心在于 “知识更新的频率”。我觉得 RAG 的成本其实还算合理：向量数据库（vector database）有很多成熟的解决方案，大规模部署起来效率也不错；但持续微调新模型的成本通常很高，这确实是个需要考虑的重要维度。

我还很好奇 “长上下文与微调的长期发展方向”—— 比如未来 3 年，长上下文领域会有什么变化？从你的经验来看，3 年后我们还会专门讨论 “长上下文” 吗？还是说 “长上下文” 会成为模型的基础能力，用户无需关注，只需默认它 “能正常工作”？

Nikolay：我可以做几个预测。首先，当前 100 万、200 万令牌上下文的模型质量会大幅提升，我们很快就能在几乎所有 “检索类任务” 上达到接近完美的效果。可能有人会问 “为什么不直接扩展上下文长度，而要停留在 100 万、200 万令牌”—— 关键在于 “当前中等长度的上下文（100 万 - 200 万令牌）还远未达到完美”。在质量未达标的情况下，盲目扩展长度意义不大；而当我

们实现 “接近完美的 100 万令牌上下文” 时，会解锁很多之前无法想象的应用 —— 模型处理信息、建立关联的能力会大幅提升。

目前模型已经能同时处理 “远超人类接收能力” 的信息 —— 比如让模型看完 1 小时的视频后，精准回答 “视频中第 X 秒有人掉落了一张纸”，这种任务人类很难精准完成，但模型可以。未来这种 “超人类能力” 会变得更普遍：长上下文质量越高，我们能解锁的 “意想不到的能力” 就越多 —— 这是第一步，即 “质量提升 + 接近完美的检索能力”。

第二步是 “长上下文成本下降”。这可能需要稍长时间，但一定会实现。随着成本降低，更长的上下文窗口也会逐步普及 —— 比如 1000 万令牌的上下文窗口，未来可能会成为 “常规配置”，像现在的 100 万令牌一样普遍。当这一天到来时，某些应用（比如代码生成）会迎来突破性变化：目前 100 万 - 200 万令牌的上下文，只能容纳中小型代码库；但 1000 万令牌的上下文，可以完整容纳大型代码项目。再结合 “全上下文接近完美召回” 的技术创新，代码生成应用会发生质的飞跃 —— 人类编码时需要在大脑中记忆大量信息，还要频繁在文件间切换，注意力范围有限；但模型可以一次性 “记住” 所有代码信息，精准复现任意部分内容，还能发现文件间的关联，成为效率极高的 “编码助手”。我相信很快就会出现 “超人类水平的 AI 编码系统”，它们将成为全球开发者的标配工具 —— 这就是 1000 万令牌上下文带来的改变。

至于 1 亿（100M）令牌的上下文，可行性争议更大。我认为它最终会实现，但时间不确定，而且可能需要深度学习领域的更多突破性创新。

Logan：太精彩了。针对你提到的这三个阶段（质量提升、成本下降、更长上下文），我还想追问：从你的角度来看，“硬件 / 基础设施” 和 “模型技术” 这两个因素，哪个更关键？显然，大规模部署长上下文需要大量工作，这也是长上下文成本高的原因之一。你在做研究时，会考虑硬件因素吗？还是说你认为 “硬件会自行发展”（比如 TPU 会持续升级），自己只需专注于模型研究？

Nikolay：只拥有芯片是不够的，还需要非常优秀的推理工程师（inference engineers）。我们团队的推理工程师所做的工作让我印象深刻 ——100 万令牌上下文的落地，离不开他们的努力。如果没有这么强大的推理工程师团队，我们不可能向客户交付 100 万、200 万令牌的长上下文功能。这背后需要巨大的推理工程投入，所以我认为 “硬件问题不会自行解决”。

Logan：我们的推理工程师确实一直在全力以赴，因为我们总想为模型增加更长的上下文，而这绝非易事。接下来聊聊 “智能体（agents）与长上下文的关联”—— 你认为长上下文是不是 “实现更优智能体体验” 的核心驱动力？两者之间存在怎样的互动关系？

Nikolay：这是个很有意思的问题。我认为智能体既是长上下文的 “消费者”，也是长上下文的 “提供者”。具体来说：

首先，智能体要主动运行，需要跟踪 “历史状态”（比如之前执行的动作、观察到的信息）和 “当前状态”，而要存储这些历史交互信息，就需要更长的上下文 —— 这是智能体作为 “长上下文消费者” 的一面。

其次，从另一个角度看，智能体也是 “长上下文提供者”—— 因为手动打包长上下文非常繁琐：比如每次都要手动上传文档、视频，或者从网上复制内容，没人愿意做这种事。你希望模型能自动完成，而实现这一点的方式之一就是 “智能体调用工具”（agentic tool calls）—— 模型可以自主决定 “此时需要获取更多信息”，然后自动抓取信息并打包成上下文。从这个角度来说，智能体为长上下文提供了 “自动生成来源”。

Logan：这个例子太贴切了。我个人也和很多人聊过这个话题，我认为这是当前人们与 AI 交互的主要痛点之一 —— 就像你说的，“手动找上下文” 是最麻烦的环节：我可能在屏幕上、电脑里已经有了相关上下文，但必须自己手动把它输入给模型，做所有 “体力活”。所以我特别期待未来能出现 “长上下文智能体系统” —— 它能自动从各种地方获取我的上下文，不用我手动操作。我觉得这能解决一个核心问题，无论是对开发者还是普通 AI 用户来说，都非常有价值。

Nikolay：让智能体自动处理上下文，绝对是未来的方向。

Logan：太赞同了，Niko。这次访谈非常精彩，感谢你的时间。希望我们以后能有机会面对面交流，也感谢你和长上下文团队付出的所有努力。期待未来能有更多令人兴奋的长上下文功能与大家分享。

Nikolay：感谢邀请，这次对话很愉快。谢谢！