DeepSeek-V3.2 技术报告整理与数据分析

> 说明：本报告基于 DeepSeek-V3.2 技术报告及其与 DeepSeek-V3 / V3.1 相关的公开资料，总结模型架构、训练方式与数据设计，并特别补充了 V3 / V3.1 在数据上的内容，以便整体理解整个系列的“数据谱系”。

目录

引言

DeepSeek-V3.2 是 DeepSeek 系列中在**推理能力、Agent 能力和推理成本**之间做平衡的一代关键模型。它不是从零重新设计，而是建立在 DeepSeek-V3 / V3.1 的架构与数据基础之上：

- 在架构上：仍然是 **decoder-only Transformer + MLA（Multi-head Latent Attention）+ MoE（Mixture-of-Experts）** 的主干，属于典型的大规模 MoE LLM；
- 在注意力机制上：引入了 **DeepSeek Sparse Attention（DSA）**，在不破坏原 MLA 结构的前提下做稀疏化，从而大幅降低长上下文推理的成本；
- 在训练方式上：采用 **继续预训练（continued pre-training）+ 专家蒸馏 + 大规模混合 RL** 的三段式流程；
- 在数据设计上：**底座语料继承自 V3 / V3.1 的 14.8T+0.84T 真实文本数据**，并叠加大量“可验证、高难度的 Agent / RL 数据”。

接下来将依次从架构、训练，再到数据进行系统梳理。

模型架构

2.1 总体架构概览

DeepSeek-V3.2 的整体仍是：

- **模型类型**：decoder-only 大语言模型；
- **内部结构**：
  - 大规模 **MoE（Mixture-of-Experts）**；
  - 基于 **MLA（Multi-head Latent Attention）** 的 KV 结构优化；
  - 新增 **DeepSeek Sparse Attention（DSA）** 作为“稀疏控制器”；

可以理解为：

> **DeepSeek-V3.2 = DeepSeek-V3.1-Terminus 架构 + 一个可学习的稀疏注意力层（DSA）**

其中：

- 与 DeepSeek-V3.2-Exp（更偏向“极致推理能力”的版本）结构完全一致；
- 与 DeepSeek-V3.1-Terminus 相比，唯一的结构修改就是加入 DSA，其余网络结构保持不变。

2.2 MLA + MoE 结构回顾

MLA（Multi-head Latent Attention） 是 DeepSeek 系列为了解决 KV 缓存成本与吞吐问题而引入的一种注意力结构。核心思想：

- 将 KV 表示压缩为一组“latent 向量”，多个 attention head 共用这些 latent；
- 在推理时，**以较小的 KV 存储成本换取多头注意力的表达能力**；
- 本质上是一种高效的 MQA / GQA 风格设计，但实现细节更加针对大规模 MoE 调优。

MoE（Mixture-of-Experts） 方面：

- DeepSeek-V3 系列采用的是大规模 MoE 架构：
  - 总参数量 ~671B；
  - 每个 token 激活约 37B 参数；
- 通过路由机制在多个专家之间选择少量专家进行计算，从而实现“参数量大但推理成本可控”。

DeepSeek-V3.2 在这些基础设计上没有做大的改变，而是重点在“如何让注意力更便宜、更聪明（DSA）”上做文章。

2.3 DeepSeek Sparse Attention（DSA）

DSA 是 DeepSeek-V3.2 中最重要的架构创新，目标是：

> 在**不牺牲长程依赖建模能力**的前提下，将注意力复杂度从 O(L²) 降到 O(L·k)，显著降低长上下文推理成本。

DSA 由两个主要组件构成：

2.3.1 Lightning Indexer（闪电索引器）

- 作用：对每个 query token，快速估计“当前 token 需要关注哪些历史 token”；
- 对于每个 query h_t 和所有历史 token h_s，计算一个索引分数 $I_{t,s}$，形成一个“重要性分布”；
- 特点：
  - 使用较少的 head，计算图轻量；
  - 支持低精度（如 FP8）实现，计算与显存开销都相对较小；
  - 输出是一个“稀疏化之前的注意力热度图”。

可以把 Lightning Indexer 理解为：

> 在主注意力层之前，先用一个廉价的“前座”粗略看一眼所有历史 token，决定真正需要精算的对象。

2.3.2 Fine-grained Token Selection（细粒度 token 选择）

- 依据 Lightning Indexer 的输出分数，对于每个 query：
  - 只选取 **Top-k 个最重要的 key/value token** 参与真正的 MLA 注意力计算；
- 结果：
  - 主注意力从原来的 O(L²) 降到 O(L·k)；
  - 索引器本身仍是 O(L²)，但计算量小很多（head 少、维度低、精度低），整体算力和显存占用显著降低。

这一步类似：

> “我先用低配模型看一遍所有 token，圈出最值得看的那一批，再用高配 MLA 在这些关键 token 上精算。”

2.3.3 在 MLA 中如何集成 DSA

为了兼容现有 MLA 的实现与优化，DSA 在下述约束下集成：

- 在 MLA 中以 **MQA 模式**实例化：
  - 一个 latent KV 向量会被所有 attention head 共享；
  - 这样可以最大化复用已有的高效 kernel 实现；
- KV 缓存结构不变，只是对“哪些 KV 参与注意力”做了一层稀疏筛选。

最终效果：

- 主模型中的注意力计算量大幅下降；

- 在长上下文长度（如 128K）下，推理成本曲线明显低于原始 dense attention；
- 在性能方面，通过精心训练与蒸馏，使得模型在主流基准上的表现与 dense 版本接近甚至持平。

2.4 与 DeepSeek-V3 / V3.1 的架构关系

架构层面可以这样概括三代模型：

- **DeepSeek-V3**
  - 引入大规模 MoE + MLA；
  - 支持 4K → 32K → 128K 的上下文扩展；
- **DeepSeek-V3.1**
  - 在 V3 架构基础上继续预训练（扩展 0.84T tokens），强化 128K 长上下文和“Thinking / Non-Thinking 混合模式”；
  - 架构仍是 MLA + MoE，没有引入 DSA；
- **DeepSeek-V3.2**
  - 在 V3.1-Terminus checkpoint 基础上继续训练；
  - 显式引入 DSA（Lightning Indexer + Top-k 选择），实现稀疏注意力；
  - 其它网络结构与 V3.1-Terminus 保持一致。

训练方式

整体训练流程可以分为三个阶段：

1. **继续预训练（continued pre-training）**：在 V3.1-Terminus 上加入 DSA，并训练索引器与主模型；
2. **专家蒸馏（Specialist Distillation）**：从多个领域专家模型蒸馏知识到统一大模型；
3. **混合强化学习（Mixed RL with GRPO）**：在多种真实 / 合成环境中进行统一 RL 训练，强化推理与 Agent 能力。

3.1 DSA 继续预训练流程

DSA 引入后，需要通过“继续预训练”来让模型适应新的注意力机制。训练分两阶段：

3.1.1 Dense Warm-up Stage（稠密预热阶段）

目标：**在保持主模型使用 dense attention 的前提下，只训练 Lightning Indexer，让它学会复刻原注意力分布。**

- 初始化：
  - 从 DeepSeek-V3.1-Terminus（已支持 128K 上下文） checkpoint 出发；
  - 保持主模型的 attention 仍是稠密注意力；
- 参数策略：
  - **冻结主模型全部参数**；
  - 只训练 Lightning Indexer；
- 训练信号：
  - 对主模型中所有 attention head 的注意力分布按 head 求和、L1 归一，得到 target 分布 $p_{t,:}$；
  - 用 Lightning Indexer 输出的索引分布 $q_{t,:}$ 与 $p_{t,:}$ 做 **KL 散度损失**，逼近原注意力热度；
- 训练配置（典型）：
  - 学习率：约 1×10^{-3}；
  - 步数：1000 steps；
  - 每步：16 个 128K 序列；
  - 总 token 规模：约 **2.1B tokens**。

这一阶段结束时，Lightning Indexer 已经“知道”原来的 dense attention 更关注哪些历史 token。

3.1.2 Sparse Training Stage（稀疏训练阶段）

目标：**启用稀疏 token 选择机制，同时微调主模型 + 索引器，让整体性能在稀疏模式下收敛。**

- 核心变化：
  - 启用 Top-k token selection，每个 query 只向前选取 **k = 2048** 个 KV token 参与 attention；
  - KL 对齐只对这些被选中的 token 进行（因为其它 token 直接被屏蔽）；

- 参数更新方式：
  - 主模型仍然优化语言建模损失（LM loss）；
  - Lightning Indexer 使用 KL 损失继续更新；
  - 为了稳定训练，在计算图上对索引器输入进行 **detach**，使索引器只收到 indexer loss，而不受 LM loss 干扰；
- 训练配置（典型）：
  - 学习率：约 7.3×10^{-6}；
  - 步数：15000 steps；
  - 每步：480 个 128K 序列；
  - 总 token 规模：约 **943.7B tokens**。

> 两阶段使用的数据分布与 V3.1 做 128K 长上下文扩展时保持一致，保证“模型变化来源于架构而不是数据偏移”。

3.2 专家蒸馏（Specialist Distillation）

在 DSA 继续预训练完成后，DeepSeek-V3.2 还要通过大规模“专家蒸馏”进一步提升：

3.2.1 整体流程

1. **从 unified base checkpoint 出发**：
   - 以统一大模型为起点（DeepSeek-V3.2 base）；
2. **在不同领域训练若干 specialist 模型**：
   - 写作 & 一般问答；
   - 六大领域：
     - 数学（math）
     - 编程（coding）
     - 一般逻辑推理（reasoning）
     - 通用 agent 任务（general agent）
     - agent coding
     - agent search
   - 每个领域都有 **thinking / non-thinking 两种模式**，并通过 RL 强化各自能力；
3. **由 specialists 生成高质量训练数据**：
   - thinking 模型生成带长链式思维的 CoT 数据；
   - non-thinking 模型生成直接回答的数据；
4. **统一蒸馏回一个大模型**：
   - 将各 specialist 在各自领域的“风格 + 能力”集中到统一 DeepSeek-V3.2 模型；
   - 实验表明：统一模型在各域性能仅略低于各 specialist，配合 RL 后差距进一步缩小。

可以形象理解为：

> 先培养一堆“学科尖子生”，让他们出题、写答案；
> 再把这些题库和标准答案拿来教一个“全科尖子生”。

3.3 混合强化学习（Mixed RL with GRPO）

DeepSeek-V3.2 的 RL 使用 **GRPO（Group Relative Policy Optimization）** 作为基础算法，并采用**混合 RL 策略**：

3.3.1 统一 RL 目标

在一个统一 RL 阶段中训练以下能力：

- 推理任务（数学、逻辑、代码推理等）；
- 各类 Agent 任务（search agent / code agent / general agent / code interpreter）；
- 对齐类任务（安全性、稳健性、忠实度等）。

通过统一 RL 训练而不是分阶段 RL，可以减轻灾难性遗忘问题，同时大幅简化系统工程。

3.3.2 奖励设计

- 对 **推理 / agent 类任务**：
  - 明确 outcome reward（如答案正确性）；
  - 加入长度惩罚，防止无意义长输出；

    - 引入语言一致性、格式正确性等辅助 reward；
- 对 **一般任务 / 对话类任务**：
    - 为每个 prompt 设计评价 rubric（维度如：有用性、礼貌性、安全性等）；
    - 使用生成式 reward 模型对模型输出打分，作为 RL 奖励。

训练 budget 方面，官方明确表示：

> RL 训练所消耗的算力已经超过了预训练算力的 10%，并且继续增加 RL budget 仍然带来推理能力的提升。

这与“单纯堆预训练数据”的路线不同，更强调“强 RL + 高质量奖励模型”的作用。

3.4 GRPO 稳定训练的关键技巧

为了在大规模 RL 训练中保持稳定，DeepSeek-V3.2 对 GRPO 做了一系列改进：

3.4.1 Unbiased KL Estimate（无偏 KL 估计）

- 经典的 Schulman K3 KL 估计在重要性采样比值很极端时会导致梯度偏差；
- DeepSeek 使用重要性采样修正 KL 估计，使得 KL 项的梯度近似无偏，从而避免在策略差异很大时出现梯度爆炸或方向失真。

3.4.2 Off-Policy Sequence Masking（脱轨序列屏蔽）

- 对于那些：
    - KL 偏离过大，且
    - advantage < 0 的序列，
- 直接在训练中对其做 mask，不让这些“过度偏离旧策略且表现差的样本”参与更新；
- 目的：防止 RL 过程被极端 bad case 拖偏。

3.4.3 Keep Routing（MoE 路由保持）

- 在 MoE 模型中，训练与推理可能会因为实现差异导致“路由路径不同”；
- DeepSeek 在 RL 中显式保持采样时的专家路由路径：
    - 采样阶段选择了哪些专家；
    - 更新阶段也沿用同样的专家，从而减小训练/推理不一致带来的抖动。

3.4.4 Keep Sampling Mask（采样截断 mask 保持）

- 在 RL 中保留采样时使用的 top-p / top-k 截断 mask；
- 使得旧策略和当前策略共享相同的可选 action 子空间；
- 这样重要性采样的比值不会因为“可选动作集合不同”而退化。

这些技巧合在一起，使得：

> 大规模 MoE LLM 在高维度、长轨迹、复杂奖励的 RL 场景下依然能够稳定训练，避免常见的 reward hacking 或性能崩塌。

3.5 DeepSeek-V3 / V3.1 与 V3.2 的训练关系

从“训练流程”的视角看三代模型的关系如下：

- **V3**：
    - 主体是 14.8T 实体世界语料 + math/code 加强；
    - 采用 MLA + MoE 架构，进行常规自回归预训练；
    - 通过阶段性长上下文扩展（4K → 32K → 128K）；
    - 在后续阶段加入基于 DeepSeek-R1 的蒸馏与 RL，以增强推理能力。
- **V3.1**：
    - 在 V3 的 14.8T 预训练基础上，额外加入约 0.84T tokens 继续预训练；
    - 重点强化：
        - 128K 长上下文稳定性；
        - Thinking / Non-Thinking 混合模式；
        - Agent / 工具调用链路。
- **V3.2**：
    - 以 V3.1-Terminus 为起点；

- 加入 DSA，并通过两个阶段继续预训练（总约 945.8B 长上下文 tokens）让模型适应稀疏注意力；
- 再通过专家蒸馏 + 大规模混合 RL，在 math / code / reasoning / agent / 对齐等任务上综合强化。

训练数据体系的完整分析

下面是本报告的重点——结合 V3 / V3.1 / V3.2 三代，系统梳理整个数据体系。

4.1 DeepSeek-V3：14.8T 预训练语料设计

4.1.1 规模与设计目标

DeepSeek-V3 的预训练语料规模为：

- **约 14.8T tokens**；

设计目标包括：

1. 提升 **数学（math）和编程（programming）样本的占比**；
2. 在保证中文 / 英文强势的同时，**进一步扩展多语种覆盖**；
3. 保持高质量与多样性，尽量减少低质、重复和无意义文本。

这些目标与 V3 的定位高度一致——一个“数学好、代码强、多语言通”的通用大模型基座。

4.1.2 数据来源与类型

从公开信息可知：

- 主体来源是：
  - 大规模 **Web 文本**（网页正文、技术博客、论坛等）；
  - **电子书**（books、technical manuals 等）；
- 其中包含大量 math / code 相关内容：
  - 编程教程、开源项目文档、技术问答；
  - 数学教材、论文、题解、博客等。

更重要的是，官方强调：

> 预训练阶段主要使用真实互联网与书籍语料，**没有刻意大规模使用其它大模型生成的数据**。
> 抓取的 Web 数据中可能混入一些 LLM 内容，但不是刻意“合成数据堆山”。

这一点在当下的 LLM 生态中比较少见——很多模型会主动在预训练阶段注入大量合成 CoT。DeepSeek 的路线更偏向：

> “**预训练尽量真实世界，推理模式靠后训练 + RL 来塑形。**”

4.1.3 清洗与结构化增强：去冗余、document packing 与 FIM

数据处理与结构化增强主要包括：

1. **去重与质量过滤**
   - 多轮去重，剔除高度重复的网页；
   - 过滤低质量内容（如过短、乱码、广告、脚本化垃圾文本等）；
   - 目标是“最小冗余 + 保多样性”。
2. **Document Packing**
   - 将多个短文档打包到同一个训练样本中，提高 token 利用率；
   - 与某些工作不同：DeepSeek 没有对不同文档间做严格的 attention mask，即允许模型在 pack 后的文档之间建立跨文档注意力；
   - 从建模角度来看，这有利于模型学习跨文档的长程依赖。
3. **FIM（Fill-in-the-Middle）策略**
   - 继承自 DeepSeekCoder-V2，将部分样本转换为“中洞补全”形式：

     ```
     <|fim_begin|> f_pre <|fim_hole|> f_suf <|fim_end|> f_middle <|eos_token|>
     ```

   - 约有 **10% 左右的样本使用 FIM 格式**；

- 尤其有利于代码补全 / 中间插入类任务。

4.1.4 多语言 & math / code 增强的含义

虽然官方没有给出精确百分比，但从效果与描述可推断：

- 数学与编程样本在 14.8T 语料中占比较高：
  - 不仅包含经典数学数据集，更有大量“自然文本形式”的数学内容；
  - 代码相关不仅是代码本身，还包含注释、文档、问答等配套自然语言；
- 多语言方面：
  - 在 V2 中已经强化了中文语料；
  - V3 中进一步扩展多国语言（欧洲多语种、亚洲语言等），提升多语种泛化能力。

总结一句：

> **V3 的数据可以理解为：**
> **“高质量 Web + 电子书作为底座 + math/code 加强 + 多语种扩展 + 结构化增强（packing & FIM）”。**

4.2 DeepSeek-V3.1：在 V3 之上的扩展训练

公开资料显示：

- **V3.1 继承了 V3 同一套 14.8T 预训练语料**；
- 在此基础上，额外加入约 **839B tokens** 的扩展训练数据，一般被拆分为两个阶段（约 630B + 209B tokens）。

可以将 V3.1 理解为：

> **“以 14.8T 底座 + 0.84T 扩展数据构成的增强版 V3”，**
> **主要目标是长上下文与思维模式升级。**

4.2.1 839B tokens 的主要作用（结合线索的合理推断）

官方没有给出 0.84T 数据的精细来源拆分，但结合现有信息，可以推断其主要服务于：

1. **长上下文进一步强化**
   - V3 已经通过多阶段训练实现 4K → 32K → 128K；
   - V3.1 在此基础上继续“长上下文 continued pre-training”，使模型在真实 128K 场景（长文档、多文件代码、多轮对话）下表现更稳定；
   - 这部分数据中应包含大量长文档、跨文档语料与多轮任务轨迹。
2. **混合思维模式（Thinking / Non-Thinking）打基础**
   - V3.1 被描述为“支持 Thinking / Non-Thinking 混合模式的 hybrid 模型”；
   - 这意味着 0.84T 数据中很可能加入了更多具有显式推理结构的文本（包括内部 CoT 风格数据），以便模型学会“何时展开思维链、何时直接给答案”。
3. **强化 Agent / 工具调用链路**
   - V3.1 的特性之一是更强的 Agent 能力；
   - 这部分数据应包含更多“工具调用轨迹”：搜索、代码执行、环境交互日志等，通常具有长上下文特征。

因此你在自己的报告中可以这样写：

> *“DeepSeek-V3.1 并未重新构造一套完全不同的通用语料，而是在 V3 的 14.8T 预训练数据基础上，通过额外约 0.84T tokens 的 continued pre-training，面向 128K 上下文与混合思维模式进行能力扩展，并补充更多带工具调用轨迹的长程任务数据。”*

4.3 DeepSeek-V3.2：继续预训练与后训练数据

DeepSeek-V3.2 的数据使用可以分为三层：

1. **DSA 继续预训练所用的长上下文数据**；
2. **专家蒸馏阶段由各 specialist 生成的蒸馏数据**；
3. **RL / Agent 阶段在真实 / 合成环境中构建的可验证任务数据**。

4.3.1 DSA 继续预训练数据

如前所述，DSA 的两阶段训练数据具有以下特点：

- **数据分布与 V3.1 长上下文扩展阶段保持一致**；
- 都是 128K 长上下文序列；
- 规模：
  - Dense warm-up：约 2.1B tokens；
  - Sparse training：约 943.7B tokens；
  - 总计近 **10^{12} 级别长上下文 token**。

这确保了：

> 架构改动（引入 DSA）是在“已知分布”的长上下文数据上完成的，性能变化可以主要归因于架构设计而非数据换血。

4.3.2 专家蒸馏数据

专家蒸馏带来的数据具有以下结构特征：

1. **任务 / 领域划分清晰**

   - 对应的任务域包括：
     - 写作 & 一般 QA；
     - 数学、编程、一般逻辑推理；
     - 通用 agent、agent coding、agent search；
   - 每个域内都有 thinking / non-thinking 两种数据形态。

2. **思维链与直接回答并存**

   - Thinking 模式：
     - 输出带显式推理链，一般包裹在类似 `<think> ... </think>` 标签中；
     - 适合作为 CoT 学习信号；
   - Non-thinking 模式：
     - 输出 concise answer，不展开推理；
     - 训练模型在需要时能短答。

3. **数据质量高、风格统一**

   - 由于是“从 RL 强化过的 specialist 模型中采样”；
   - 这些数据在每个领域内具有较统一的风格与较高的任务成功率；
   - 有利于统一模型进行“跨领域、多风格”的蒸馏学习。

从数据视角可总结为：

> **专家蒸馏阶段的数据是高密度、高质量的“领域内 teacher 输出”。**
> **它们专注于提升模型在特定领域的能力与风格，而不是泛化用语料。**

4.3.3 RL / Agent 阶段数据：真实环境 + 合成环境

这是 DeepSeek-V3.2 报告中最详细、也最有特色的一部分数据设计。

4.3.3.1 冷启动数据：Thinking in Tool-Use

- 利用已有的：
  - 非 Agent 推理数据（纯数学 / 逻辑推理 CoT）；
  - 非推理 Agent 数据（纯工具调用轨迹）；
- 通过系统 prompt 设计，将两类数据“拼接成一条轨迹”：
  - 既要求模型在 `<think>` 中进行多步推理；
  - 又要求模型在推理中调用工具（search / code / interpreter 等）；
- 从而构造出“推理 + 工具调用一体化”的冷启动数据；
- 虽然这类数据不完美，但足以作为后续真正 RL 的 seed 数据。

4.3.3.2 Agent 任务规模概览

RL 所用的 agent 任务数据大致由以下部分构成：

任务类型	任务数量	环境类型	Prompt 来源
code agent	24,667	real	extracted
search agent	50,275	real	synthesized
general agent	4,417	synthesized	synthesized
code interpreter	5,908	real	extracted

下面分别展开。

4.3.3.3 Search Agent 数据

- 环境：接入真实 Web 搜索 API；
- Prompt 来源：主要是合成的，配合从大规模 Web 语料中挖掘的 **long-tail 实体** 构造问题；
- 数据构建流程（多-Agent pipeline）：
  i. 采样长尾实体（如冷门人物、地名、专业术语）；
  ii. 由“问题构造 Agent”使用搜索工具探索实体信息，整理成 QA 任务；
  iii. 多个配置不同的回答 Agent 生成候选答案；
  iv. “验证 Agent”具备搜索能力，对候选答案进行事实核查；
  v. 只保留：**标准答案正确、所有候选答案错误**的样本，用于训练强化“正确查证”的能力；
  vi. 补充来自已存在 RL 数据中，对搜索明显有帮助的样本；
  vii. 对每个样本的回答由生成式 reward 模型按 rubric 打分。

特点：

- 题目多为“长尾知识 + 检索密集型”，对模型参数中已有的 world knowledge 不是很友好；
- 强调“如何用搜索工具查到对的东西，并正确整合成答案”。

4.3.3.4 Code Agent 数据

- 数据源：
  - 从 GitHub 上挖掘大规模 issue-PR 对；
  - 过滤得到“issue 描述合理 + patch 确实修复 bug + 有可执行测试”的样本；
- 环境构建：
  - 使用自动环境构建 Agent 负责：
    - 拉取代码；
    - 安装依赖；
    - 配置并运行测试；
  - 统一将各语言的测试结果转换为 JUnit 格式；
  - 若“应用 gold patch 后：
    - 新增 failing-to-passing 测试（F2P>0）；
    - 且不存在 passing-to-failing 测试（P2F=0）；
    - 则该任务被视为“环境构建成功”；
- 任务覆盖语言：
  - Python、Java、JavaScript / TypeScript、C/C++、Go、PHP 等。

特点：

- reward 完全可以由“测试是否通过”自动给出；
- 是典型的“**难度高且可高度验证**”的数据；
- 对 Terminal Bench、SWE-bench 等实际工程级 coding benchmark 提升显著。

4.3.3.5 Code Interpreter 数据

- 环境：Jupyter Notebook；
- 数据：涵盖数学、逻辑、数据分析等需要“代码 + 推理”解决的问题；
- 模型通过工具：
  - 写代码；
  - 执行代码；
  - 解析结果；
- 用于训练模型学会在复杂推理中“借助代码提高可靠性”。

4.3.3.6 General Agent 合成环境

- 通过自动化的环境构建 Agent 合成：
  - **1,827 个任务导向环境 + 4,417 个任务**；

- 合成流程要点：
  i. 根据类别（如“旅行规划”），在带 Bash + Search 的 sandbox 中抓取 Web 数据构建数据库；
  ii. 自动合成一批任务相关工具函数，如查询票价、筛选酒店、计算总成本等；
  iii. 对于每一个任务：
    - 合成一个 solution function（仅能使用工具函数和逻辑，不能直接看数据库）；
    - 合成一个 verification function，用来检查 solution 的输出是否满足约束；
    - 不断自动修改 solution 直至 verifier 通过；

  iv. 逐步提高任务复杂度（更多约束条件、更多工具组合），生成大量“复杂但可验证”的任务。

这些合成环境的设计原则是：

> **难度高（对闭源 SOTA 模型也具挑战） + reward 可自动化计算。**

实验显示：只用这些 general synthetic agent 数据做 RL，即便不显式训练现实世界环境，也能使模型在 Tau2Bench、MCP-Mark、MCP-Universe 等真实 agent benchmark 上明显提升。

4.4 三代模型在“数据观”上的共性与差异

综合来看，DeepSeek-V3 / V3.1 / V3.2 在数据设计上呈现出以下图景：

4.4.1 共性：预训练真实世界、后训练大规模合成 + RL

- **预训练阶段**
  - 以真实世界语料为主（Web + 电子书）；
  - math / code / 多语种明确增强；
  - 尽量避免大规模引入其它 LLM 生成的合成数据；

- **后训练阶段**
  - 把大量合成 / CoT / Agent 数据放在 SFT + RL 阶段；
  - 通过 R1 等“强化推理模型”产出 teacher-style CoT；
  - 再通过 RL 和蒸馏塑形最终行为。

一句总结：

> **DeepSeek 更像是在“真实世界预训练 + 强 RL 后训练”这条路线走到极致。**

4.4.2 差异：V3 → V3.1 → V3.2 的数据演化

- **V3**
  - 14.8T 高质量真实语料；
  - 强调 math / code / 多语种；
  - 结构化增强：document packing + FIM；
  - 初步引入 R1 蒸馏与 RL 推理增强。

- **V3.1**
  - 在 V3 基础上增加 0.84T tokens 继续预训练；
  - 重点是：
    - 128K 长上下文稳定训练；
    - Thinking / Non-Thinking 混合模式；
    - 更长、更复杂的工具调用轨迹数据。

- **V3.2**
  - 进一步在 V3.1-Terminus 上使用约 945.8B 长上下文 token 进行 DSA 继续预训练；
  - 大规模使用专家蒸馏数据（各领域 specialist 输出）；
  - 在搜索、代码、通用 Agent、code interpreter 等任务上构造十万级可验证 RL 数据；
  - RL 训练算力投入达到预训练的 10% 以上。

总结与对你项目的启发

从 DeepSeek-V3.2 及其前两代的技术与数据设计中，可以抽象出几条对你有用的经验（尤其是你在做 VLM / VLA 和云端推理模型）：

1. **预训练阶段尽量“干净 + 真”，后训练阶段再疯狂玩“合成 + RL”**
   - DeepSeek 在预训练时坚持以真实世界文本为主体，避免被其它模型的错误知识污染；
   - 推理模式、长链 CoT、工具调用策略则主要在 SFT + RL 阶段塑造；
   - 对你自己做 VLM / VLA 来说，也可以考虑把“合成多步推理轨迹”集中到后训练阶段进行。
2. **长上下文能力不要寄希望于“多给点长文档就完事”，而要有明确的扩展 / 微调阶段**
   - V3 → V3.1 → V3.2 的长上下文演化，几乎每一步都有“专门为长上下文设计的 continued pre-training 阶段”；
   - 对你做长视频 / 多视图 / 多轮交互 VLM 时，可以借鉴这种“专门的长上下文阶段”，而不是在主预训练里混着做。
3. **Agent / 工具调用数据的关键是“可验证 + 足够难”**
   - 搜索 / 代码 / 泛 Agent 任务，都围绕两个核心：
     - 真实或完整的环境（包括合成环境）；
     - 严格、可自动计算的验证函数；
   - 这对你在做工业 / 机器人 / 车载助手时同样成立：
     不一定要一上来就用真实物理环境，大量“合成但可验证”的任务环境可以先让模型学会“规划 - 调用工具 - 验证 - 修正”的闭环。
4. **如果你要复刻一套类似的数据系统，可以考虑这样的优先级：**
   - 第一层：构建“高质量真实世界语料 + math/code 强化”的底座（对应 V3）；
   - 第二层：针对你关心的场景（如车载 cabin、多模态安全巡检），设计专门的长上下文 continued pre-training 数据（对应 V3.1 / DSA 的 continued pre-training）；
   - 第三层：搭一套“真实 + 合成”的工具环境（logs、API 仿真、task planner 环境）构造 RL / Agent 数据（对应 V3.2 的 RL 部分）。

如果你接下来希望我帮你做的，是“在你自己的项目上，对标 DeepSeek 的这一整套数据策略”，我们可以继续往下拆成：

- 你的 VLM/VLA 需要哪些“真实底座数据”；
- 哪些部分适合放在“长上下文 continued pre-training”阶段；
- 哪些场景可以用“合成环境 + 自动验证”方式构造大规模 RL 任务。

这样就能从“读 paper 很爽”走到“自己 pipeline 更强”这一步。