



deepseek

Day2: DeepEP

深度解读



ZOMI



DeepSeek ✓
@deepseek_ai



 Day 2 of #OpenSourceWeek: DeepEP

Excited to introduce DeepEP - the first open-source EP communication library for MoE model training and inference.

- ✓ Efficient and optimized all-to-all communication
- ✓ Both intranode and internode support with NVLink and RDMA
- ✓ High-throughput kernels for training and inference prefilling
- ✓ Low-latency kernels for inference decoding
- ✓ Native FP8 dispatch support
- ✓ Flexible GPU resource control for computation-communication overlapping

 GitHub: [github.com/deepseek-ai/De...](https://github.com/deepseek-ai/DeepEP)

DeepSeek 开源 DeepEP

- DeepEP 提供高吞吐量和低延迟的 all-to-all Hopper 架构 GPU Kernel, 包括 MoE dispatch and combine。支持 FP8 低精度运算, 特别适用于 DeepSeek 系列模型。
- Github 开源地址: <https://github.com/deepseek-ai/DeepEP>
- 本 PPT 开源: <https://github.com/chenzomi12/AllInfra/>
- 夸克链接: <https://pan.quark.cn/s/374bc7960241>



DeepSeek 开源 DeepEP

1. 高效优化的 All-to-All 通信
2. 支持 NVLink 和 RDMA 的节点内 / 跨节点通信
3. 训练 Training 及推理预填充 Prefill 阶段的高吞吐量计算核心
4. 推理解码 Decoder 阶段的低延迟计算核心
5. 原生支持 FP8 数据分发
6. 灵活控制 GPU 资源，实现计算与通信的高效重叠



视频目录大纲（上）

1. DeepSeek MoE: MoE 架构通信
2. MoE Demo: 原理与实现
3. DeepEP 使用核心工具 (Hopper & NVSCHMEM)
4. DeepEP 之前是怎么用的?



视频目录大纲（下）

1. DeepEP 之前是怎么用的?
2. DeepEP: 项目基本介绍
3. DeepEP: 核心代码理解
4. DeepEP: 代码注释与走读
5. 思考与小结



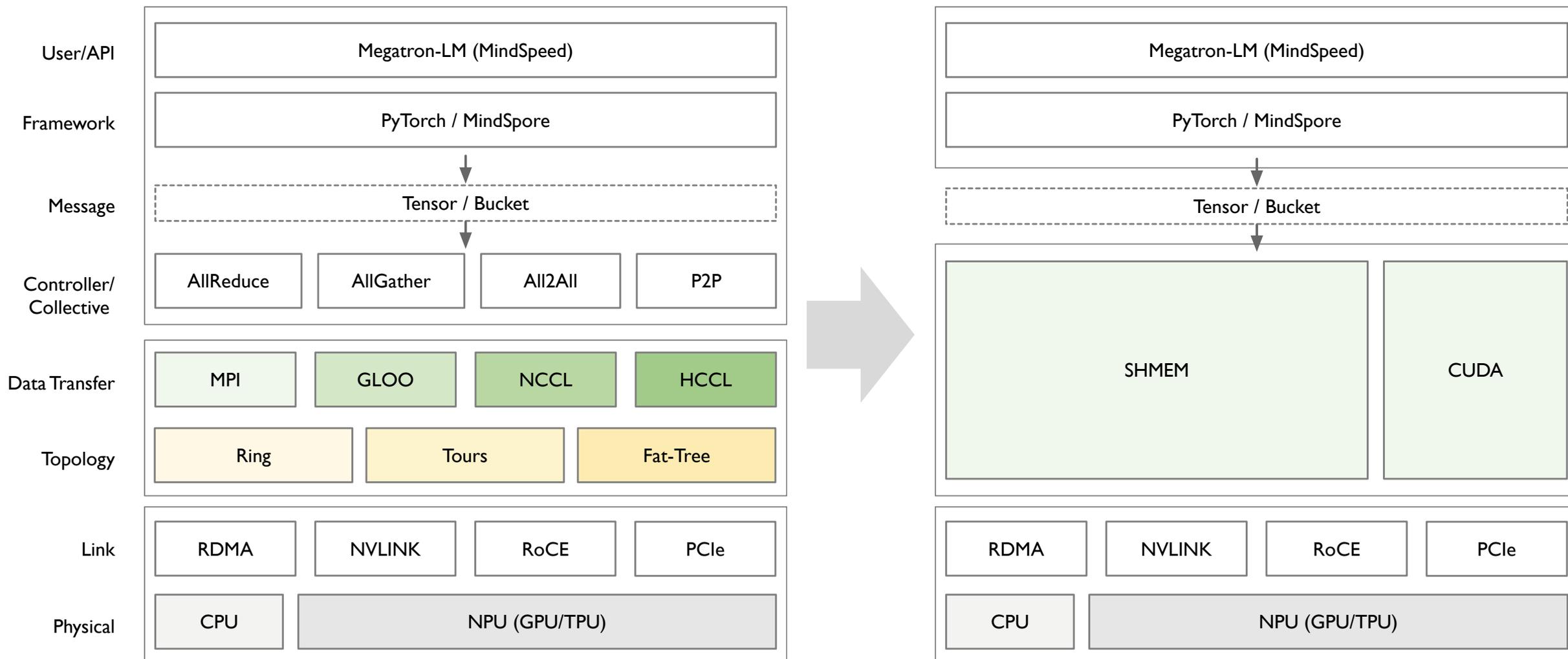
01

DeepEP

Before & After



XCCL 在 AI 系统中的位置



NCCL vs NVSHMEM

特性	NCCL	NVSHMEM
主要用途	针对集合通信（如AllReduce、Broadcast）优化，专为深度学习分布式训练设计。	基于PGAS模型的细粒度内存访问，支持任意GPU/节点间的直接内存读写。
设计哲学	强调高吞吐、低延迟的集合操作，适合紧密同步的并行任务。	提供全局地址空间抽象，支持灵活的异步通信，适合非规则或动态通信模式。
信协议	基于NVIDIA NVLink/InfiniBand优化集合通信算法（如Ring AllReduce）。	利用PGAS模型和CUDA-aware技术，直接操作远程内存地址。



NCCL vs NVSHMEM

特性	NCCL	NVSHMEM
通信粒度	基于集体操作（如AllReduce、AllGather），需要所有进程参与同一操作。	支持单边通信（Put/Get/Atomics），允许单个GPU直接读写远程内存。
同步机制	隐式同步，操作完成后自动保证数据一致性。	需显式同步（如nvshmem_fence或nvshmem_quiet）确保内存可见性。
编程范式	通过显式调用通信函数（如ncclAllReduce）。	类似共享内存的地址直接访问（如nvshmem_put）。



02

DeepEP

项目基本介绍



NVSCHMEM

- <https://github.com/deepseek-ai/DeepEP>



03

DeepEP:

代码注释与解读



internode_ll.c 特点

1. 低延迟通信:

- 基于 IBGDA 提供低延迟的跨节点通信功能, 适用于对通信延迟要求极高的场景。

2. 异步通信:

- 通信过程异步, 发送方和接收方可以快速建立连接并直发数据。

3. 数据分发与合并:

- 融合 forward pass 和 backward pass 通信, 通过分片和融合操作减少通信开销。

4. 数据打包与解包:

- 通过 pack2() 和 unpack2() 将数据打包成特定 FP8 格式, 高效传输处理。

5. 内存栅栏:

- 使用 nvshmem 内存栅栏操作, 确保线程间内存可见性和顺序。



internode_ll.c 代码

```
20  
21 // 获取当前  
22 // Clean  
23 auto thre  
24  
25 // 清理 cl  
26 // 使用循环  
27 #pragma u  
28 for (int  
29 | clean  
30  
31 // 清理 cl  
32 #pragma u  
33 for (int  
34 | clean  
35
```



internode_ll.c 数据分发 dispatch

- **分片处理:**

- 将数据分片后通过 IBGDA 接口异步发送到目标节点。e.g. 使用 `nvshmemi_ibgda_put_nbi_warp` 函数将数据分片发送到远程节点。

- **专家计数:**

- 通过同步和异步操作，计算每个 expert 接收的 Token 数量，并确保数据按需分发和选择。

- **缓冲区管理:**

- 使用共享内存 (`__shared__`) 来存储专家接收的 Token 数量，并使用 `atomicAdd` 和 `atomic_add_release_global` 等操作同步和更新计数。



internode_ll.c 数据合并 combine

- **数据接收:**

- 从远程节点接收数据分片，并将其合并到本地缓冲区。 e.g. 通过 `ld_nc_global` 和 `st_na_global` 函数高效地读取和写入数据。

- **同步机制:**

- 使用 `nvshmemi_ibgda_poll_recv` 和 `cg::this_grid().sync()` 同步操作，确保所有节点完成数据传输后再进行合并操作。

- **FP8 和 BF16 转换:**

- 数据合并过程中，将 FP8 数据转换为 BF16 格式，以便进行高效的计算和存储。



intranode.c

- DeepEP 中实现节点内通信，通过高效的同步机制、缓冲区管理、原子操作和异步通信等技术，实现了数据在节点内不同 GPU 之间的高效传输和处理。核心实现逻辑：
 1. 通知调度 (notify_dispatch 函数)
 2. 数据分发 (dispatch 函数)
 3. 数据合并 (combine 函数)

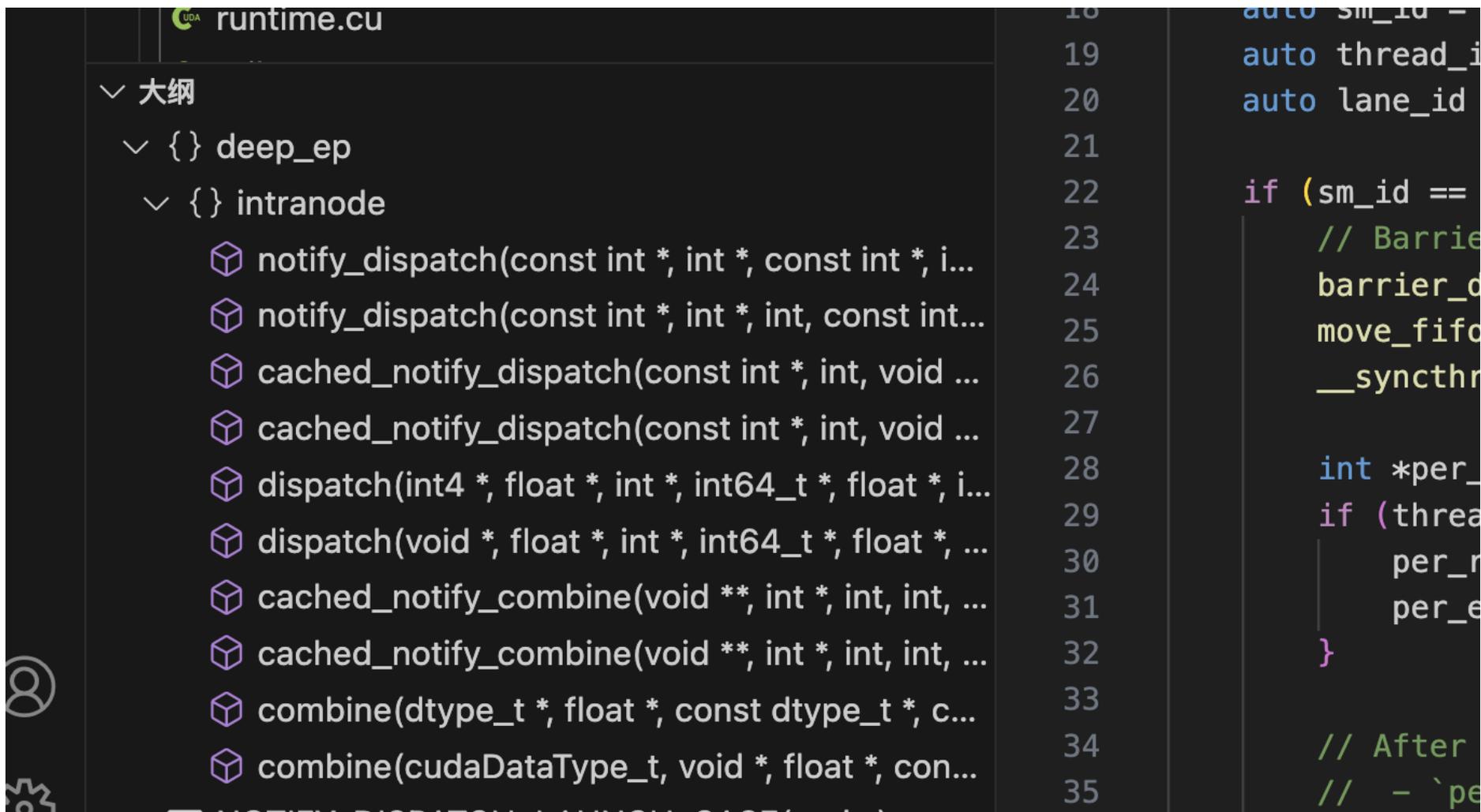


intranode.c

- **任务同步**: 使用 `barrier_device` 和 `move_fifo_slots` 函数进行同步操作, 确保所有线程在进入关键区域之前完成前一个任务。
- **缓冲区管理**: 通过 `buffer_ptrs` 和 `task_fifo_ptrs` 管理缓冲区和任务队列, 确保数据在传输过程中的正确性和一致性。
- **计数计算**: 计算每个节点接收的令牌数量和专家计数, 通过 `atomicAdd` 和 `atomic_add_release_global` 等原子操作更新计数。
- **异步通信**: 发送方和接收方可以快速建立连接并直接发送数据, 无需显式发送消息中的地址信息, 提高通信效率。
- **数据分发与合并**: 通过 `dispatch` 和 `combine` 函数, 实现数据的分发和合并, 支持复杂的通信模式和数据处理。



intranode.c 代码



The image shows a code editor window with a file explorer on the left and a code editor on the right. The file explorer shows the following structure:

- runtime.cu
 - 大纲
 - { } deep_ep
 - { } intranode
 - notify_dispatch(const int *, int *, const int *, i...
 - notify_dispatch(const int *, int *, int, const int...
 - cached_notify_dispatch(const int *, int, void ...
 - cached_notify_dispatch(const int *, int, void ...
 - dispatch(int4 *, float *, int *, int64_t *, float *, i...
 - dispatch(void *, float *, int *, int64_t *, float *, ...
 - cached_notify_combine(void **, int *, int, int, ...
 - cached_notify_combine(void **, int *, int, int, ...
 - combine(dtype_t *, float *, const dtype_t *, c...
 - combine(cudaDataType_t, void *, float *, con...

The code editor shows the following C code snippet:

```
18 auto sm_id =
19 auto thread_i
20 auto lane_id
21
22 if (sm_id ==
23     // Barrie
24     barrier_d
25     move_fifo
26     __synthr
27
28 int *per_
29 if (threa
30     per_r
31     per_e
32 }
33
34 // After
35 // - `pe
```

intranode.c 通知调度 notify_dispatch()

- **任务同步:**

- 使用 `barrier_device` 和 `move_fifo_slots` 函数进行同步操作, 确保所有线程在进入关键区域之前完成前一个任务。

- **缓冲区管理:**

- 通过 `buffer_ptrs` 和 `task_fifo_ptrs` 管理缓冲区和任务队列, 确保数据在传输过程中的正确性和一致性。

- **计数计算:**

- 计算每个节点接收的令牌数量和专家计数, 通过 `atomicAdd` 和 `atomic_add_release_global` 等原子操作更新计数。



intranode.c 数据分发 dispatch()

- **分片处理:**

- 将数据分片后通过异步操作发送到目标 GPU。例如，使用 `nvshmemi_ibgda_put_nbi_warp` 函数将数据分片发送到远程节点。

- **异步通信:**

- 发送方和接收方可以快速建立连接并直接发送数据，无需显式发送消息中的地址信息，提高通信效率。

- **数据转换:**

- 在数据发送过程中，对数据进行必要的转换和打包，例如将浮点数据转换为更高效的格式（如 `nv_bfloat16`）。



intranode.c 数据合并 combine()

- **数据接收:**

- 从远程节点接收数据分片，并将其合并到本地缓冲区。通过 `ld_nc_global` 和 `st_na_global` 函数高效地读取和写入数据。

- **同步机制:**

- 使用 `nvshmemi_ibgda_poll_recv` 和 `cg::this_grid().sync()` 等同步操作，确保所有节点完成数据传输后再进行合并操作。

- **FP8 和 BF16 转换:**

- 在数据合并过程中，将 FP8 数据转换为 BF16 格式，以便进行高效的计算和存储。



04

DeepEP:

代码走读



internode_ll.c



End

总结与思考



DeepSeek 第二天开源 DeepEP

1. 对大模型厂商、互联网厂商带来哪些影响和思考?
2. 对国产芯片厂商带来哪些影响和思考?
3. 对 AI 产业带来哪些新的变化?



DeepSeek 第二天开源 DeepEP

对大模型厂商、互联网厂商带来哪些影响和思考？

- DeepEP 直接使用底层 NVSHMEM 编程接口进行 Infra 加速，而不是 NCCL 等，让中台压力大；
- 优化 GPU 性能和降低信息传递延迟，提升了模型训练和推理的效率，有助于厂商降低计算成本；
- 技术路径重构：从“堆算力”到“算法-通信协同优化”；



DeepSeek 第二天开源 DeepEP

对国产芯片厂商带来哪些影响和思考??

- DeepEP或成为行业通信库的新标准, 迫使厂商在开源基础上差异化竞争;
- CUDA生态兼容性参考: DeepEP的开源为国产GPU 提供了新通信协议优化的方案;



DeepSeek 第二天开源 DeepEP

1. 对 AI 产业带来哪些新的变化? ? ?

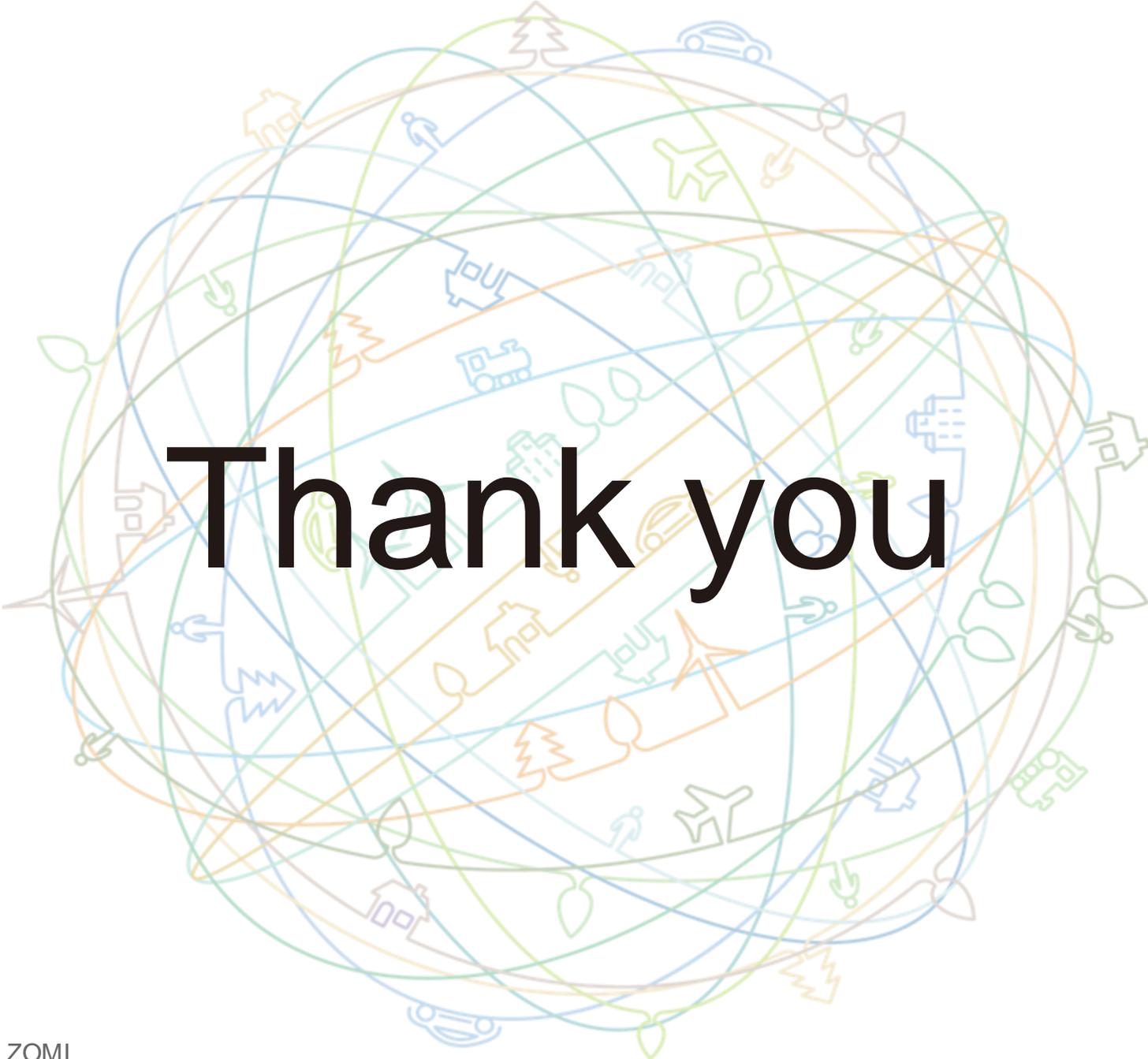
- 加速MoE架构普及: DeepEP解决了MoE模型中专家间通信的瓶颈, 支持千亿参数模型的高效扩展;
- DeepEP的低成本特性可能引发大模型服务价格战;
- “效率驱动” 替代 “规模驱动”: DeepSeek的算法优化推动行业从堆算力转向智能优化;



引用与参考

- Github 开源地址: <https://github.com/deepseek-ai/FlashMLA>
- 本 PPT 开源: <https://github.com/chenzomi12/AllInfra/tree/main/06AlgoData>
- 夸克链接: <https://pan.quark.cn/s/374bc7960241> (代码、论文、注释)





Thank you

把AI系统带入每个开发者、每个家庭、
每个组织，构建万物互联的智能世界

Bring AI System to every person, home and
organization for a fully connected,
intelligent world.

Copyright © 2024 XXX Technologies Co., Ltd.
All Rights Reserved.

The information in this document may contain predictive statements including, without limitation, statements regarding the future financial and operating results, future product portfolio, new technology, etc. There are a number of factors that could cause actual results and developments to differ materially from those expressed or implied in the predictive statements. Therefore, such information is provided for reference purpose only and constitutes neither an offer nor an acceptance. XXX may change the information at any time without notice.



GitHub github.com/chenzomi/2/Allinfra