ByteDance | Seed

VeOmni: Scaling Any Modality Model Training with Model-Centric Distributed Recipe Zoo

Qianli Ma[1,*,†], **Yaowei Zheng**[1,*], **Zhelun Shi**[1,*], **Zhongkai Zhao**[1,*], **Bin Jia**[1,*], **Ziyue Huang**[1,*], **Zhiqi Lin**[1], **Youjie Li**[1], **Jiacheng Yang**[1], **Yanghua Peng**[1,†], **Zhi Zhang**[1,†], **Xin Liu**[1,†]

[1]ByteDance Seed

[*]Equal Contribution, [†]Corresponding authors

Abstract

Recent advances in large language models (LLMs) have driven impressive progress in omni-modal understanding and generation. However, training omni-modal LLMs remains a significant challenge due to the heterogeneous model architectures required to process diverse modalities, necessitating sophisticated system design for efficient large-scale training. Existing frameworks typically entangle model definition with parallel logic, incurring limited scalability and substantial engineering overhead for end-to-end omni-modal training. We present VeOmni, a modular and efficient training framework to accelerate the development of omni-modal LLMs. VeOmni introduces model-centric distributed recipes that decouples communication from computation, enabling efficient 3D parallelism on omni-modal LLMs. VeOmni also features a flexible configuration interface supporting seamless integration of new modalities with minimal code change. Using VeOmni, a omni-modal mixture-of-experts (MoE) model with 30B parameters can be trained with over 2,800 tokens/sec/GPU throughput and scale to 160K context lengths via 3D parallelism on 128 GPUs, showcasing its superior efficiency and scalability for training large omni-modal LLMs.

Date: August 5, 2025
Correspondence: maqianli.fazzie@bytedance.com
Project Page: https://github.com/ByteDance-Seed/VeOmni

1 Introduction

The evolution of large language models (LLMs) has progressed from unimodal specialization towards unified omni-modal understanding and generation [38, 55, 67, 68]. Recent models such as GPT-4o [28] and BAGEL [11] exhibit strong performance across diverse multimodal tasks. These tasks include visual question answering [1, 21], controllable image generation [35, 75], and multimodal reasoning [26, 52, 66], highlighting the growing potential of LLMs as general-purpose omni-modal agents.

As LLMs are extended to handle diverse modalities, their architectures have become increasingly heterogeneous and complicated. State-of-the-art omni-modal LLMs typically incorporate multiple modality-specific pre-trained networks to process inputs with fundamentally different properties, such as continuous signals (*e.g.*, images, audio) and discrete sequences (*e.g.*, text). In these omni-modal models, a language model often serves as the central backbone [3, 61], connecting with vision encoders [15], audio encoders [14], and image

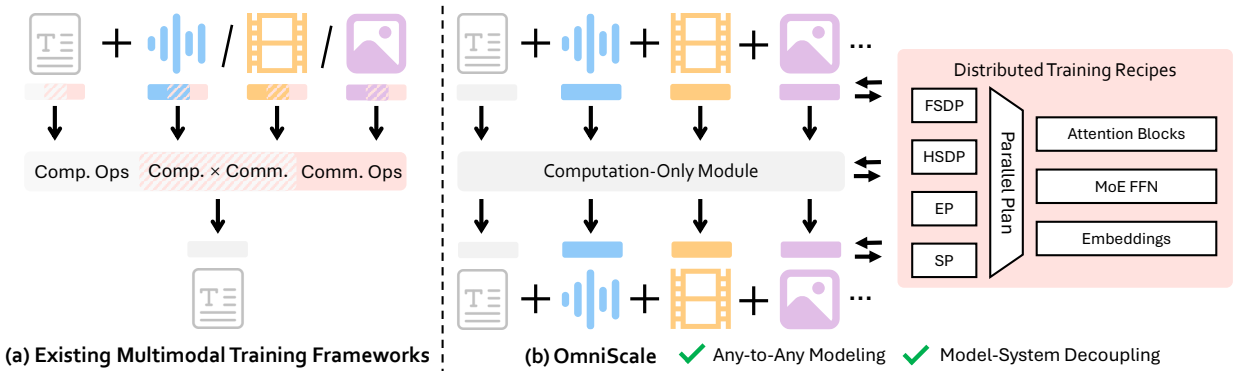

Figure 1 Comparison between VeOmni and Existing Training Frameworks.

generation networks [20, 22] through learned bridging mechanisms.

Despite these advances, the development of omni-modal LLMs still lags behind their unimodal counterparts. This gap is largely attributed to the absence of scalable training frameworks tailored for omni-modal tasks. While numerous mature and widely adopted systems exist for training language models on text-to-text tasks [33, 36, 40, 41, 50, 51, 62], few frameworks are designed to support any-to-text tasks [17, 24, 30, 77]. Crucially, none of these frameworks enable end-to-end training for fully omni-modal scenarios, *i.e.*, any-to-any learning, revealing the necessity for scalable infrastructure to build the next generation of omni-modal LLMs.

Extending existing training frameworks to omni-modal LLMs is non-trivial. To address the growing parameter sizes, modern training frameworks leverage various distributed training strategies [32, 40, 41, 51] to alleviate computation and memory bottlenecks. For example, Megatron-LM [41, 51] provides optimized transformer blocks [61] with advanced parallelization strategies like tensor parallelism (TP) and pipeline parallelism (PP). However, as omni-modal architectures become complex in both functionality and parameter size, directly applying these techniques to omni-modal LLMs often leads to load imbalance and poor scalability [17], due to the current frameworks tightly couple model definition with parallel logic. This entanglement hiders generalization to more diverse model architectures. Although recent efforts [17, 24, 77] attempt to mitigate the inefficiencies of omni-modal training, they still suffer from the coupling of communication and computation, resulting in significant engineering cost and limited extensibility.

To efficiently train large omni-modal LLMs, VeOmni introduces model-centric distributed recipes that decouple model definition from parallel logic. As shown in Figure 1, distributed strategies such as fully sharded data parallel (FSDP), hybrid sharded data parallel (HSDP) [36, 78], sequence parallelism (SP) [29], and expert parallelism (EP) [74] can be applied to model blocks via a high-level *parallel plan* API. This design effectively separates communication from computation and has been validated in recent LLM training systems [36]. VeOmni supports flexible composition of parallel strategies (*e.g.*, FSDP+SP for 2D and FSDP+SP+EP for 3D parallelism), enabling a range of training recipes tailored to dense or MoE models. By decoupling parallel logic, new modality-specific modules can be integrated with minimal engineering effort, as computation modules need not account for distributed concerns. In addition, VeOmni provides a lightweight interface for customizing omni-modal LLMs, allowing users to easily add or remove modality-specific encoders and decoders.

Our contributions are as follows:

(1) We propose model-centric distributed recipes that efficiently scale omni-modal training using n-D parallelism with minimal engineering overhead.

(2) We introduce a light-weight configuration interface for easy customization of omni-modal LLMs.

(3) We demonstrate VeOmni's competitive efficiency and scalability across 8–128 GPUs on models ranging from 7B to 72B parameters under omni-modal scenarios.

2 Related Work

2.1 Multi-Modal and Omni-Modal LLMs

Recent advances in large language models (LLMs) have increasingly focused on developing multi-modal and omni-modal capabilities. The prevailing approaches include incorporating modality-specific encoders into the input space of the LLMs for multimodal understanding [19, 37, 39, 65, 72], and attaching generative decoders to the output space for multimodal generation [27, 59] or embodied action prediction [2, 31]. More recently, omni-modal LLMs that support unified understanding and generation across modalities have emerged, aiming to align arbitrary modality features with language in a shared latent space. Based on a unified paradigm of auto-regressive modeling over multimodal embeddings, these models differ significantly in how the modality features are encoded and decoded. Some [56, 67] adopt discrete-token generation pipelines based on VQ-VAE series [16, 46]. Some [25, 54, 64] employ continuous-token generation via latent diffusion models [44, 48]. Others [12, 71, 80] explore hybrid architectures that combine diffusion-based generation with auto-regressive decoding. In addition, some works proposed alternative next-token prediction schemes, such as masked generation [34], next patch prediction [42], next scale prediction [58], next block prediction [47], offering increased flexibility beyond standard auto-regressive decoding. Given this diverse landscape of model architectures, VeOmni offers a flexible and extensible framework for building and scaling new modeling paradigms.

2.2 LLM Training Frameworks

The landscape of large language model training frameworks has evolved significantly to address the computational demands of scaling. For pure text training, several mature frameworks have established themselves as industry standards. Megatron-LM [41, 51] pioneered optimized transformer blocks with advanced parallelization strategies including tensor parallelism (TP) and pipeline parallelism (PP), becoming the foundation for many subsequent frameworks. Colossal-AI [33] extends this paradigm by offering comprehensive 3D parallel training strategies that combine data, tensor, and pipeline parallelism for enhanced scalability. NeMo [50] provides an end-to-end cloud solution specifically designed for training large-scale LLMs with enterprise-grade features, while TorchTitan [36] and veScale [62] represent the newer generation of PyTorch-native frameworks that emphasize auto-parallelization and simplified programming models.

In the multi-modal domain, specialized frameworks have emerged to handle the unique challenges of any-to-text training. DistMM [24] introduces optimizations specifically tailored for multimodal LLM training, while DistTrain [77] addresses model and data heterogeneity through disaggregated training approaches. Align Anything [30] focuses on cross-modality model training with feedback mechanisms, and Optimus [17] accelerates large-scale multi-modal LLM training through bubble exploitation techniques. However, these existing frameworks primarily target either text-to-text or any-to-text scenarios, leaving a significant gap in supporting comprehensive omni-modal training (any-to-any) scenarios. Our framework addresses this limitation by providing a scalable and modular solution specifically designed for the complexities of omni-modal training, enabling seamless integration of diverse modalities within a unified training pipeline.

3 VeOmni: Scalable Omni-Modal Training

VeOmni designs a model-centric framework that is natively suitable for training omni-modal models. This framework includes a diverse range of distributed training strategies, enabling any modality to be easily scaled up for large-scale clusters.

3.1 Light-Weight Omni-Modal Customization

To support training across arbitrary modalities, we propose a plug-and-play architecture that allows any combination of multimodal encoders and decoders to be flexibly attached to the input and output sides of a foundation model, as shown in Fig. 2. This modular and fully decoupled design decouples system-level operations from model-specific computation, allowing the streaming pipeline to operate independently of the model pipeline. Therefore, VeOmni enables easy extension and scaling of diverse omni-modal LLMs.

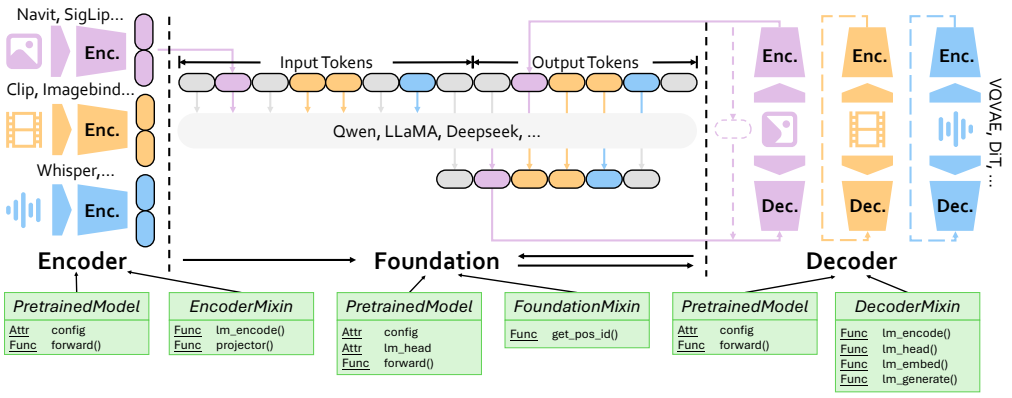

Figure 2 Composite Architecture for Omni-Modal LLMs. The architecture consists of three fully decoupled modules: Encoder, Foundation, and Decoder.

Our core design centers around a lightweight protocol tailored for each component. Specifically, encoder and decoder modules implement a unified interface by extending the `PreTrainedModel` class from HuggingFace along with a task-specific mixin. This design ensures compatibility with standard training workflows and enables all modules to be registered, initialized, and executed consistently.

During training, for input modalities, each encoder implements an `lm_encode` function that encodes raw modality data into token embeddings, which are then inserted into the input embeddings of the foundation model. Similarly, decoder modules implement the `lm_encode` function to provide training-time inputs representing the target output tokens. The final outputs from the foundation model are decoded into the target modality using `lm_head` function, completing the end-to-end mapping.

In the inference phase, for each prediction step, the output hidden states is passed through the corresponding modality decoder's `lm_embed` function, which generates embeddings that are used as inputs for the foundation model's next token prediction. Once all tokens have been predicted, the decoder's `lm_generate` function is invoked to generate the final modality-specific output.

The details of the training and inference processes are listed in Appendix B.3.

3.2 Model-Centric Distributed Recipe Zoo

To support large-scale distributed training of omni-modal LLMs, VeOmni offers a modular suite of distributed training strategies, including Fully Sharded Data Parallelism (FSDP), Sequence Parallelism (SP), and Expert Parallelism (EP), designed to handle challenges such as training with ultra-long sequences and efficiently scaling large mixture-of-experts (MoE) models. VeOmni enables users to freely compose distributed training recipes tailored to their model architectures and modality-specific requirements, without needing to manage low-level implementation details. The framework is designed with the following principles:

1. **Non-intrusive distributed training APIs.** VeOmni decouples the model architecture from the underlying distributed training mechanisms, enabling users to compose multimodal models flexibly without modifying low-level parallelization code.
2. **Support for long context training.** VeOmni accommodates training workloads involving extremely long sequences across different modalities, which is a common challenge in omni-modal training.
3. **Efficient scaling for MoE models.** VeOmni facilitates efficient training of large-scale MoE models through integrated expert parallelism, achieving both scalability and compute efficiency.

In the following sections, we introduce each of these distributed technologies in VeOmni and demonstrate how they are integrated to support scalable omni-modal training.

3.2.1 Fully and Hybrid Sharded Data Parallel

Fully Sharded Data Parallel (FSDP) is a highly efficient implementation of the Zero Redundancy Optimizer (ZeRO-3) [45] in PyTorch. Its primary advantage is the significant reduction in memory required on each GPU during training. By sharding a model's parameters, gradients, and optimizer states across all available devices, FSDP allows for the training of extremely large models that would otherwise exceed the memory capacity of a single GPU. A key advantage of FSDP is its non-intrusive design, which decouples the model's architecture from the parallelization strategy. This means developers can focus on designing their models without needing to modify the underlying code to accommodate the distributed training setup. This non-intrusive nature makes FSDP exceptionally well-suited for training omni-modal LLMs. Since these models are defined by their complex and heterogeneous architectures, FSDP's ability to parallelize training without requiring any changes to the model's code makes it an ideal solution. VeOmni integrates both FSDP1 [78] and FSDP2 [36] as fundamental components of its distributed training stack, and provides a unified API for easy configuration. More details in Appendix B.5.

To further improve training efficiency, VeOmni integrates Hybrid Sharded Data Parallel (HSDP), an extension of FSDP designed to minimize communication overhead. HSDP utilizes a 2D device mesh, combining FSDP within "shard groups" and Distributed Data Parallel (DDP) across "replicate groups". This hybrid approach drastically cuts down on inter-node communication, enabling even greater scalability. The switch to the more efficient HSDP is as simple as changing one parameter in the configuration, with no modifications to the underlying code required.

3.2.2 Sequence Parallelism for Long Context Training

With the advancement of state-of-the-art omni-modal LLMs, the sequence lengths required for training have grown significantly, particularly in domains such as high-resolution image or video understanding and generation [18, 65, 76]. This trend toward longer sequences poses substantial challenges in terms of both computational cost and memory consumption. Efficiently addressing the demands of long context training is therefore critical for scaling model capacity and unleashing the full potential of omni-modal architectures. To address this, VeOmni adopts DeepSpeed Ulysses [29], a highly efficient sequence parallelism technique specifically designed for transformer training on ultra-long sequences. DeepSpeed Ulysses splits activations along the sequence dimension and strategically orchestrates all-to-all communications during attention computation, ensuring that communication volume remains constant when both sequence length and device count are scaled proportionally. This design enables high efficiency and scalability for long-context model training. A core advantage of VeOmni is its provision of a non-intrusive interface for deploying Ulysses. Developers can seamlessly leverage DeepSpeed Ulysses without needing to modify model-building code or directly interact with the underlying implementation details. In particular, VeOmni enhances the implementation of Flash Attention [9] by integrating DeepSpeed Ulysses while keeping fully compatible with the default attention, enabling the straightforward adoption of advanced attention acceleration methods within sequence parallel workflows. More details in Appendix B.6.

Furthermore, to optimize training throughput, we introduce Async-Ulysses, an enhanced implementation designed to overlap communication and computation. This version schedules the all-to-all communication operations to execute concurrently with the linear projection computations. By effectively hiding a significant portion of the communication latency behind computation, this approach yields substantial improvements in training performance.

3.2.3 Expert Parallelism for MoE Model Scaling

Mixture-of-Experts (MoE) architectures have emerged as a mainstream solution for scaling large models efficiently [10], particularly due to their ability to sparsely activate subsets of parameters during inference and training, enabling significant improvements in both computational efficiency and model capacity. In the context of omni-modal foundation models, adopting MoE architectures as the backbone not only reduces training cost but also enhances model performance across diverse modalities by dynamically allocating expert capacity based on input content.

To support the scalable training of MoE-based omni-modal LLMs, VeOmni introduces a user-friendly and flexible interface for expert parallelism, allowing users to easily apply expert sharding across devices without manual configuration. This interface is compatible with widely used MoE variants and supports plug-and-play integration, thereby significantly lowering the barrier to implementing distributed expert parallelism. More details in Appendix B.7.

A major bottleneck in large-scale MoE training lies in all-to-all communication required for routing tokens to their assigned experts, introducing substantial latency. To mitigate this, VeOmni incorporates fine-grained communication-computation overlapping techniques inspired by recent advances [4, 74]. These works hide communication latency by scheduling collective operations concurrently with local expert computation, eliminating the need for complex pipeline-level solutions such as DualPipe [10]. Unlike pipeline-centric designs, which are often rigid and sensitive to modality-specific imbalance, VeOmni's operator-level approach is lightweight, model-agnostic, and particularly well-suited to multi-modal settings. This results in higher utilization and faster iteration during large-scale MoE training.

3.2.4 n-D Parallelism Training Strategies

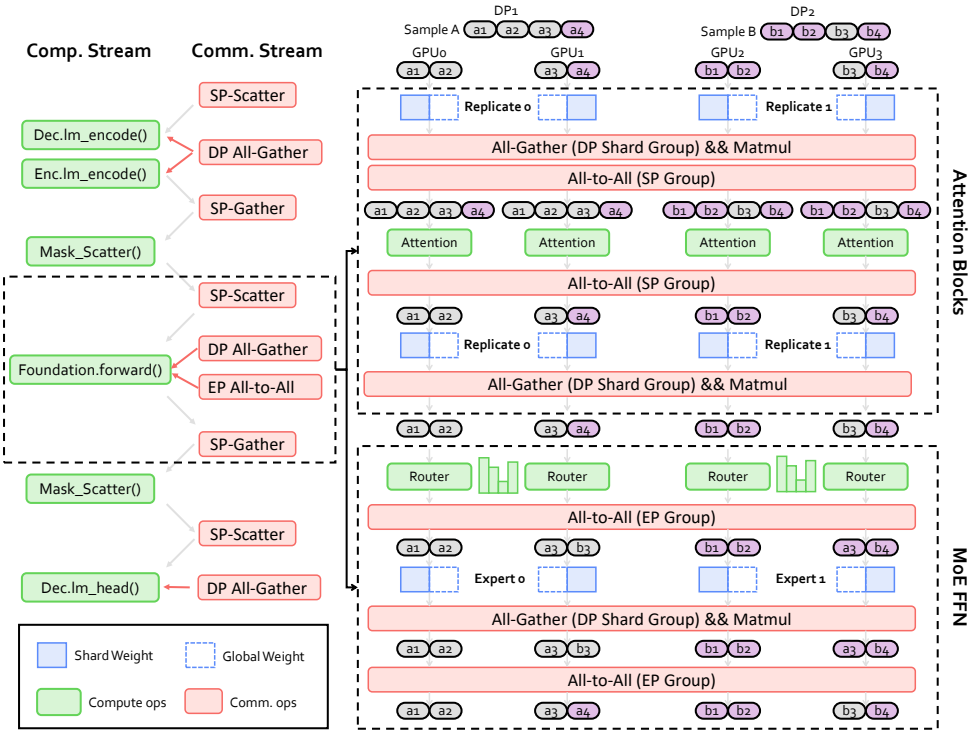

Figure 3 Overview of VeOmni's n-D Parallelism Design. The figure on the left illustrates the data flow of VeOmni during the training of omni-modal LLMs. The encoder for each modality processes its respective input, and then scatters the features to the corresponding ranks through an all-to-all communication operation. The right figure shows the 3D parallelism system of VeOmni. Attention blocks apply HSDP with data sharded size 2 and data replicate size 2, input apply Ulysses with sequence parallel size 2, Mixture-of-Experts blocks apply expert parallel and FSDP with experts parallel size 2 and data sharded size 2.

In VeOmni, core parallelism strategies (*i.e.*, Fully Sharded Data Parallel (FSDP), Sequence Parallelism (SP), and Expert Parallelism (EP)) are designed to be fully composable and can be flexibly applied to different components of an omni-modal LLM. This composability is particularly advantageous for multimodal training, where different modalities or architectural components may benefit from distinct parallelism techniques. For example, vision encoders can adopt FSDP or standard data parallelism depending on their scale, while language backbones can leverage a combination of EP for MoE layers and SP to support long-context processing. This fine-grained flexibility enables efficient and scalable training across diverse model architectures.

Figure 3 provides an overview of VeOmni's n-D parallelism design. The left side of the figure illustrates the communication and computation flow for a unified multimodal model, where different modality-specific encoders process their respective inputs and distribute intermediate features to downstream modules. The right side visualizes the application of hybrid parallelism strategies across different parts of the model, demonstrating how various parallelism techniques coexist within a single training configuration. To support this flexible parallelism composition, VeOmni introduces a unified abstraction for distributed training based on a global device mesh. Unlike approaches [41, 51] that require manually managing multiple process groups, VeOmni significantly simplifies the configuration and coordination of n-D parallelism(Appendix B.4). This not only reduces the complexity of process group management but also improves extensibility, making it easier to adapt and extend to future parallelism strategies or new model components.

3.2.5 Other System Optimization Strategies

In addition to the aforementioned parallelism strategies, VeOmni also incorporates a wide range of other system-level optimizations, following the key design principle of VeOmni, which is the decoupling of these optimization implementations from the model's core logic. This modular architecture allows for seamless integration, enabling these system-level enhancements to be readily applied across various models with minimal to no modification of the model code. These optimizations include, but are not limited to, the following:

- **Dynamic Batching.** Padding all samples in a batch to a fixed sequence length often leads to substantial computational inefficiency, particularly when there is significant variation in sequence lengths across samples. To mitigate this issue and improve training efficiency, VeOmni employs a dynamic batching mechanism that accumulates samples in a buffer and strategically packs them to approximate a target sequence length. Leveraging FlashAttention [8], this packing strategy enables efficient utilization of the batch budget with minimal padding overhead, while preserving the correctness of attention computation across samples.
- **Efficient Kernels.** To maximize training throughput, VeOmni incorporates a suite of highly optimized operator kernels, including RMSNorm, LayerNorm, RoPE, SwiGLU, and in-place CrossEntropy from Liger-kernel [23], along with FlashAttention [8, 9, 49] and MoE-specific operations [4, 74]. These kernels are carefully engineered for both high performance and broad compatibility, enabling efficient execution across diverse transformer-based architectures and model variants.
- **Memory Optimization.** VeOmni incorporates layer-wise recomputation [6], activation offloading, and optimizer state offloading to substantially reduce memory consumption during training. These memory-saving techniques enable the use of larger micro-batch sizes per GPU, which in turn improves the amortization of communication costs and facilitates better overlap with the communication overhead introduced by Fully Sharded Data Parallel (FSDP), ultimately enhancing overall training efficiency.
- **Efficient Distributed Checkpointing.** VeOmni leverages ByteCheckpoint [63] to enable efficient checkpoint saving and resumption across varying distributed configurations with minimal overhead. Beyond facilitating elastic training and enhancing cluster utilization, our framework further extends ByteCheckpoint to support multimodal models. This extension ensures consistent and reliable saving and loading of heterogeneous model components.
- **Meta Device Initialization.** VeOmni supports model initialization on the meta device for large models without allocating physical memory during the definition phase. After the model structure is instantiated on the meta device, we perform parameter sharding and parallel loading by converting parameters into the DTensor format, significantly accelerating the initialization and loading processes for large-scale models.

4 Experiments

In this section, we present experimental results highlighting the performance and scalability of VeOmni.

4.1 Experimental Setup

Environments. We evaluate the training performance and scalability of VeOmni on large-scale productive GPU clusters across configurations ranging from 8 to 128 GPUs, enabling a comprehensive study of VeOmni's behavior under both moderate and large-scale distributed settings.

Models and Datasets. We evaluate a diverse set of model architectures, including dense models such as Qwen2-VL 7B and 72B [65], and a mixture-of-experts (MoE) omni-modal LLM based on Qwen3-MoE [57]. To assess the multimodal capabilities of VeOmni, we use the following domain-specific datasets: FineWeb-100T [43] for text understanding, ShareGPT4V [5] for image understanding, LLaVA-Video [76] for video understanding, Voice Assistant [69] for audio understanding, and ImageNet [13] for image generation tasks. Each model is adapted using its native instruction template and augmented with special tokens to delineate modality boundaries (*e.g.* `<image_start>` and `<image_end>`).

Workloads and Metrics. To assess scalability across different model sizes and GPU counts, we progressively scale the input context length from 8K to 256K tokens. We evaluate the performance and scalability of VeOmni by measuring training throughput (tokens per second) and Memory Footprint Utilization (MFU) [7], which together provide a comprehensive view of system efficiency under varying workloads and parallel strategies. Additionally, we analyze loss convergence behavior across three structurally distinct omni-modal LLMs to validate training stability and overall effectiveness. For all experiments, we freeze the modality-specific encoders and decoders, while fully fine-tuning the remaining components, including the foundation model and multimodal projectors.

4.2 Training Recipes under Different Scenarios

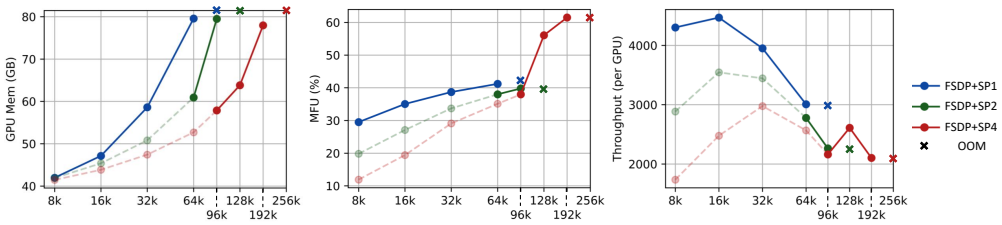

Figure 4 2D Parallelism (FSDP+SP) on Qwen2-VL 7B with 8 GPUs. The maximum sequence length varies from 8K to 256K tokens, with supported sequence parallel sizes ranging from 1 to 4.

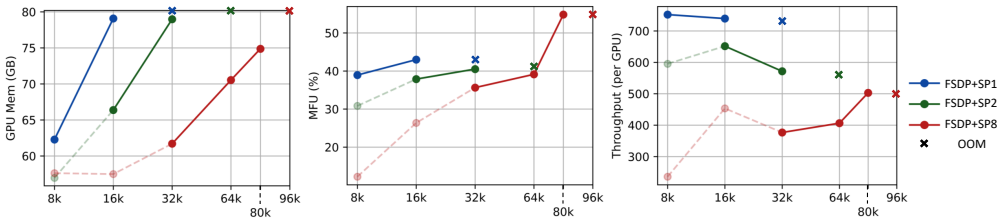

Figure 5 2D Parallelism (FSDP+SP) on Qwen2-VL 72B with 128 GPUs. The maximum sequence length varies from 8K to 96K tokens, with supported sequence parallel sizes ranging from 1 to 8.

Figures 4–6 (a) present a comprehensive comparison of the efficiency of various parallelism strategies—fully sharded data parallel (FSDP), sequence parallelism (SP), and expert parallelism (EP)—across diverse model scales and hardware configurations. Figures 4-5 show results of training Qwen2-VL on 8 and 128 GPUs, respectively. The findings reveal that increasing the degree of sequence parallelism enables the models to handle significantly longer context lengths. In particular, VeOmni can support up to 192K tokens for training the 7B model with an MFU of 61.5% and 96K tokens for training the 72B model with an MFU of 54.82%. Figure 6 further extends this analysis to a 3D parallelism configuration (FSDP + SP + EP) using a 30B parameter a mixture-of-experts (MoE) omni-modal LLM based on Qwen3-MoE, on 128 GPUs. In this setting,

combinations involving moderate levels of SP and EP achieve a balanced trade-off, supporting extended sequence lengths of up to 160K tokens, while maintaining competitive throughput.

The experimental results underscore the efficiency of VeOmni in handling long-sequence training and scaling mixture-of-experts (MoE) models. Specifically, VeOmni demonstrates strong performance by minimizing overhead in short-context scenarios, while effectively leveraging sequence and expert parallelism to maintain scalability and feasibility in extended-context and MoE settings.

4.3 Convergence Study on Omni-Modal LLMs

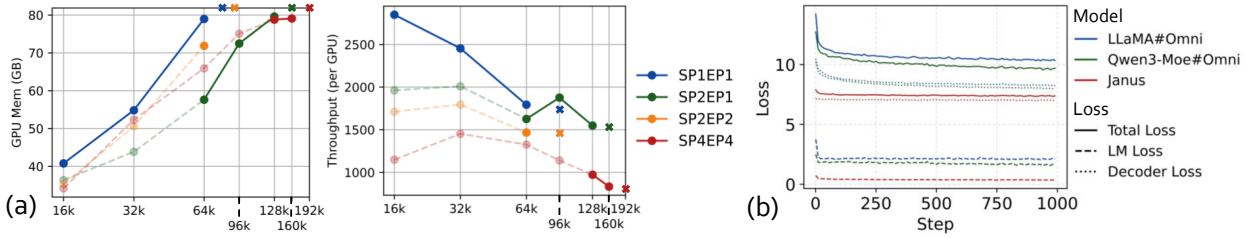

Figure 6 (a) 3D Parallelism (FSDP+SP+EP) on a 30B Qwen3MoE-based omni-modal LLM with 128 GPUs. The maximum sequence length varies from 16K to 192K tokens, with supported sequence parallel sizes ranging from 1 to 4 and expert parallel sizes ranging from 1 to 4. (b) Convergence Study on three Distinct omni-modal LLMs.

Figure 6 (b) presents the convergence behavior of three omni-modal LLMs on multimodal understanding and generation tasks. The understanding tasks span four modalities, *i.e.*, text, image, video, and audio, while the generation tasks involve text and image synthesis. Janus [67] is trained solely on image understanding and image generation. LLaMA#Omni and Qwen3-Moe#Omni are two custom models that share similar architectures: Qwen2.5-Omni [70] ViT serves as the image and video encoder, Qwen2.5-Omni Whisper as the audio encoder, and MoVQGAN [79] as the image decoder. LLaMA#Omni uses LLaMA [60] as the foundation, while Qwen3-Moe#Omni adopts Qwen3-MoE [57]. Janus adopts SigLip [73] as the image encoder, LLaMA as the foundation model, and LlamaGen [53] as the image decoder. "LM loss" refers to the cross-entropy loss over text tokens, and "decoder loss" refers to the cross-entropy loss over image tokens. As shown in Figure 6 (b), all models exhibit stable convergence across both understanding and generation tasks, demonstrating that VeOmni enables efficient and robust training for large omni-modal LLMs.

5 Conclusion

In this work, we introduce VeOmni, a model-centric distributed training framework designed to scale any-modality models efficiently. By integrating multiple parallelism methods and system optimizations into a composable recipe-based design, VeOmni enables seamless and efficient distributed training strategies across omni-modal LLMs. We further demonstrate the system-level optimizations, modular APIs, and real-world training applications on large-scale vision-language-audio models. Experimental analysis shows that VeOmni not only achieves high throughput and scalability but also provides developer-friendly abstractions for fast prototyping and production-scale deployment. In future work, we plan to extend VeOmni to support non-intrusive pipeline parallelism, enabling further decoupling of model definition and parallel execution. Additionally, we aim to enhance sequence parallelism with modality-aware data balancing strategies to better support multimodal training scenarios.

Acknowledgments

We thank Yawei Wen, Junda Feng, Junda Zhang, Haotian Zhou, Huiyao Shu, Hongyu Zhu, Changyue Liao, Yuanhang Gao, Weidong Chen, Shu Yun, Jie Wang, Yifan Pi, Ziyi Wang, Baixi Sun as well as other colleagues at ByteDance for their contribution for the VeOmni project.

References

[1] Jean-Baptiste Alayrac, Jeff Donahue, Pauline Luc, Antoine Miech, Iain Barr, Yana Hasson, Karel Lenc, Arthur Mensch, Katherine Millican, Malcolm Reynolds, et al. Flamingo: a visual language model for few-shot learning. Advances in neural information processing systems, 35:23716–23736, 2022.

[2] Kevin Black, Noah Brown, Danny Driess, Adnan Esmail, Michael Equi, Chelsea Finn, Niccolo Fusai, Lachy Groom, Karol Hausman, Brian Ichter, Szymon Jakubczak, Tim Jones, Liyiming Ke, Sergey Levine, Adrian Li-Bell, Mohith Mothukuri, Suraj Nair, Karl Pertsch, Lucy Xiaoyang Shi, James Tanner, Quan Vuong, Anna Walling, Haohuan Wang, and Ury Zhilinsky. π_0: A vision-language-action flow model for general robot control, 2024. URL https://arxiv.org/abs/2410.24164.

[3] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. Advances in neural information processing systems, 33:1877–1901, 2020.

[4] Li-Wen Chang, Wenlei Bao, Qi Hou, Chengquan Jiang, Ningxin Zheng, Yinmin Zhong, Xuanrun Zhang, Zuquan Song, Chengji Yao, Ziheng Jiang, Haibin Lin, Xin Jin, and Xin Liu. Flux: Fast software-based communication overlap on gpus through kernel fusion, 2024. URL https://arxiv.org/abs/2406.06858.

[5] Lin Chen, Jinsong Li, Xiaoyi Dong, Pan Zhang, Conghui He, Jiaqi Wang, Feng Zhao, and Dahua Lin. Sharegpt4v: Improving large multi-modal models with better captions. In European Conference on Computer Vision, pages 370–387. Springer, 2024.

[6] Tianqi Chen, Bing Xu, Chiyuan Zhang, and Carlos Guestrin. Training deep nets with sublinear memory cost. arXiv preprint arXiv:1604.06174, 2016.

[7] Aakanksha Chowdhery, Sharan Narang, Jacob Devlin, Maarten Bosma, Gaurav Mishra, Adam Roberts, Paul Barham, Hyung Won Chung, Charles Sutton, Sebastian Gehrmann, et al. Palm: Scaling language modeling with pathways. Journal of Machine Learning Research, 24(240):1–113, 2023.

[8] Tri Dao. Flashattention-2: Faster attention with better parallelism and work partitioning. arXiv preprint arXiv:2307.08691, 2023.

[9] Tri Dao, Dan Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. FlashAttention: Fast and memory-efficient exact attention with io-awareness. Advances in Neural Information Processing Systems, 35:16344–16359, 2022. URL https://proceedings.neurips.cc/paper_files/paper/2022/hash/67d57c32e20fd0a7a302cb81d36e40d5-Abstract-Conference.html.

[10] DeepSeek-AI. Deepseek-v3 technical report, 2025. URL https://arxiv.org/abs/2412.19437.

[11] Chaorui Deng, Deyao Zhu, Kunchang Li, Chenhui Gou, Feng Li, Zeyu Wang, Shu Zhong, Weihao Yu, Xiaonan Nie, Ziang Song, Guang Shi, and Haoqi Fan. Emerging properties in unified multimodal pretraining. arXiv preprint arXiv:2505.14683, 2025.

[12] Chaorui Deng, Deyao Zhu, Kunchang Li, Chenhui Gou, Feng Li, Zeyu Wang, Shu Zhong, Weihao Yu, Xiaonan Nie, Ziang Song, et al. Emerging properties in unified multimodal pretraining. arXiv preprint arXiv:2505.14683, 2025.

[13] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In 2009 IEEE conference on computer vision and pattern recognition, pages 248–255. Ieee, 2009.

[14] Linhao Dong, Shuang Xu, and Bo Xu. Speech-transformer: a no-recurrence sequence-to-sequence model for speech recognition. In 2018 IEEE international conference on acoustics, speech and signal processing (ICASSP), pages 5884–5888. IEEE, 2018.

[15] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al. An image is worth 16x16 words: Transformers for image recognition at scale. In International Conference on Learning Representations, 2021.

[16] Patrick Esser, Robin Rombach, and Björn Ommer. Taming transformers for high-resolution image synthesis, 2020.

[17] Weiqi Feng, Yangrui Chen, Shaoyu Wang, Yanghua Peng, Haibin Lin, and Minlan Yu. Optimus: Accelerating {Large-Scale}{Multi-Modal}{LLM} training by bubble exploitation. In 2025 USENIX Annual Technical Conference (USENIX ATC 25), pages 161–177, 2025.

[18] Yu Gao, Haoyuan Guo, Tuyen Hoang, Weilin Huang, Lu Jiang, Fangyuan Kong, Huixia Li, Jiashi Li, Liang Li, Xiaojie Li, et al. Seedance 1.0: Exploring the boundaries of video generation models. arXiv preprint arXiv:2506.09113, 2025.

[19] Rohit Girdhar, Alaaeldin El-Nouby, Zhuang Liu, Mannat Singh, Kalyan Vasudev Alwala, Armand Joulin, and Ishan Misra. Imagebind: One embedding space to bind them all. In Proceedings of the IEEE/CVF conference on computer vision and pattern recognition, pages 15180–15190, 2023.

[20] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial networks. Communications of the ACM, 63(11):139–144, 2020.

[21] Jiaxian Guo, Junnan Li, Dongxu Li, Anthony Meng Huat Tiong, Boyang Li, Dacheng Tao, and Steven Hoi. From images to textual prompts: Zero-shot visual question answering with frozen large language models. In Proceedings of the IEEE/CVF conference on computer vision and pattern recognition, pages 10867–10877, 2023.

[22] Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models. Advances in neural information processing systems, 33:6840–6851, 2020.

[23] Pin-Lun Hsu, Yun Dai, Vignesh Kothapalli, Qingquan Song, Shao Tang, Siyu Zhu, Steven Shimizu, Shivam Sahni, Haowen Ning, and Yanning Chen. Liger kernel: Efficient triton kernels for llm training, 2025. URL https://arxiv.org/abs/2410.10989.

[24] Jun Huang, Zhen Zhang, Shuai Zheng, Feng Qin, and Yida Wang. Distmm: accelerating distributed multimodal model training. In Proceedings of the 21st USENIX Symposium on Networked Systems Design and Implementation, pages 1157–1171, 2024.

[25] Runhui Huang, Chunwei Wang, Junwei Yang, Guansong Lu, Yunlong Yuan, Jianhua Han, Lu Hou, Wei Zhang, Lanqing Hong, Hengshuang Zhao, et al. Illume+: Illuminating unified mllm with dual visual tokenization and diffusion refinement. arXiv preprint arXiv:2504.01934, 2025.

[26] Wenxuan Huang, Bohan Jia, Zijie Zhai, Shaosheng Cao, Zheyu Ye, Fei Zhao, Zhe Xu, Yao Hu, and Shaohui Lin. Vision-r1: Incentivizing reasoning capability in multimodal large language models. arXiv preprint arXiv:2503.06749, 2025.

[27] Yuzhou Huang, Liangbin Xie, Xintao Wang, Ziyang Yuan, Xiaodong Cun, Yixiao Ge, Jiantao Zhou, Chao Dong, Rui Huang, Ruimao Zhang, et al. Smartedit: Exploring complex instruction-based image editing with multimodal large language models. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pages 8362–8371, 2024.

[28] Aaron Hurst, Adam Lerer, Adam P Goucher, Adam Perelman, Aditya Ramesh, Aidan Clark, AJ Ostrow, Akila Welihinda, Alan Hayes, Alec Radford, et al. Gpt-4o system card. arXiv preprint arXiv:2410.21276, 2024.

[29] Sam Ade Jacobs, Masahiro Tanaka, Chengming Zhang, Minjia Zhang, Shuaiwen Leon Song, Samyam Rajbhandari, and Yuxiong He. Deepspeed ulysses: System optimizations for enabling training of extreme long sequence transformer models. arXiv preprint arXiv:2309.14509, 2023.

[30] Jiaming Ji, Jiayi Zhou, Hantao Lou, Boyuan Chen, Donghai Hong, Xuyao Wang, Wenqi Chen, Kaile Wang, Rui Pan, Jiahao Li, et al. Align anything: Training all-modality models to follow instructions with language feedback. arXiv preprint arXiv:2412.15838, 2024.

[31] Moo Jin Kim, Karl Pertsch, Siddharth Karamcheti, Ted Xiao, Ashwin Balakrishna, Suraj Nair, Rafael Rafailov, Ethan Foster, Grace Lam, Pannag Sanketi, Quan Vuong, Thomas Kollar, Benjamin Burchfiel, Russ Tedrake, Dorsa Sadigh, Sergey Levine, Percy Liang, and Chelsea Finn. Openvla: An open-source vision-language-action model, 2024. URL https://arxiv.org/abs/2406.09246.

[32] Shen Li, Yanli Zhao, Rohan Varma, Omkar Salpekar, Pieter Noordhuis, Teng Li, Adam Paszke, Jeff Smith, Brian Vaughan, Pritam Damania, et al. Pytorch distributed: experiences on accelerating data parallel training. Proceedings of the VLDB Endowment, 13(12):3005–3018, 2020.

[33] Shenggui Li, Hongxin Liu, Zhengda Bian, Jiarui Fang, Haichen Huang, Yuliang Liu, Boxiang Wang, and Yang You. Colossal-ai: A unified deep learning system for large-scale parallel training. In Proceedings of the 52nd International Conference on Parallel Processing, pages 766–775, 2023.

[34] Tianhong Li, Yonglong Tian, He Li, Mingyang Deng, and Kaiming He. Autoregressive image generation without vector quantization. Advances in Neural Information Processing Systems, 37:56424–56445, 2024.

[35] Zongming Li, Tianheng Cheng, Shoufa Chen, Peize Sun, Haocheng Shen, Longjin Ran, Xiaoxin Chen, Wenyu Liu, and Xinggang Wang. Controlar: Controllable image generation with autoregressive models. In The Thirteenth International Conference on Learning Representations, 2025.

[36] Wanchao Liang, Tianyu Liu, Less Wright, Will Constable, Andrew Gu, Chien-Chin Huang, Iris Zhang, Wei Feng, Howard Huang, Junjie Wang, et al. Torchtitan: One-stop pytorch native solution for production ready llm pretraining. In The Thirteenth International Conference on Learning Representations, 2025.

[37] Bin Lin, Bin Zhu, Yang Ye, Munan Ning, Peng Jin, and Li Yuan. Video-llava: Learning united visual representation by alignment before projection. arXiv preprint arXiv:2311.10122, 2023.

[38] Hao Liu, Wilson Yan, Matei Zaharia, and Pieter Abbeel. World model on million-length video and language with blockwise ringattention. In The Thirteenth International Conference on Learning Representations, 2025.

[39] Haotian Liu, Chunyuan Li, Qingyang Wu, and Yong Jae Lee. Visual instruction tuning. In A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine, editors, Advances in Neural Information Processing Systems, volume 36, pages 34892–34916. Curran Associates, Inc., 2023. URL https://proceedings.neurips.cc/paper_files/paper/2023/file/6dcf277ea32ce3288914faf369fe6de0-Paper-Conference.pdf.

[40] Deepak Narayanan, Aaron Harlap, Amar Phanishayee, Vivek Seshadri, Nikhil R Devanur, Gregory R Ganger, Phillip B Gibbons, and Matei Zaharia. Pipedream: Generalized pipeline parallelism for dnn training. In Proceedings of the 27th ACM symposium on operating systems principles, pages 1–15, 2019.

[41] Deepak Narayanan, Mohammad Shoeybi, Jared Casper, Patrick LeGresley, Mostofa Patwary, Vijay Korthikanti, Dmitri Vainbrand, Prethvi Kashinkunti, Julie Bernauer, Bryan Catanzaro, et al. Efficient large-scale language model training on gpu clusters using megatron-lm. In Proceedings of the international conference for high performance computing, networking, storage and analysis, pages 1–15, 2021.

[42] Yatian Pang, Peng Jin, Shuo Yang, Bin Lin, Bin Zhu, Zhenyu Tang, Liuhan Chen, Francis EH Tay, Ser-Nam Lim, Harry Yang, et al. Next patch prediction for autoregressive visual generation. arXiv preprint arXiv:2412.15321, 2024.

[43] Guilherme Penedo, Hynek Kydlíček, Loubna Ben allal, Anton Lozhkov, Margaret Mitchell, Colin Raffel, Leandro Von Werra, and Thomas Wolf. The fineweb datasets: Decanting the web for the finest text data at scale. In The Thirty-eight Conference on Neural Information Processing Systems Datasets and Benchmarks Track, 2024. URL https://openreview.net/forum?id=n6SCkn2QaG.

[44] Dustin Podell, Zion English, Kyle Lacey, Andreas Blattmann, Tim Dockhorn, Jonas Müller, Joe Penna, and Robin Rombach. Sdxl: Improving latent diffusion models for high-resolution image synthesis. arXiv preprint arXiv:2307.01952, 2023.

[45] Samyam Rajbhandari, Jeff Rasley, Olatunji Ruwase, and Yuxiong He. Zero: Memory optimizations toward training trillion parameter models. In SC20: International Conference for High Performance Computing, Networking, Storage and Analysis, pages 1–16. IEEE, 2020.

[46] Ali Razavi, Aaron Van den Oord, and Oriol Vinyals. Generating diverse high-fidelity images with vq-vae-2. Advances in neural information processing systems, 32, 2019.

[47] Shuhuai Ren, Shuming Ma, Xu Sun, and Furu Wei. Next block prediction: Video generation via semi-autoregressive modeling. arXiv preprint arXiv:2502.07737, 2025.

[48] Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. High-resolution image synthesis with latent diffusion models. In Proceedings of the IEEE/CVF conference on computer vision and pattern recognition, pages 10684–10695, 2022.

[49] Jay Shah, Ganesh Bikshandi, Ying Zhang, Vijay Thakkar, Pradeep Ramani, and Tri Dao. Flashattention-3: Fast and accurate attention with asynchrony and low-precision. Advances in Neural Information Processing Systems, 37:68658–68685, 2024.

[50] Gerald Shen, Zhilin Wang, Olivier Delalleau, Jiaqi Zeng, Yi Dong, Daniel Egert, Shengyang Sun, Jimmy J Zhang, Sahil Jain, Ali Taghibakhshi, et al. Nemo-aligner: Scalable toolkit for efficient model alignment. In First Conference on Language Modeling, 2024.

[51] Mohammad Shoeybi, Mostofa Patwary, Raul Puri, Patrick LeGresley, Jared Casper, and Bryan Catanzaro. Megatron-lm: Training multi-billion parameter language models using model parallelism. arXiv preprint arXiv:1909.08053, 2019.

[52] Zhaochen Su, Peng Xia, Hangyu Guo, Zhenhua Liu, Yan Ma, Xiaoye Qu, Jiaqi Liu, Yanshu Li, Kaide Zeng, Zhengyuan Yang, et al. Thinking with images for multimodal reasoning: Foundations, methods, and future frontiers. arXiv preprint arXiv:2506.23918, 2025.

[53] Peize Sun, Yi Jiang, Shoufa Chen, Shilong Zhang, Bingyue Peng, Ping Luo, and Zehuan Yuan. Autoregressive model beats diffusion: Llama for scalable image generation. arXiv preprint arXiv:2406.06525, 2024.

[54] Quan Sun, Yufeng Cui, Xiaosong Zhang, Fan Zhang, Qiying Yu, Yueze Wang, Yongming Rao, Jingjing Liu, Tiejun Huang, and Xinlong Wang. Generative multimodal models are in-context learners. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pages 14398–14409, 2024.

[55] Quan Sun, Qiying Yu, Yufeng Cui, Fan Zhang, Xiaosong Zhang, Yueze Wang, Hongcheng Gao, Jingjing Liu, Tiejun Huang, and Xinlong Wang. Emu: Generative pretraining in multimodality. In The Twelfth International Conference on Learning Representations, 2024.

[56] Chameleon Team. Chameleon: Mixed-modal early-fusion foundation models. arXiv preprint arXiv:2405.09818, 2024. doi: 10.48550/arXiv.2405.09818. URL https://github.com/facebookresearch/chameleon.

[57] Qwen Team. Qwen3 technical report, 2025. URL https://arxiv.org/abs/2505.09388.

[58] Keyu Tian, Yi Jiang, Zehuan Yuan, Bingyue Peng, and Liwei Wang. Visual autoregressive modeling: Scalable image generation via next-scale prediction. Advances in neural information processing systems, 37:84839–84865, 2024.

[59] Xueyun Tian, Wei Li, Bingbing Xu, Yige Yuan, Yuanzhuo Wang, and Huawei Shen. Mige: A unified framework for multimodal instruction-based image generation and editing, 2025. URL https://arxiv.org/abs/2502.21291.

[60] Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, et al. Llama: Open and efficient foundation language models. arXiv preprint arXiv:2302.13971, 2023.

[61] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. Advances in neural information processing systems, 30, 2017.

[62] veScale Team. vescale: Towards pytorch-native auto-parallel framework. https://github.com/volcengine/veScale, 2024.

[63] Borui Wan, Mingji Han, Yiyao Sheng, Yanghua Peng, Haibin Lin, Mofan Zhang, Zhichao Lai, Menghan Yu, Junda Zhang, Zuquan Song, et al. Bytecheckpoint: A unified checkpointing system for large foundation model development. arXiv preprint arXiv:2407.20143, 2024.

[64] Chunwei Wang, Guansong Lu, Junwei Yang, Runhui Huang, Jianhua Han, Lu Hou, Wei Zhang, and Hang Xu. Illume: Illuminating your llms to see, draw, and self-enhance. arXiv preprint arXiv:2412.06673, 2024.

[65] Peng Wang, Shuai Bai, Sinan Tan, Shijie Wang, Zhihao Fan, Jinze Bai, Keqin Chen, Xuejing Liu, Jialin Wang, Wenbin Ge, Yang Fan, Kai Dang, Mengfei Du, Xuancheng Ren, Rui Men, Dayiheng Liu, Chang Zhou, Jingren Zhou, and Junyang Lin. Qwen2-vl: Enhancing vision-language model's perception of the world at any resolution. arXiv preprint arXiv:2409.12191, 2024.

[66] Yiqi Wang, Wentao Chen, Xiaotian Han, Xudong Lin, Haiteng Zhao, Yongfei Liu, Bohan Zhai, Jianbo Yuan, Quanzeng You, and Hongxia Yang. Exploring the reasoning abilities of multimodal large language models (mllms): A comprehensive survey on emerging trends in multimodal reasoning. arXiv preprint arXiv:2401.06805, 2024.

[67] Chengyue Wu, Xiaokang Chen, Zhiyu Wu, Yiyang Ma, Xingchao Liu, Zizheng Pan, Wen Liu, Zhenda Xie, Xingkai Yu, Chong Ruan, et al. Janus: Decoupling visual encoding for unified multimodal understanding and generation. In Proceedings of the Computer Vision and Pattern Recognition Conference, pages 12966–12977, 2025.

[68] Jinheng Xie, Weijia Mao, Zechen Bai, David Junhao Zhang, Weihao Wang, Kevin Qinghong Lin, Yuchao Gu, Zhijie Chen, Zhenheng Yang, and Mike Zheng Shou. Show-o: One single transformer to unify multimodal understanding and generation. In The Thirteenth International Conference on Learning Representations, 2025.

[69] Zhifei Xie and Changqiao Wu. Mini-omni: Language models can hear, talk while thinking in streaming. arXiv preprint arXiv:2408.16725, 2024.

[70] Jin Xu, Zhifang Guo, Jinzheng He, Hangrui Hu, Ting He, Shuai Bai, Keqin Chen, Jialin Wang, Yang Fan, Kai Dang, et al. Qwen2.5-omni technical report. arXiv preprint arXiv:2503.20215, 2025.

[71] Ling Yang, Ye Tian, Bowen Li, Xinchen Zhang, Ke Shen, Yunhai Tong, and Mengdi Wang. Mmada: Multimodal large diffusion language models. arXiv preprint arXiv:2505.15809, 2025.

[72] Zhenfei Yin, Jiong Wang, Jianjian Cao, Zhelun Shi, Dingning Liu, Mukai Li, Xiaoshui Huang, Zhiyong Wang, Lu Sheng, LEI BAI, Jing Shao, and Wanli Ouyang. Lamm: Language-assisted multi-modal instruction-tuning dataset, framework, and benchmark. In A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine, editors, Advances in Neural Information Processing Systems, volume 36, pages 26650–26685. Curran Associates, Inc., 2023. URL https://proceedings.neurips.cc/paper_files/paper/2023/file/548a41b9cac6f50dccf7e63e9e1b1b9b-Paper-Datasets_and_Benchmarks.pdf.

[73] Xiaohua Zhai, Basil Mustafa, Alexander Kolesnikov, and Lucas Beyer. Sigmoid loss for language image pre-training. In Proceedings of the IEEE/CVF international conference on computer vision, pages 11975–11986, 2023.

[74] Shulai Zhang, Ningxin Zheng, Haibin Lin, Ziheng Jiang, Wenlei Bao, Chengquan Jiang, Qi Hou, Weihao Cui, Size Zheng, Li-Wen Chang, Quan Chen, and Xin Liu. Comet: Fine-grained computation-communication overlapping for mixture-of-experts, 2025. URL https://arxiv.org/abs/2502.19811.

[75] Tianjun Zhang, Yi Zhang, Vibhav Vineet, Neel Joshi, and Xin Wang. Controllable text-to-image generation with gpt-4. arXiv preprint arXiv:2305.18583, 2023.

[76] Yuanhan Zhang, Jinming Wu, Wei Li, Bo Li, Zejun Ma, Ziwei Liu, and Chunyuan Li. Video instruction tuning with synthetic data, 2024. URL https://arxiv.org/abs/2410.02713.

[77] Zili Zhang, Yinmin Zhong, Ranchen Ming, Hanpeng Hu, Jianjian Sun, Zheng Ge, Yibo Zhu, and Xin Jin. Disttrain: Addressing model and data heterogeneity with disaggregated training for multimodal large language models. arXiv preprint arXiv:2408.04275, 2024.

[78] Yanli Zhao, Andrew Gu, Rohan Varma, Liang Luo, Chien-Chin Huang, Min Xu, Less Wright, Hamid Shojanazeri, Myle Ott, Sam Shleifer, et al. Pytorch fsdp: experiences on scaling fully sharded data parallel. arXiv preprint arXiv:2304.11277, 2023.

[79] Chuanxia Zheng, Tung-Long Vuong, Jianfei Cai, and Dinh Phung. Movq: Modulating quantized vectors for high-fidelity image generation. Advances in Neural Information Processing Systems, 35:23412–23425, 2022.

[80] Chunting Zhou, Lili Yu, Arun Babu, Kushal Tirumala, Michihiro Yasunaga, Leonid Shamis, Jacob Kahn, Xuezhe Ma, Luke Zettlemoyer, and Omer Levy. Transfusion: Predict the next token and diffuse images with one multi-modal model. arXiv preprint arXiv:2408.11039, 2024.

Appendix

A Technical Appendices and Supplementary Material

A.1 Controlled Benchmarking of VeOmni vs. TorchTitan on Large Language Models

In the main paper, we demonstrate that VeOmni supports efficient large-scale training for multimodal models, a capability not yet supported by other existing frameworks. As such, a direct comparison in multimodal training scenarios is infeasible. To provide a fair benchmark, we conduct controlled experiments on large-scale text-only models to compare VeOmni with *TorchTitan* [36], a state-of-the-art distributed training framework.

Table 1 Performance Comparison between VeOmni and TorchTitan on Qwen2-7B, 128GPUs. Both use mixed precision training and full activation checkpointing. micro batch size of 1, global batch size of 128.

Techniques	Sequence Length	Memory		Throughput		MFU	
		VeOmni	TorchTitan	VeOmni	TorchTitan	VeOmni	TorchTitan
FSDP	8k	21.91GB	23.26GB	**6940**	6639	38.76	34.61
FSDP+SP4	8k	23.23GB	22.55GB	**3830**	3590	20.66	18.72
FSDP	16k	24.45GB	39.58GB	**6583**	6082	43.22	37.62
FSDP+SP4	16k	23.92GB	25.08GB	**4889**	4056	31.92	25.24
FSDP	32k	31.41GB	71.81GB	**5315**	4878	44.42	39.68
FSDP+SP4	32k	25.27GB	30.13GB	**4629**	3727	38.69	30.32
FSDP	64k	43.00GB	OOM	**3725**	OOM	45.92	OOM
FSDP+SP4	64k	28.70GB	44.82GB	**3452**	2853	42.35	34.32
FSDP	128k	70.26GB	OOM	**2258**	OOM	44.95	OOM
FSDP+SP4	128k	35.49GB	OOM	**2187**	OOM	43.49	OOM

Table 2 Performance Comparison between VeOmni and TorchTitan on Qwen2.5-32B, 128GPUs. Both use mixed precision training and full activation checkpointing. Micro batch size of 1, global batch size of 128.

Techniques	Sequence Length	Memory		Throughput		MFU	
		VeOmni	TorchTitan	VeOmni	TorchTitan	VeOmni	TorchTitan
FSDP+SP4	8k	35.98GB	34.37GB	**1140**	1017	26.05	22.90
FSDP+SP8	8k	36.47GB	33.42GB	**668**	596	15.14	13.43
FSDP+SP4	16k	37.43GB	39.48GB	**1316**	1131	34.52	29.05
FSDP+SP8	16k	37.31GB	37.14GB	**1074**	789	27.87	20.28
FSDP+SP4	32k	42.42GB	46.78GB	**1240**	1022	36.11	32.75
FSDP+SP8	32k	38.76GB	44.04GB	**1115**	821	40.36	26.31
FSDP+SP4	64k	49.06GB	68.16GB	**957**	817	43.02	36.58
FSDP+SP8	64k	43.75GB	58.58GB	**903**	689	40.74	30.84
FSDP+SP4	128k	64.85GB	OOM	**632**	OOM	44.20	OOM
FSDP+SP8	128k	50.41GB	79.64GB	**613**	507	42.92	35.57

Table 3 Performance Comparison between VeOmni and TorchTitan on Qwen2-72B, 128GPUs. Both use mixed precision training and full activation checkpointing. Micro batch size of 1, global batch size of 128.

Techniques	Sequence Length	Memory		Throughput		MFU	
		VeOmni	TorchTitan	VeOmni	TorchTitan	VeOmni	TorchTitan
FSDP+SP4	8k	58.62GB	54.08GB	565	499	28.04	24.74
FSDP+SP8	8k	58.35GB	51.73GB	330	297	16.11	14.75
FSDP+SP4	16k	62.15GB	64.35GB	624	544	35.34	30.44
FSDP+SP8	16k	59.98GB	58.02GB	532	394	29.82	22.03
FSDP+SP4	32k	68.35GB	79.36GB	592	497	41.05	34.16
FSDP+SP8	32k	63.49GB	70.39GB	533	398	36.72	27.31
FSDP+SP4	64k	79.10GB	OOM	468	OOM	43.98	OOM
FSDP+SP8	64k	70.09GB	79.04GB	441	335	41.63	31.56
FSDP+SP4	128k	OOM	OOM	OOM	OOM	OOM	OOM
FSDP+SP8	128k	76.67GB	OOM	303	OOM	43.68	OOM

Table 4 Performance of VeOmni on Qwen3Moe-30B, 128GPUs. Use mixed precision training and full activation checkpointing. Micro batch size of 1, global batch size of 128. (**TorchTitan** not supported)

Techniques	Sequence Length	Memory	Throughput	MFU
FSDP+SP1+EP8	8k	45.58GB	2776	7.45
FSDP+SP4+EP8	8k	22.56GB	1216	3.02
FSDP+SP1+EP8	16k	69.99GB	2853	10.74
FSDP+SP4+EP8	16k	34.32GB	1897	6.55
FSDP+SP1+EP8	32k	77.25GB	2417	13.40
FSDP+SP4+EP8	32k	63.08GB	2064	11.35
FSDP+SP1+EP8	64k	OOM	OOM	OOM
FSDP+SP4+EP8	64k	77.92GB	1716	15.65
FSDP+SP1+EP8	128k	OOM	OOM	OOM
FSDP+SP4+EP8	128k	78.01GB	1075	17.92

These benchmark results demonstrate that VeOmni consistently achieves higher throughput and memory efficiency across a range of model sizes and sequence lengths. Notably, VeOmni supports large-sequence and MoE models that exceed the memory or capability limits of TorchTitan.

B API Design

B.1 OmniData Process and Collator API

Listing 1 demonstrates the data collation pipeline in VeOmni, which provides a unified interface for processing and batching multimodal inputs, including text, image, video, and audio. The `OmniDataCollatorWithPacking` class enables fine-grained control over feature packing and concatenation, making it straightforward to prepare batched training data for diverse modalities. This modular design simplifies the integration of new modalities and supports flexible token-level and feature-level alignment strategies. The resulting `train_dataloader` is constructed using `build_dataloader()`, which automatically aligns with the parallel training configuration.

Listing 1 VeOmni OmniDataCollator API

```
from omniscale.data import (
    OmniDataCollatorWithPacking,
    build_dataloader,
)
data_collate_fn.append(
    OmniDataCollatorWithPacking(
        packing_features=[
            "input_ids",
            "attention_mask",
            "labels",
            "position_ids",
            "image_input_mask",
            "image_output_mask",
            "video_input_mask",
            "video_output_mask",
            "audio_input_mask",
            "audio_output_mask",
        ],
        concat_features=[
            "image_input_features",
            "image_output_features",
            "video_input_features",
            "video_output_features",
            "audio_input_features",
            "audio_output_features",
        ],
    )
)

train_dataloader = build_dataloader(
    dataset=train_dataset,
    micro_batch_size=micro_batch_size,
    global_batch_size=global_batch_size,
    collate_fn=data_collate_fn,
)
```

B.2 Building Omni-Modal LLMs

Listing 2 shows the usage of VeOmni's `build_omni_model` API, which serves as the unified entry point for constructing multimodal OmniModels. Given the configuration file and model component paths (including modality-specific encoders, decoders, and the foundation model), this function automatically assembles and initializes the model on a designated device. By abstracting away the underlying initialization logic, VeOmni ensures that users can easily instantiate complex multimodal models in a modular and scalable fashion, while maintaining compatibility with distributed parallelization workflows.

Listing 2 `build_omni_model` API usage

```
from omniscale.models import build_omni_model

model: SeedOmniModel = build_omni_model(
    config_path=config_path,
    weights_path=model_path,
    encoders=encoders,
    decoders=decoders,
    foundation=foundation,
    init_device=init_device,
)
```

B.3 Forward Process of OmniModel in VeOmni

Listing 3-4 illustrates the partial forward process implementation of the plug-and-play architecture in VeOmni, named OmniModel, which adopts an encoder–foundation–decoder architecture. The snippet demonstrates how the forward logic across training and inference modes integrates each module—encoder, foundation model, and decoder—in a seamless and modular pipeline. During inference, the control flow dynamically switches to modality-specific generation branches based on the generated text special tokens, enabling flexible multimodal outputs. During inference, OmniModel supports parallelized generation and classifier-free guidance [67]. This architecture highlights the clear decoupling between encoder, foundation, and decoder modules, allowing each component to define and execute its own forward logic independently.

Listing 3 Training Process

```
class OmniEncoder:
    def forward(**inputs):
        inputs_embeds = self.text_encoder(inputs['input_ids'])
        image_input_embeds = self.image_encoder.lm_encode(
            inputs.get("image_input_features")
        )
        video_input_embeds = self.video_encoder.lm_encode(
            inputs.get("video_input_features")
        )
        audio_input_embeds = self.audio_encoder.lm_encode(
            inputs.get("audio_input_features")
        )
        inputs_embeds = self.masked_scatter(
            inputs_embeds,
            image_input_embeds,
            inputs.get("image_input_mask"),
            video_input_embeds,
            inputs.get("video_input_mask"),
            audio_input_embeds,
            inputs.get("audio_input_mask"),
        )
        return inputs_embeds

class OmniDecoder:
    def forward(inputs_embeds, **inputs):
        image_output_embeds = self.image_decoder.lm_encode(
            inputs.get("image_output_features")
            )
        video_output_embeds = self.video_encoder.lm_encode(
            inputs.get("video_output_features")
        )
        audio_output_embeds = self.audio_encoder.lm_encode(
```

```
            inputs.get("audio_output_features")
        )
        inputs_embeds = self.masked_scatter(
            inputs_embeds,
            image_output_embeds,
            inputs.get("image_output_mask"),
            video_output_embeds,
            inputs.get("video_output_mask"),
            audio_output_embeds,
            inputs.get("audio_output_mask"),
        )
        return inputs_embeds

    def lm_head(hidden_states, **inputs):
        loss = self.image_decoder.lm_head(outputs.hidden_states, **inputs)
        loss += self.video_decoder.lm_head(outputs.hidden_states, **inputs)
        loss += self.audio_decoder.lm_head(outputs.hidden_states, **inputs)
        return loss


class OmniModel:
    encoder: OmniEncoder
    decoder: OmniDecoder

    def forward(self, **inputs):
        inputs_embeds = self.encoder(**inputs)
        inputs_embeds = self.decoder(inputs_embeds, **inputs)
        outputs = self.foundation(inputs_embeds, **inputs)

        hidden_states = outputs.hidden_states
        loss = None

        if self.training:
            loss = outputs.loss
            loss += self.decoder.lm_head(hidden_states, **inputs)

        return OmniOutput(
            loss = loss,
            hidden_states = hidden_states
        )
```

Listing 4 Inference Process

```
class OmniModel():
    def setup_image_generation():
        self.generation_type = "image"
        self.setup_parallelize_generation()
        self.setup_classifier_free_guidance()
        self.setup_position_id_map()

    def setup_video_generation():
        ...

    def setup_audio_generation():
        ...

    def setup_text_generation():
        ...
```

```
    def prepare_inputs_for_generation(self, **inputs):
        model_inputs = self.foundation.prepare_inputs_for_generation(
            **inputs
        )
        if cache_position[0] == 0:
            inputs_embeds = self.encoder(**inputs)
            model_inputs["inputs_embeds"] = inputs_embeds
            return model_inputs

        if self.generation_type == "text":
            if input_ids[0][-1] == self.image_start_token
                self.setup_image_generation()
            if input_ids[0][-1] == self.video_start_token
                self.setup_video_generation()
            if input_ids[0][-1] == self.audio_start_token
                self.setup_audio_generation()

        if self.generation_type == "image":
            hidden_states = model_inputs["hidden_states"]
            input_embeds, next_tokens = self.decoder.image_decoder.lm_embed(
                hidden_states, **self.image_generation_configs
            )
            model_inputs["inputs_embeds"] = input_embeds
            self.image_tokens.append(next_tokens)
            if len(self.image_tokens) == self.image_token_num:
                self.setup_text_generation()
                self.images.append(self.decoder.image_decoder.lm_generate(
                    self.image_tokens
                )
        if self.generation_type == "video":
            ...
        if self.generation_type == "audio":
            ...

        return model_inputs
```

B.4 Parallel State of VeOmni

Listing 5 demonstrates the core interface of VeOmni's `parallel_state` API, which provides a unified and declarative initialization of all N-dimensional parallel configurations. Unlike traditional process group–based parallel management, VeOmni leverages a DeviceMesh abstraction to represent and organize various parallel topologies, including data parallel (DP), tensor parallel (TP), expert parallel (EP), pipeline parallel (PP), and Ulysses-based sequence parallel (SP). Once initialized via `init_parallel_state`, users can conveniently access mesh and group objects, ranks, and communication topologies through the global `parallel_state` object. This design simplifies the orchestration of hybrid-parallel strategies and decouples low-level distributed logic from model implementation.

Listing 5 `parallel_state` API

```
from omniscale.distributed.parallel_state import get_parallel_state, init_parallel_state

init_parallel_state(
    dp_size=data_parallel_size,
    dp_replicate_size=data_parallel_replicate_size,
    dp_shard_size=data_parallel_shard_size,
    tp_size=tensor_parallel_size,
    ep_size=expert_parallel_size,
    pp_size=pipeline_parallel_size,
```

```
    ulysses_size=ulysses_parallel_size,
    dp_mode=data_parallel_mode,
)

# Get global parallel_state
parallel_state = get_parallel_state()

# Get DP Mesh
dp_mesh = parallel_state.dp_mesh
# Get DP Group
dp_group = parallel_state.dp_group
# Get DP Shard Rank
dp_shard_rank = parallel_state.dp_shard_rank
# Get Ulysses Group
ulysses_group = parallel_state.ulysses_group
```

B.5 FSDP and parallelized

Listing 6 illustrates the usage of the **build_parallelize_model** API, which applies Fully Sharded Data Parallelism (FSDP) and a user-defined parallel plan to the target model. This interface abstracts away low-level distributed training details, such as parameter sharding, precision configuration, and gradient checkpointing. As a result, users can enable efficient large-scale distributed training with minimal code changes, achieving scalability without compromising usability.

Listing 6 `build_parallelize_model` API

```
from omniscale.distributed.torch_parallelize import build_parallelize_model

model = build_parallelize_model(
    model,
    weights_path=model_path,
    enable_full_shard=enable_full_shard,
    enable_mixed_precision=enable_mixed_precision,
    enable_gradient_checkpointing=enable_gradient_checkpointing,
    init_device=init_device,
    enable_fsdp_offload=enable_fsdp_offload,
)
```

B.6 Ulysess flash attention forward Function

Listing 7 presents the implementation of `flash_attention_forward` in VeOmni, which extends HuggingFace's native FlashAttention interface to support Ulysses-style sequence parallelism. When sequence parallelism is enabled via the global parallel state, the input query, key, and value tensors are first transformed from sequence-sharded to head-sharded layouts using all-to-all collective operations. After performing attention computation through the standard `_flash_attention_forward` function, the output is transformed back from head-sharded to sequence-sharded format. This function maintains modular compatibility with HuggingFace models, requiring no changes to attention logic, and enables seamless integration of sequence parallelism with FlashAttention.

Listing 7 Ulysess flash attention forward Function

```
from transformers.modeling_flash_attention_utils import _flash_attention_forward
from omniscale.distributed.parallel_state import get_parallel_state
from omniscale.distributed.sequence_parallel import (
    gather_heads_scatter_seq,
    gather_seq_scatter_heads,
)
```

```python
def flash_attention_forward(
    module: torch.nn.Module,
    query: torch.Tensor,
    key: torch.Tensor,
    value: torch.Tensor,
    attention_mask: Optional[torch.Tensor],
    **kwargs,
) -> Tuple[torch.Tensor, None]:

    # Ulysses all-to-all
    ulysses_enabled = get_parallel_state().ulysses_enabled
    if ulysses_enabled:
        ulysses_group = get_parallel_state().ulysses_group

        query = gather_seq_scatter_heads(
            query, seq_dim=1, head_dim=2, group=ulysses_group
        )
        key = gather_seq_scatter_heads(
            key, seq_dim=1, head_dim=2, group=ulysses_group
        )
        value = gather_seq_scatter_heads(
            value, seq_dim=1, head_dim=2, group=ulysses_group
        )

    attn_output: torch.Tensor = _flash_attention_forward(
        query,
        key,
        value,
        attention_mask,
        **kwargs,
    )

    # Ulysses all-to-all
    if ulysses_enabled:
        ulysses_group = get_parallel_state().ulysses_group

        attn_output = gather_heads_scatter_seq(
            attn_output, seq_dim=1, head_dim=2, group=ulysses_group
        )

    return attn_output, None
```

B.7 Expert Parallel Plan and Expert Parallel Function

Listing 8 demonstrates the definition of an expert parallel (EP) plan using VeOmni's `ParallelPlan` interface. This declarative design allows users to specify which modules (e.g., expert projection layers in an MoE block) should be sharded across expert devices. The syntax follows a wildcard-matching convention similar to PyTorch module names, and leverages DTensor-compatible placement primitives like `Shard(0)`. When passed to `build_parallelize_model()`, the plan is automatically applied to partition the target model along expert dimensions. This flexible mechanism not only simplifies expert parallelism but also enables clean integration with other parallelism strategies such as tensor parallelism and FSDP.

Listing 8 ParallelPlan API

```python
from omniscale.distributed.parallel_plan import ParallelPlan

def get_paralle_plan():
```

```
    ep_plan = {
        "model.layers.*.mlp.experts.gate_proj": Shard(0),
        "model.layers.*.mlp.experts.up_proj": Shard(0),
        "model.layers.*.mlp.experts.down_proj": Shard(0),
    }
    parallel_plan = ParallelPlan(
        ep_plan=ep_plan,
    )
    return parallel_plan
```