GeoServe: Leveraging Disaggregated Data Processing for Scalable Geospatial Model Serving

Gerard Finol*
gerard.finol@urv.cat
Universitat Rovira i Virgili
Tarragona, Spain

Christian Pinto
christian.pinto@ibm.com
IBM Research Europe
Dublin, Ireland

Abstract

Geospatial foundation models (GFMs) operate on large, multi-band raster products (e.g., GeoTIFF) that require expensive data access and preprocessing – reprojection, decoding, normalization, and tiling – before GPU inference. In our measurements, reading and preprocessing geospatial inputs can be orders of magnitude slower than tokenization or standard image preprocessing, and constitute 31 – 43% of end-to-end request time for a representative GFM. Existing inference frameworks such as vLLM execute this preprocessing inline with request handling, which under load serializes CPU and I/O work, increasing queueing delay, and leaving GPUs underutilized. We present GeoServe, a Ray-based serving system that decouples the geospatial data pipeline from GPU inference by disaggregating I/O- and CPU-heavy preprocessing to a scalable pool of CPU workers, while keeping GPU nodes dedicated to model forward passes. We show experimentally that GeoServe reduces the p90 request latency by up to 414.9× at high load and improves throughput by up to 4.74× compared to vanilla vLLM, while increasing the achieved model forward-pass rate from ∼ 16 inf./sec to ∼ 72 inf./sec via better batching opportunities.

CCS Concepts: • **Computing methodologies** → **Distributed artificial intelligence**; • **Computer systems organization** → **Distributed architectures**.

Keywords: Geospatial Foundation Models, Model Serving, Disaggregation, Preprocessing, vLLM, Ray

ACM Reference Format:
Gerard Finol and Christian Pinto. 2026. GeoServe: Leveraging Disaggregated Data Processing for Scalable Geospatial Model Serving. In *The 6th Workshop on Machine Learning and Systems (EuroMLSys '26), April 27–30, 2026, Edinburgh, Scotland Uk.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3805621.3807611

*This work was done as part of a research secondment at IBM Research Europe (Dublin).

EuroMLSys '26, Edinburgh, Scotland Uk

ACM ISBN 979-8-4007-2605-7/26/04
https://doi.org/10.1145/3805621.3807611

1 Introduction

The rapid evolution of foundation models has revolutionized artificial intelligence, with most advances centered on large language models (LLMs) [4, 2] and multimodal architectures for text, image, and audio data [22, 8, 1]. These models have benefited from highly optimized frameworks and inference pipelines tailored to the specific characteristics of their input modalities [11, 15, 23, 13]. However, as foundation models are increasingly applied to new domains [20], such as geospatial data, significant challenges arise when serving these models, due to the unique nature of the input data [24].

Geospatial data, unlike data for more common models (such as text or RGB images), often consist of multi-band (multispectral) imagery, variable spatial resolutions, diverse cartographic projections, and specialized file formats (e.g., GeoTIFF, NetCDF, HDF5). These features necessitate complex preprocessing steps (e.g., reprojection, cropping, resampling, normalization, or tiling) before the data can be fed into a model [24]. Such preprocessing is computationally intensive and introduces substantial latency into the inference workflow. This also impacts serving throughput, especially under high-load scenarios, where long preprocessing stages, if not parallelized across requests and decoupled from inference, limit the throughput of a model-serving instance.

In contrast to text-based models, where preprocessing is typically limited to tokenization and generation of multimodal embeddings [25], geospatial foundation models require a much more elaborate and resource-demanding pipeline [24]. This can create bottlenecks, resulting in underutilization of inference hardware such as GPUs, and ultimately limiting the scalability and responsiveness of geospatial AI systems [3].

To solve similar issues in the context of LLMs, some works have explored disaggregation as a possible solution. Recent examples include multimodal models that decouple modality-specific tasks [18], while text-based models disaggregate inference phases (e.g., encoding, prefill, and decoding) [28, 19, 5].

To address these challenges, we propose GeoServe, a system designed to serve geospatial foundation models. By decoupling preprocessing from inference, GeoServe enables more efficient resource utilization, reduces end-to-end latency under heavy loads, and improves throughput for real-world geospatial applications.

Our contributions are as follows:

- We analyze the unique characteristics of geospatial data preprocessing.
- We design and implement GeoServe, a system that disaggregates the inference pipeline for geospatial foundation models, allowing independent scaling of preprocessing and inference.
- We validate GeoServe on real-world geospatial data, demonstrating its effectiveness in reducing latency and increasing GPU utilization compared to vLLM, an industry-standard inference framework.

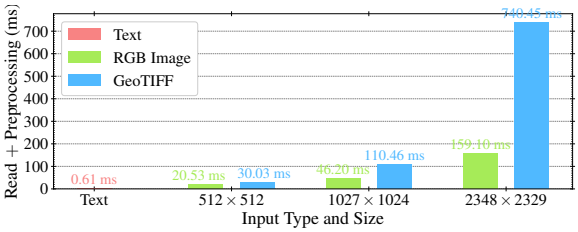


Figure 1. Read and preprocessing time comparison between text/image data and geospatial data.

2 Background

In this section, we provide an overview of geospatial foundation models (GFMs) and highlight the key differences in preprocessing requirements compared to large language models (LLMs). We also discuss the challenges of using existing inference frameworks, such as vLLM, for serving GFMs.

2.1 Geospatial Foundation Models

Geospatial Foundation Models (GFMs) are a class of models used for earth observation applications [9, 7, 21, 26, 10, 27] that range from flood and burn scar detection to land use, forest canopy height estimation, and others. GFMs have gained popularity because they can be applied to satellite images at different times to study the effects of deforestation, pollution, and natural disasters. Unlike multimodal LLMs, that usually process standard image types (e.g., JPEG, PNG, etc.), GFMs process satellite imagery often including multispectral information (images in multiple wavelength ranges), cartographic information, GPS coordinates and use various formats such as GeoTIFF, NetCDF, etc. In addition, the size of the images processed ranges from tens of MBs to GBs, depending on the dimensions of the area in the image and the number of bands. An example GFM, that we will use as case study in this article, is the Prithvi models family [21] based on the Vision Transformer (ViT) architecture.

2.2 Preprocessing differences

The preprocessing in LLMs, both text-based or multimodal, is generally minimal and normally consists of tokenization for text data, and fast resizing and embeddings calculation for image data [25]. In contrast, geospatial data require more preprocessing. For example, GeoTIFF files may be large (up to several GBs) [16], contain multiple bands, or need to be split into several tiles. In Figure 1, we show a comparison of the preprocessing time required for text and images for a multimodal LLM (LLaVA-1.5 [12]), and the preprocessing time for a GeoTIFF to be used with the Prithvi geospatial model. In this comparison, we use the same image resolution for both PNG and GeoTIFF. From our measurements, the time taken to read and preprocess the geospatial data ranges from 50% to 4.6× higher than that for image data. Compared to text data, the difference is even more pronounced, with geospatial data requiring up to 1213× more time for reading and preprocessing. This highlights that geospatial data loading and preprocessing can become a bottleneck in the inference pipeline if the frameworks used are not optimized for such data types and requirements.

2.3 Using the vLLM framework for geospatial models

vLLM [11] is a high-performance inference framework optimized for large language models (LLMs). Recently, vLLM was enhanced to support GFMs by leveraging multimodal embedding models and the IO Processor plugin [17], which enables pre- and post-processing of model inputs and outputs. However, vLLM is primarily designed for text-based models, and assumes that the preprocessing step is minimal, executing it on the same thread as the API serving. When serving GFMs with vLLM, the long data acquisition and preprocessing required can create a bottleneck that blocks the serving thread, preventing the GPU from being fully utilized during inference and delaying the handling of other incoming requests. This results in increased latency and reduced throughput, undermining the benefits of using a high-performance inference framework.

3 GeoServe Design

In this section, we describe the architecture of GeoServe, which is composed of three primary layers: a data processing service operating on CPU-only nodes, an inference service running on GPU-equipped nodes, and an asynchronous data access connectivity layer. The central principle of GeoServe is the disaggregation of the data processing from the model inference engine, enabling each stage to be scaled and optimized independently. This separation also allows for pipelining of data loading, preprocessing, inference tasks and post-processing, a key advantage for Python-based systems, where the Global Interpreter Lock (GIL) can limit concurrency. This architectural separation enables efficient and cost-effective resource allocation. As demonstrated in Section 4, it reduces end-to-end latency under load and prevents GPU idle time caused by CPU-bound or I/O bottlenecks.

Figure 2 provides an overview of GeoServe's architecture, illustrating the interaction between its components.

3.1 Data Access Layer

Geospatial data is typically large (up-to GBs) and usually accessed via web based services or hosted in storage systems (e.g., Cloud object storage, distributed file system, etc.) accessible to the data processing pipeline. Geospatial inference requests often include metadata pointers to the data location, rather than embedding the data itself, and data retrieval is part of the data processing pipeline.

GeoServe features a flexible data access layer designed to efficiently handle large geospatial files across multiple data access backends. Data processing workers access this layer asynchronously, retrieving raw binary data without blocking, enabling concurrent processing of multiple requests. By abstracting the storage interface, GeoServe supports a variety of data backends and optimizes data transfer for high-throughput workloads. As illustrated in the architecture's "Data Backend" module, the supported data backends include:

- Object Storage: For cloud-native deployments (e.g., S3-compatible buckets).
- Persistent Volumes (PVC): For Kubernetes-based deployments requiring high-throughput file system access.
- Local/Remote HTTP Servers: For direct file access.

3.2 Data processing Service

The data processing service consists of a pool of CPU-only workers responsible for asynchronous data loading through the data access layer, and all geospatial data preparation tasks required before and after inference. The data processing service performs the following tasks: (i) Asynchronous data loading from storage.; (ii) Extraction of multiframe pixel data and metadata.; (iii) Band selection and normalization.; (iv) Spatial tiling (padding and splitting images into fixed–size windows).; (v) Standardization to match the model's expectations.; and (vi) Post-processing of inference outputs (e.g., merging tiled results).

Each data processing worker operates independently, processing incoming requests from a queue and producing pre-processed tensors ready for inference. This design enables horizontal scalability, allowing the number of workers to be adjusted independently of GPU resources to efficiently meet data processing demand. After inference, the data processing service handles post-processing of the inference outputs and returns the final results to the user.

3.3 Inference Service

The inference service is hosted on GPU-equipped nodes and is dedicated to running the models' forward pass. This service receives tensors generated by the data processing service. After computation, the inference results are returned

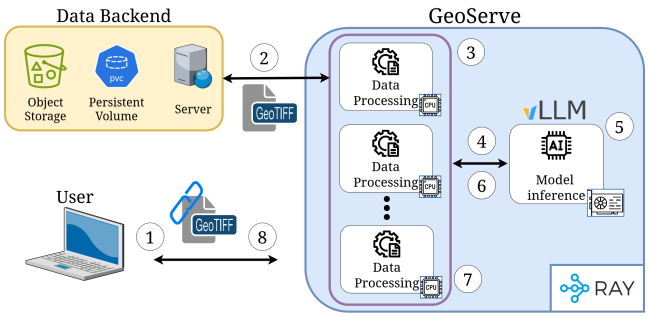


Figure 2. GeoServe design overview.

to the data processing service, which will post-process the inference outputs, and return the final result to the user. The inference service is stateful, maintaining the model weights in GPU memory across requests to avoid repeated loading and to ensure rapid response times.

3.4 Execution Workflow

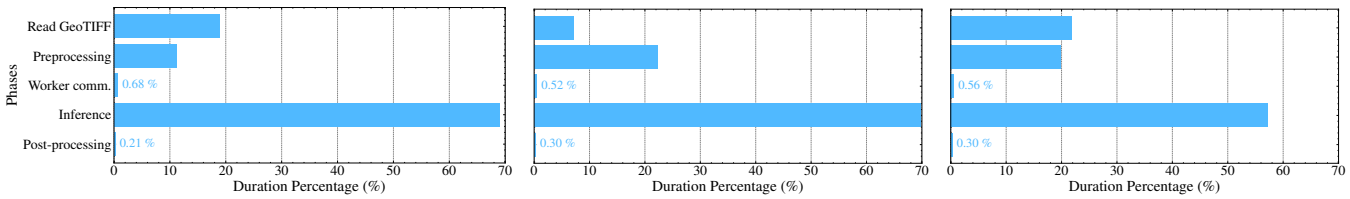
Figure 2 illustrates the execution workflow of GeoServe. The key steps involved in processing a geospatial inference request are as follows:

1. **Request Submission:** The user submits an inference request ①, which contains metadata pointing to the geospatial data source.
2. **Data Ingestion:** A data processing worker receives the metadata associated with the inference request and retrieves the corresponding GeoTIFF from the data backend ②.
3. **Preprocessing:** The data processing worker processes the geospatial data, splitting it into fixed-size tiles according to the model's requirements ③, and produces a batch of tensors ready for inference.
4. **Inference Handoff:** The processed batch of tensors is transmitted to the inference service ④.
5. **Computation and Response:** The inference node executes the model ⑤ and returns the output tensor to the data processing worker ⑥.
6. **Post-processing:** The data processing worker gathers all tiled inference results, and post-process them ⑦, generating a unique result which is returned to the user ⑧.

3.5 Implementation details

GeoServe[1] is implemented on top of Ray, a high-performance distributed computing framework. We use Ray Actors [14] to manage both the stateful inference service and the data processing workers, enabling long-lived actors to maintain resources and parallelize work. Communication between data processing and inference actors is handled via Ray Serve

[1]The GeoServe source code is available at https://github.com/gfinol/GeoServe and at Zenodo [6].



(a) *Small* input size **(b)** *Medium* input size **(c)** *Large* input size

Figure 3. Phase breakdown for different input sizes.

DeploymentHandles, which route requests over HTTP or gRPC. For large data objects, Ray's shared memory object store is used, while smaller objects are sent directly over the network. This hybrid approach optimizes data transfer based on object size, minimizing latency while simplifying the communication layer and its implementation complexity. The inference service uses vLLM as the underlying model serving framework, configured to perform no preprocessing. By leveraging vLLM, we benefit from its optimized batching and scheduling capabilities.

4 Validation

In this section, we validate the performance of GeoServe by measuring end-to-end latency, throughput and GPU utilization under varying request rates and input sizes. We compare the results against a baseline setup where preprocessing is performed on vanilla vLLM, highlighting the improvements achieved by disaggregating preprocessing from inference with GeoServe.

4.1 Configuration

Experimental Setup: All experiments were conducted on an OpenShift (v4.19) cluster. For GeoServe, we used a Ray cluster consisting of a head node (4 CPUs, 16 GB RAM) and a worker node (40 CPUs, 40 GB RAM) equipped with a single NVIDIA A100 80GB GPU. The software stack included Ray version 2.48.0, Python 3.12, and vLLM 0.10.2. For the vanilla vLLM baseline, we deployed vLLM on a pod with 40 CPUs, 40 GB RAM, and a single NVIDIA A100 80GB GPU, matching the hardware configuration of the GeoServe inference node. The vanilla vLLM baseline uses the IO Processor plugin [17] to handle geospatial data pre- and post-processing. This setup ensures a fair comparison between both systems under equivalent hardware and software conditions. During all the experiments, the GeoTIFF files were stored in a PersistentVolumeClaim (PVC) accessible to all the deployments involved with the experiments.

Dataset: We use three different GeoTIFF files for validation, representing *small*, *medium*, and *large* input sizes. The characteristics of each file are summarized in Table 1. The Prithvi tiles column indicates the number of tiles the input is divided into during preprocessing. At inference time, if not batched, each tile corresponds to one model forward pass.

Table 1. Validation dataset characteristics

File	Size (MB)	Bands	Resolution	Prithvi tiles
Small	2.2	13	512×512	1
Medium	29.5	7	1027×1024	6
Large	43.3	7	2348×2329	25

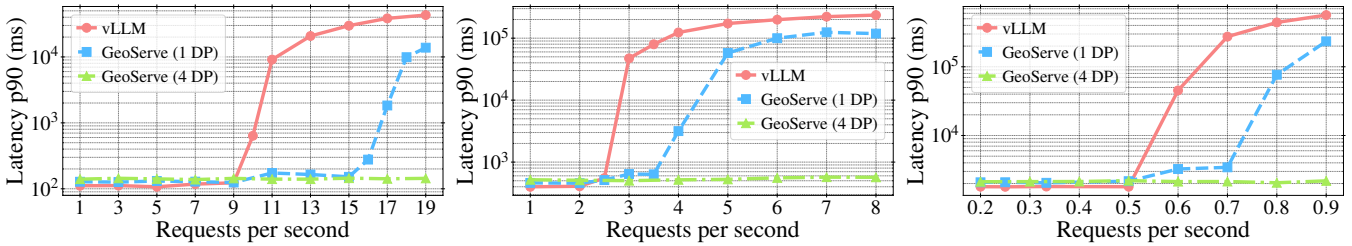
GeoServe: In this article, we do not study the effects of horizontally scaling the inference service, as we assume it will naturally increase throughput. We focus instead on scaling the number of data processing (DP) instances to study their impact on a single inference instance performance. We conduct experiments with 1, 2, 4, and 8 DP workers, all using a single inference node with one GPU.

4.2 GeoServe Inference Request Breakdown

We begin by analyzing the breakdown of the different phases involved in the inference process with GeoServe, using a single data processing worker and a single inference node. Figures 3a, 3b, and 3c present the phase breakdown for *small*, *medium*, and *large* input sizes, respectively.

The results indicate that inference time is the dominant contributor to total latency, accounting for between 56% and 70% of the overall processing time, depending on the input size. Data preprocessing constitutes 11% to 22% of the total latency, while data loading from data backend represents 7% to 22%. This phase includes both reading the raw bytes from the PVC and the time spent by the `rasterio` library decoding the image and loading it into its internal in-memory data structures, causing it to not grow proportionally to file size. Data post-processing consistently accounts for less than 1% of the total latency across all input sizes, amounting to approximately 5 ms for the largest input. This is expected, as post-processing primarily involves merging the tiled outputs from the model, which is a relatively lightweight operation.

Overall, the data processor workload (i.e., read data from PVC, preprocessing, and post-processing) comprises between 31% and 43% of the total latency. While not the dominant factor, it remains a significant portion that can impact overall system performance, particularly under high-load scenarios with concurrent requests. Communication between the data processing node and the inference node contributes less



(a) P90 latency vs requests per second for *small* input size.

(b) P90 latency vs requests per second for *medium* input size.

(c) P90 latency vs requests per second for *large* input size.

Figure 4. P90 latency vs requests per second for *small*, *medium*, and *large* input sizes.

than 1% of the total latency for all input sizes, demonstrating that the communication channel is efficient and does not introduce significant overhead.

The average end-to-end latency for a single request with GeoServe is 100.39 ms for the *small* input, 376.50 ms for the *medium* input, and 1769.37 ms for the *large* input; for comparison, vLLM's latencies for the same inputs are 102.47 ms, 370.80 ms, and 1709.85 ms, respectively.

4.3 Latency-throughput tradeoff

We evaluate how the request rate impacts end-to-end latency for GeoServe and vanilla vLLM by issuing 1,000 requests at a fixed rate and recording their response latencies. Figures 4a, 4b, and 4c show the p90 latency versus requests per second (RPS) for *small*, *medium*, and *large* inputs, respectively.

The results demonstrate that GeoServe sustains substantially higher request rates while keeping latency significantly lower across all three input sizes. For *small* inputs, GeoServe achieves up to 196.23× lower p90 latency at 15 requests per second compared to vanilla vLLM. For *medium* inputs, the reduction reaches 73.66× at 3 RPS, and for *large* inputs, GeoServe reduces latency by up to 79.78× at 0.7 RPS. Moreover, configuring GeoServe with multiple DP workers further improves scalability, enabling support for higher request rates at lower latency. With 4 DP workers, GeoServe attains up to 299.64× lower p90 latency at 19 requests per second for *small* inputs relative to vanilla vLLM. For *medium* inputs, the reduction increases to 414.95× at 8 RPS, and for *large* inputs, GeoServe reduces latency by up to 258.53× at 0.9 RPS. At low request rates, however, vanilla vLLM achieves slightly lower latencies than GeoServe. In these regimes, GeoServe latencies are 13.08% higher at 1 RPS for *small* inputs, 13.55% higher at 1 RPS for *medium* inputs, and 13.68% higher at 0.2 RPS for *large* inputs. For the *small*, *medium*, and *large* inputs, 4 DP GeoServe's latency begins to rise sharply beyond 30 RPS (to ∼ 1 s), 10 RPS (to ∼ 2 s), and 2.8 RPS (to ∼ 6 s), respectively. Beyond these points, inference should be scaled up. Latency profile for 2 and 8 DP are similar to 4 DP, and are omitted from Figure 4 for clarity.

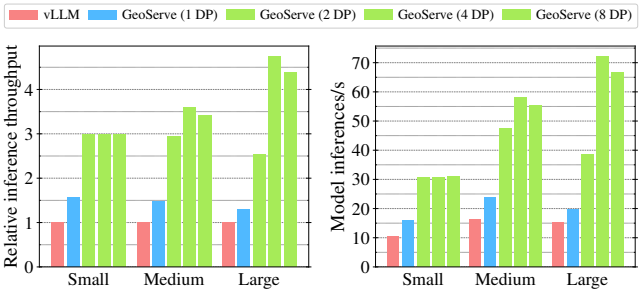
Conclusion: The effects of pre/post-processing disaggregation are not visible at lower request rates, where vLLM is between 15-17% faster than GeoServe. However, as the request rate increases, GeoServe quickly outperforms vLLM, achieving up to 196× lower latency with a single data processing worker, and up to 414× lower latency when using multiple data processing workers. More DP workers do not reduce latency at low request rates, but they preserve low latency under higher load.

4.4 Inference performance

We now compare the inference throughput achieved by GeoServe with that of vLLM for all three input sizes. For each image size, we compare the best results obtained with GeoServe and vanilla vLLM. We measure two metrics: (1) relative request throughput, which is the ratio of GeoServe's throughput to vanilla vLLM's throughput, and (2) absolute model forward passes completed per second. For example, *small* GeoTIFF results in one model forward pass per request, while *large* GeoTIFF may require up to 25 model forward passes per request.

Figure 5a shows the relative request throughput of GeoServe compared to vanilla vLLM. The results indicate that GeoServe with a single data processing worker achieves up to 55% higher request throughput for *small* inputs, 47% higher for *medium* inputs, and 28% higher for *large* inputs. When using multiple data processing workers, GeoServe further increases request throughput, achieving up to 3× higher for *small*, 3.60× higher for *medium*, and 4.74× higher for *large*.

Figure 5b presents the absolute model forward pass rate for both GeoServe and vanilla vLLM. The results show that vanilla vLLM achieves a maximum of 16 model forward passes per second, which means that there is no batching of requests taking place. We know from Section 4.2 that the average inference time for a single tile is ∼ 69*ms*, and a maximum theoretical sequential inference throughput of ∼ 14 inf./sec. In contrast, GeoServe even with a 1 DP worker achieves up to 23.73 model forward passes per second (47.89% higher). With multiple data processing workers, GeoServe reaches up to 72.09 model inferences per second (4.47×



(a) Relative throughput comparison (relative to vanilla vLLM)

(b) Model inferences per second comparison.

Figure 5. Inference performance

vanilla vLLM). This shows that by disaggregating and parallelizing the data processing of multiple requests, we create more opportunities for inferences to be submitted concurrently to the inference engine, and for vLLM to batch them together.

Conclusion: The design of GeoServe enables a significant increase in the number of requests served per second, achieving up to 55% higher throughput compared to vanilla vLLM when using an equal number of data processing threads. Furthermore, the ability to scale the number of data processors allows GeoServe to serve up to 4.74× more requests than vanilla vLLM. This increase in performance is also due to the effects of decoupling preprocessing from inference, which creates more opportunities for concurrent requests to be batched together by the inference service, leading also to much more efficient utilization of GPU resources. Observe that 4 DPs provide the best results for *medium* and *large*; beyond that point, adding more DPs is counterproductive, and the inference service should be scaled. In contrast, for *small* inputs, 2 DPs are sufficient to achieve the best results. As data processing characteristics differ across inputs, the optimal number of DPs varies, despite using the same GFM.

4.5 Phase concurrency

To help the reader visually understand the effects of GeoServe when handling multiple requests, Figures 6 and 7 show a 1-second timeline, for *small* input size using 1 DP and 4 DP. Each color represents a different phase of the inference request. By disaggregating pre-processing from inference, GeoServe is able to process two requests at the same time with 1 DP and four requests with 4 DP. In some cases we see the inference phases execute concurrently, due to batching in the inference service.

5 Related Work

Disaggregating AI inference pipelines has been explored in several recent systems. ModServe [18] modularizes serving

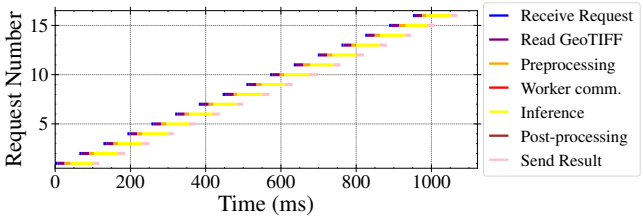


Figure 6. GeoServe phase breakdown for concurrent requests for *small* input size and 1 DP.

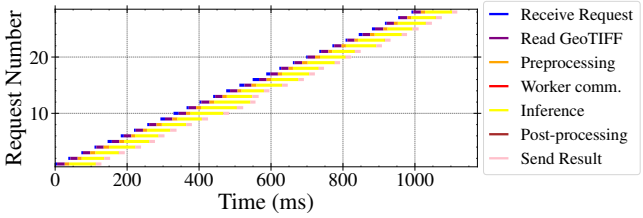


Figure 7. GeoServe phase breakdown for concurrent requests for *small* input size and 4 DP.

of large multimodal models (LMMs) by decoupling modality-specific tasks into separate services (e.g., image and text). DistServe [28] separates LLM prefill and decoding across GPUs to avoid interference. EPD [19] disaggregates encoding, prefill, and decoding onto dedicated resources, a strategy also used by HydraInfer [5], which can flexibly reallocate resources across stages. These systems, however, focus on model-internal computation and do not disaggregate data preprocessing and post-processing, which is particularly critical for some models, such as geospatial foundation models. To the best of our knowledge, GeoServe is the first system to explicitly decouple and evaluate data processing and inference stages for geospatial foundation models.

6 Conclusion

We presented GeoServe, a system designed to disaggregate the data processing pipeline from the inference pipeline for geospatial foundation models. This separation enables independent scaling of each stage, maximizing hardware utilization and minimizing inference latency under high loads. Our validation on real-world geospatial data and state of the art GFM demonstrated that GeoServe can significantly reduce latency (414.9×) and increase GPU utilization (4.74×) compared to vLLM, an industry-standard inference framework. Future work includes exploring adaptive scaling strategies for the data processing service based on workload characteristics.

Acknowledgments

We thank Boris Lublinsky for giving us part of the code that led to the implementation of GeoServe. This work was partly

funded by the European Union through the Horizon Europe CLOUDSTARS project (101086248) and by the Spanish Government through PID2023-148202OB-C21 Project. Gerard Finol is a Martí i Franquès PhD fellow at Universitat Rovira i Virgili.

References

[1] Jean-Baptiste Alayrac et al. 2022. Flamingo: a visual language model for few-shot learning. In *Proceedings of the 36th International Conference on Neural Information Processing Systems* (NIPS '22) Article 1723. Curran Associates Inc., New Orleans, LA, USA, 21 pages. ISBN: 9781713871088.

[2] Tom B. Brown et al. 2020. Language models are few-shot learners. In *Proceedings of the 34th International Conference on Neural Information Processing Systems* (NIPS '20) Article 159. Curran Associates Inc., Vancouver, BC, Canada, 25 pages. ISBN: 9781713829546.

[3] Christina Butsko, Gabriel Tseng, Kristof Van Tricht, Giorgia Milli, David Rolnick, Ruben Cartuyvels, Inbal Becker-Reshef, Zoltan Szantoi, and Hannah Kerner. 2025. Deploying geospatial foundation models in the real world: lessons from worldcereal. In *Proceedings of The TerraBytes ICML Workshop: Towards global datasets and models for Earth Observation* (Proceedings of Machine Learning Research). Nicolas Audebert et al., (Eds.) Vol. 292. PMLR, (19 Jul 2025), 13–31. https://proceedings.mlr.press/v292/butsko25a.html.

[4] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*. Jill Burstein, Christy Doran, and Thamar Solorio, (Eds.) Association for Computational Linguistics, Minneapolis, Minnesota, (June 2019), 4171–4186. doi:10.18653/v1/N19-1423.

[5] Xianzhe Dong et al. 2025. Hydrainfer: hybrid disaggregated scheduling for multimodal large language model serving, (May 2025). arXiv: 2505.12658. doi:10.48550/ARXIV.2505.12658.

[6] Gerard Finol and Christian Pinto. 2026. Geoserve. en. (2026). doi:10.5281/ZENODO.19469727.

[7] Pedram Ghamisi, Weikang Yu, Xiaokang Zhang, Aldino Rizaldy, Jian Wang, Chufeng Zhou, Richard Gloaguen, and Gustau Camps-Valls. 2025. Geospatial foundation models to enable progress on sustainable development goals, (May 2025). arXiv: 2505.24528 [cs.CV]. doi:10.48550/ARXIV.2505.24528.

[8] Rohit Girdhar, Alaaeldin El-Nouby, Zhuang Liu, Mannat Singh, Kalyan Vasudev Alwala, Armand Joulin, and Ishan Misra. 2023. Imagebind: one embedding space to bind them all. (2023). https://arxiv.org/abs/2305.05665 arXiv: 2305.05665 [cs.CV].

[9] Krzysztof Janowicz, Gengchen Mai, Weiming Huang, Rui Zhu, Ni Lao, and Ling Cai. 2025. Geofm: how will geo-foundation models reshape spatial data science and geoai? *International Journal of Geographical Information Science*, 39, 9, 1849–1865. eprint: https://doi.org/10.1080/13658816.2025.2543038. doi:10.1080/13658816.2025.2543038.

[10] Saurabh Kaushik, Lalit Maurya, Elizabeth Tellman, and ZhiJie Zhang. 2025. Assessing the value of geo-foundational models for flood inundation mapping: benchmarking models for sentinel-1, sentinel-2, and planetscope for end-users. *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing (2026)*, 19, (Nov. 2025), 5649–5665. arXiv: 2511.01990 [cs.CV]. doi:10.1109/jstars.2026.3656855.

[11] Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph Gonzalez, Hao Zhang, and Ion Stoica. 2023. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the 29th Symposium on Operating Systems Principles* (SOSP '23). Association for Computing Machinery, Koblenz, Germany, (Oct. 2023), 611–626. ISBN: 9798400702297. doi:10.1145/3600006.3613165.

[12] Haotian Liu, Chunyuan Li, Yuheng Li, and Yong Jae Lee. 2024. Improved baselines with visual instruction tuning. (2024). https://arxiv.org/abs/2310.03744 arXiv: 2310.03744 [cs.CV].

[13] llm-d Team. 2026. Llm-d. https://github.com/llm-d/llm-d. GitHub repository. (2026).

[14] Philipp Moritz et al. 2018. Ray: a distributed framework for emerging ai applications. In *Proceedings of the 13th USENIX Conference on Operating Systems Design and Implementation* (OSDI'18). USENIX Association, Carlsbad, CA, USA, 561–577. ISBN: 9781931971478.

[15] NVidia. 2026. Tensorrt. https://github.com/NVIDIA/TensorRT. GitHub repository. (2026).

[16] Open Geospatial Consortium. 2023. OGC GeoTIFF Standard v1.1. Tech. rep. OGC 19-008r4. Open Geospatial Consortium. https://www.ogc.org/standard/geotiff/.

[17] Christian Pinto. 2025. Io processor plugin for vllm. https://github.com/vllm-project/vllm/pull/22820. GitHub pull request. (2025).

[18] Haoran Qiu et al. 2025. Modserve : modality- and stage-aware resource disaggregation for scalable multimodal model serving. In *Proceedings of the 2025 ACM Symposium on Cloud Computing* (SoCC '25). ACM, (Nov. 2025), 817–830. doi:10.1145/3772052.3772254.

[19] Gursimran Singh et al. 2025. Efficiently serving large multimodal models using epd disaggregation. *International Conference on Machine Proceedings of the 42nd International Conference on Machine Learning, Vancouver, Canada. PMLR 267, 2025*, (Dec. 2025). arXiv: 2501.05460 [cs.DC]. doi:10.48550/ARXIV.2501.05460.

[20] Quirin D. Strotzer et al. 2024. Toward foundation models in radiology? quantitative assessment of gpt-4v's multimodal and multianatomic region capabilities. *Radiology*, 313, 2, (Nov. 2024). doi:10.1148/radiol.240955.

[21] Daniela Szwarcman et al. 2024. Prithvi-eo-2.0: a versatile multitemporal foundation model for earth observation applications, (Dec. 2024). arXiv: 2412.02732 [cs.CV]. doi:10.48550/ARXIV.2412.02732.

[22] Gemini Team et al. 2025. Gemini: a family of highly capable multimodal models. (2025). https://arxiv.org/abs/2312.11805 arXiv: 2312.11805 [cs.CL].

[23] The AIBrix Team et al. 2025. Aibrix: towards scalable, cost-effective large language model inference infrastructure. (2025). https://arxiv.org/abs/2504.03648 arXiv: 2504.03648 [cs.DC].

[24] Ranga Raju Vatsavai. 2024. Geospatial foundation models: recent advances and applications. In *Proceedings of the 12th ACM SIGSPATIAL International Workshop on Analytics for Big Geospatial Data* (BigSpatial '24). Association for Computing Machinery, Atlanta, GA, USA, 30–33. ISBN: 9798400711435. doi:10.1145/3681763.3698478.

[25] vLLM Team. [n. d.] Multimodal model contribution guide. https://docs.vllm.ai/en/latest/contributing/model/multimodal/. Accessed: 2026-02-20. ().

[26] Zhitong Xiong et al. 2024. Neural plasticity-inspired multimodal foundation model for earth observation, (Mar. 2024). arXiv: 2403.15356 [cs.CV]. doi:10.48550/ARXIV.2403.15356.

[27] Christopher Yeh et al. 2021. Sustainbench: benchmarks for monitoring the sustainable development goals with machine learning. In *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks*. J. Vanschoren and S. Yeung, (Eds.) Vol. 1. https://datasets-benchmarks-proceedings.neurips.cc/paper_files/paper/2021/file/950a4152c2b4aa3ad78bdd6b366cc179-Paper-round2.pdf.

[28] Yinmin Zhong, Shengyu Liu, Junda Chen, Jianbo Hu, Yibo Zhu, Xuanzhe Liu, Xin Jin, and Hao Zhang. 2024. Distserve: disaggregating prefill and decoding for goodput-optimized large language model serving. (Jan. 2024). arXiv: 2401.09670 [cs.DC]. doi:10.48550/ARXIV.2401.09670.

Received 24th February 2026; accepted 23rd March 2026