

# Lessons from the bleeding edge: large-scale production inference of LLMs

Yubin Kim<sup>1,\*</sup>, Arthur Maciejewicz<sup>1</sup> and Brandon Beveridge<sup>1</sup>

<sup>1</sup>Vody, New York, NY, USA

## Abstract

Large Language Models (LLMs) have demonstrated strong generalization capabilities across a range of natural language tasks and are increasingly being integrated into production systems. However, scalable, cost-efficient, and maintainable deployment of LLMs remains underexplored in academic literature. This paper presents our experiences building a production-grade architecture for asynchronous, large-scale batch inference of LLMs at Vody, a generative AI startup focused on product data enrichment for e-commerce. We describe the key architectural decisions that enabled us to process tens of millions of products through multiple LLMs within hours. We also share pragmatic lessons learned about tooling reliability, debugging strategies, and infrastructure design. Our goal is to provide actionable guidance for teams facing similar challenges in deploying LLMs at scale using open-source tooling.

## Keywords

LLM, efficient inference, system design

## 1. Introduction

Large Language Models (LLMs) have attracted significant attention in both academia and industry due to their strong performance and ability to generalize across many tasks. While a naive implementation of an LLM can be computationally expensive to inference, efficient inference of LLMs is a thriving area of academic research [1, 2]. In addition, a robust open-source community is translating this research into common libraries and tooling [3, 4], so that scalable inference of LLMs is becoming accessible to those without deep research expertise.

However, there is very little in the literature that discusses practical architecture design choices for productionizing LLMs using commonly available tools. Ganiev et al. discusses an system architecture for a BERT-based model. Mailach et al. and Parnin et al. survey developers for what they consider challenges in designing an LLM-based application, but do not include design recommendations or a discussion of open source tools.

Vody is an early-stage generative AI startup that builds multimodal LLMs for e-commerce. Our models enrich product catalog data in ways targeted to improve the performance of downstream search & discovery applications. Some examples of product data enrichment tasks include copywriting (e.g. generating product titles, descriptions), product attribute labeling (e.g. color, style, material), and open-ended keyword generation (e.g. related search queries). Access to our offerings is provided through a software-as-a-service (SaaS) API and is an asynchronous batch process system designed to process 10s of millions of items through multiple task-specific models in a matter of hours.

This paper chronicles our journey in designing an architecture for efficient, scalable batch inference of LLMs, detailing our decision-making and the pitfalls we encountered. We share key lessons we learned along our journey. While model training and evaluation are also valuable problems to examine, in this work, we focus on the understudied topic of scalable deployment.

---

ECOM'25: SIGIR Workshop on eCommerce, Jul 17, 2025, Padua, Italy

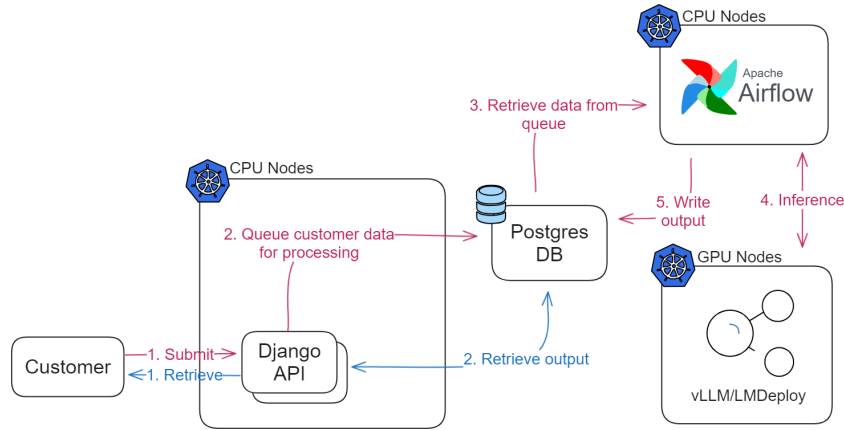
\*Corresponding author.

✉ yubin@vody.com (Y. Kim); arthur@vody.com (A. Maciejewicz); brandon@vody.com (B. Beveridge)

ORCID 0000-0001-5033-2677 (Y. Kim)



© 2025 Copyright for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).



**Figure 1:** Architecture of the product processing pipeline and data flow through the system from user request.

## 2. Architecture

The design of our system architecture was driven by the following desiderata:

1. *Consistency across different cloud providers:* our offering is deployed on both Google Cloud Platform and Amazon Web Services, thus we desire to minimize the use of cloud-specific services such that our infrastructure remains consistent across both clouds.
2. *Support for client-specific data workflows:* each of our clients have different data formats and subscribe to a different subset of our product offerings. Furthermore, our models fine-tuned to be client-specific. Thus, our system must support easily configurable custom workflows containing different data processing and model inference steps.
3. *Horizontally scalable batch inference:* our clients include large retailers that have 10s of millions of distinct items their catalog which must be re-processed on a monthly basis. In addition, there are tens of thousands of daily updates to the product catalog which must be processed. This requires a design that can quickly scale up based on the request load.
4. *Efficient serving of multiple LLMs:* as mentioned above, a client may be subscribed to multiple different product offerings. In addition, a single product offering may be executed through multiple tasks-specific models, including output quality guardrail models that ensure that generated content adhere to client-specific brand guidelines. This means a single client tenant must serve and inference multiple different LLMs.
5. *Future-proof against new open-source model releases:* one of our key value propositions is continued technology updates that incorporate the latest advances in open-source LLMs. Thus, it is imperative that our architecture can flexibly accommodate different types of base models and is easy to update and maintain.

Our architecture is designed to meet the above requirements for large-scale batch inference of LLMs with an emphasis on scalability, modularity, and operational efficiency. We implemented a system that separates data processing from model serving, standardizes API communication, and leverages containerization for deployment flexibility.

### 2.1. System overview

The overall system architecture follows a microservices design pattern with distinct components handling different aspects of the inference pipeline:

- *REST API layer:* Serves as the entry point to our system, handling request validation, authentication, and routing.

- *Orchestration layer*: Manages the flow of data through the system using Airflow directed acyclic graphs (DAGs), which coordinate batch processing tasks.
- *Processing layer*: CPU-optimized containers backed by Kubernetes that handle data preparation, pre/post-processing, and business logic.
- *Inference layer*: GPU-optimized containers backed by Kubernetes dedicated to model serving with minimal dependencies beyond inference requirements.
- *Storage layer*: Persistent storage for models, inference results, and logging data.

To meet Desideratum 1, we use open-source tools that are cross-cloud compatible where possible. Figure 1 presents an overview of our system architecture and how data flows through the system based on an API request by the user. Users can interact with our system in two ways. First, they can upload product data to our API to initiate a request for data enrichment, which is handled in a batch asynchronous fashion and shown with the red arrows in the Figure:

1. User submits product data to the Django REST API
2. API writes product data to PostgreSQL queue and metadata tables
3. Kubernetes worker retrieves N latest products from the queue and schedules Airflow jobs on CPU worker nodes
4. Worker calls VLLM/LMDeploy service via OpenAI interface for LLM enrichment

Afterwards, the user can initiate a request to retrieve the enriched product data retrieval process, shown with the blue arrows in the Figure:

1. User retrieves processed data from the API
2. API reads processed data from PostgreSQL database

## 2.2. Airflow-based workflow management

We orchestrate and schedule data processing and model inference steps through Airflow DAGs defined in Python. Defining these workflows programmatically gives us fine-grained control over execution logic, simplifies versioning and reuse, and enables us to easily customize and swap pipelines across client tenants based on their specific requirements, meeting Desideratum 2.

## 2.3. Kubernetes configuration and resource allocation

For horizontal scalability (Desideratum 3), our computation is backed by Kubernetes. A key architectural decision was to separate CPU-intensive data processing from GPU-dependent model serving in our Kubernetes deployment. This separation addresses several challenges:

- **Container Size Management**: CUDA dependencies make model-serving containers significantly larger (often 10–20 GB versus 1–2 GB for processing containers). Maintaining this separation keeps most containers lightweight, decreasing node spin up time.
- **Resource Utilization**: GPU resources are allocated only where needed, maximizing cost efficiency and computational throughput.
- **Dependency Isolation**: We encountered numerous dependency conflicts, particularly with different versions of the `transformers`<sup>1</sup> library required by different models. Isolation prevents these conflicts.

For horizontal scaling, we implemented auto-scaling policies based on queue depth and processing latency metrics. The system can dynamically adjust the number of processing nodes while maintaining a core set of inference nodes that remain warmed up with models loaded in memory.

---

<sup>1</sup><https://huggingface.co/>

## 2.4. Model management

Our model management approach focuses on flexibility and utilizing parameter-efficient fine-tuning, specifically LoRA [8], in order to allow us to easily serve multiple task-specific models (Desideratum 4). LoRA fine-tuning significantly reduces storage requirements and deployment complexity, as only the base model needs to be loaded into GPU memory, with small LoRA adapters (~10–100 MB) swapped dynamically.

Our system is flexible to the type of base model. Throughout various points in time, we have used the following open source models: LLaMA 2 [9], LLaMA 3 [10], Qwen-VL 1.5 [11], Qwen-VL 2.5 [12], Mistral [13]. Base models and fine-tuned LoRA adapters are versioned and stored in cloud storage (S3/GCS), providing a single source of truth across all deployment environments.

To address cold-start latency, particularly for large models, we implemented a warm-up strategy where frequently used models remain loaded in memory. In future iterations, we plan to implement stateful storage attachment to Kubernetes nodes to accelerate model initialization times.

We primarily use models in the 7–8 billion parameter range, for two key reasons: a) we have empirically found it to be the sweet spot in the trade-off between accuracy and efficiency for our data enrichment tasks; b) we encountered significant operational challenges while trying to secure access to GPUs with larger VRAM capacities, such as NVIDIA A100 GPUs.

In both clouds, A100 spot instances were often entirely unavailable due to high demand. Even with quota approvals for long-term reservations, due to limited physical supply, acquiring an instance required running an automated scripts polling at five-minute intervals—often over multiple hours. Furthermore, reserving A100s on a sustained basis proved cost-prohibitive. To support horizontal scalability while maintaining reasonable availability and cost, we transitioned to using NVIDIA L40S GPUs, which were more reliably accessible. This hardware constraint was a key factor in our decision to use 7–8 billion parameter models, which can run efficiently within the memory limits of L40S instances.

## 2.5. Inference server implementation

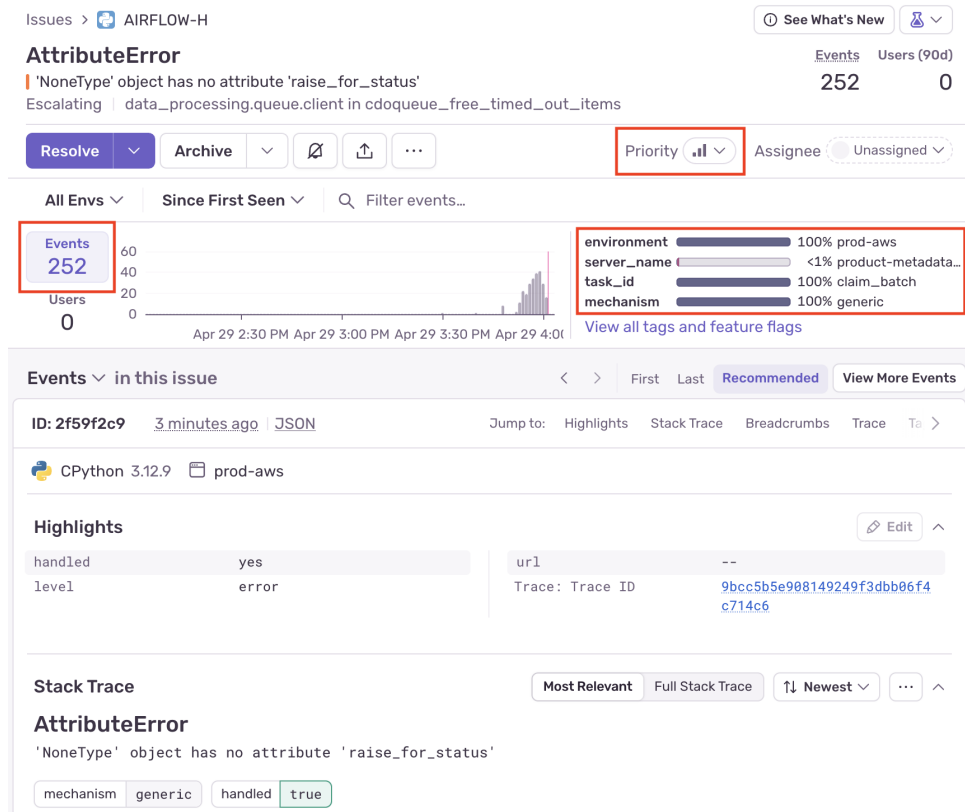
Our inference server architecture prioritizes the ability to be cross-compatible for different base models and leverages cloud storage and dynamic model loading. While our preferred inference server is vLLM [4], our architecture supports multiple different types of inference servers for broader compatibility with different base models (Desideratum 5).

- We standardized on the OpenAI API definition for inference server communication, which provides a well-documented interface that simplifies integration with various types of models and clients.
- We implemented dynamic adapter loading using vLLM’s LoRA support, allowing us to load different adapters at runtime based on request parameters without maintaining separate copies of the full model weights.
- For the Qwen-VL series of models, we implemented lmdeploy [3] which better supports the specialized architecture of these models.

Note that swapping between inference server implementations has hidden pitfalls. While they utilize the same interface, making it easy to swap them out, inference server implementations like vLLM and lmdeploy can have mechanistic differences that make configuration non-portable between them. Similarly, configuration parameters that map to the same underlying mechanism can differ in name and description, like vllm’s `--max-model-len` and lmdeploy’s `--session-len`. This tends to manifest in performance and reliability pitfalls, making tuning a separate job for each backend.

## 3. Lessons Learned and Recommendations

In this section, we present key recommendations based on our (often painful) lessons learned during the process of designing and implementing an architecture for large-scale LLM inference.



**Figure 2:** Sentry aggregates hundreds of error occurrences into one entry

One of the most important lessons we learned is that the open-source ecosystem around LLMs remains highly volatile. The underlying technologies are evolving rapidly, and many open-source projects—often developed in academic or experimental contexts—can be unstable or contain critical bugs. With that in mind, we offer the following guidance to teams getting ready to embark on similar journeys:

### 3.1. Make debugging easy

Debugging stochastic systems, i.e. systems that rely on LLM output, is inherently difficult. This combined with bleeding-edge code bases means that you will be debugging, a lot. Our most important recommendation is to make significant, up-front investments into developer experience and lean into system design choices that prioritize ease of debugging and error detection:

- **Minimize iteration time.** For effective debugging, it is critical to minimize the time between making a change and being able to test said change. We use docker-compose to run Airflow and our API server locally, while maintaining connections to remote inference servers through Kubernetes port forwarding. This hybrid approach allows us to securely expose production-equivalent inference servers to local DAG runs without the resource-intensive task of running AI models on developer machines, which reduced our iteration time from 30 minutes to less than 1 minute.
- **Invest in collated logging.** At scale, collated logs simplify maintenance. Expected, low priority, or unactionable issues can be triaged appropriately without requiring redundant investigations. Aggregations automatically organize repeated issues into bugs, reducing management toil. Causal relationships can be established through cross-analysis with other parts of your production stack. We use Sentry<sup>2</sup> for logging (Figure 2).

<sup>2</sup><https://sentry.io/>

- **Carefully curate metrics and alerting.** Ensure metrics and alerting are high quality. For example, to prevent alert exhaustion, ensure graceful exits (example: empty work queue) are not logged as failures for a given workflow.
- **Make replicating state easy.** Multi-step stochastic systems are often difficult to debug end-to-end. Log everything required to replicate a step in isolation, including intermediate data output and random seed settings, if applicable. We use a combination of home-grown scripts and tools such as Weights and Biases to be able to log and reproduce individual steps in a DAG.
- **Fail fast.** To fail fast, prefer to halt when handling errors, rather than continuing with the program. At scale, this makes it easier to identify the root cause of issues, rather than investigating a cascade of misleading comorbidities.
- **Less is more.** Our initial architecture design had several more frameworks and components, e.g. we used the Langchain framework, we were using Celery workers to manage parts of data ingestion. However, as we iterated on the design, we quickly realized that more components meant more debugging complexity. In addition, LLM-ecosystem frameworks such as Langchain, became a source of additional bugs. We thus removed unnecessary frameworks and components and simplified our architecture as much as possible: we removed the use of Langchain as found that standardizing to Open AI API was sufficient for our needs, and we removed Celery workers to ingest the data lazily as needed.

### 3.2. Follow the herd

Given the fast-moving nature of the nascent LLM tooling ecosystem, some reliance on immature technology is inevitable. Thus, wherever possible, we strongly recommend choosing established, well-documented technologies. For example, we use Airflow for workflow orchestration and Django for our API serving layer—both of which offer strong community support, mature documentation, and proven reliability in production environments, making them far easier to troubleshoot and integrate at scale. An important corollary is that GenAI coding assistants are excellent at generating code for mature frameworks compared to new libraries, which can substantially reduce developer effort.

In addition, even within the LLM ecosystem, some paths are better trodden than others: we favor using LLaMA models when reasonable because of the robust community surrounding the LLaMA project and related tooling due to its broad adoption.

There are also pairs of technologies that work better together than other options. For example, Google’s open source LLM Gemma has official published guides for integration with vLLM<sup>3</sup>. Similarly, we have found that the Qwen series of models are best supported by lmdeploy as its inference service.

### 3.3. Distrust your tools

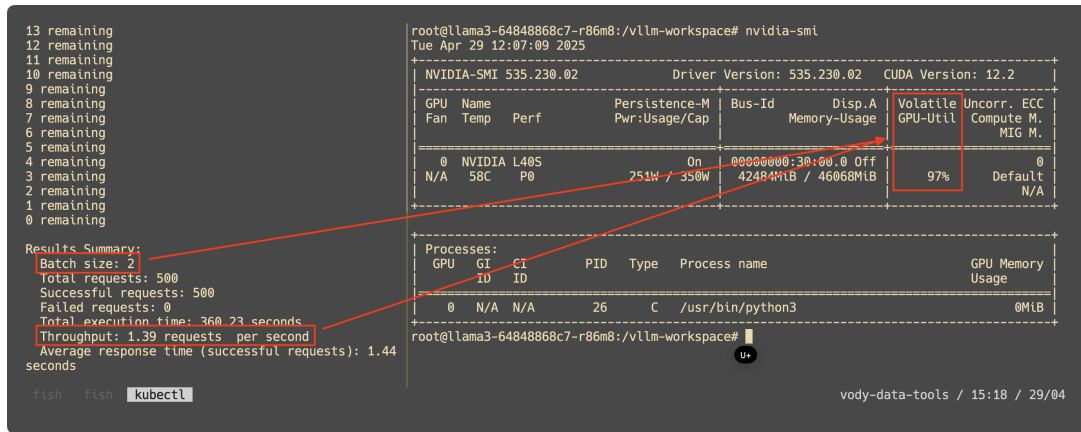
The rapid pace of development in the LLM ecosystem means that frameworks and tools are still maturing and can exhibit inconsistent or unreliable behavior. As a result, we have learned to approach these tools with a healthy degree of skepticism. In this section, we highlight one illustrative example that may be particularly relevant for teams navigating similar environments:

`nvidia-smi` is a command line tool provided by NVIDIA to monitor GPU usage. However, we came to realize that the tool’s utilization metric is a simplified proxy for GPU saturation. Inference optimizations like batching can still improve throughput even when utilization is reported to be high (Figure 3). As a proxy for relative GPU saturation, we find that power consumption is more useful, and overall recommend instrumenting your pipeline to identify your own bottlenecks with your own metrics.

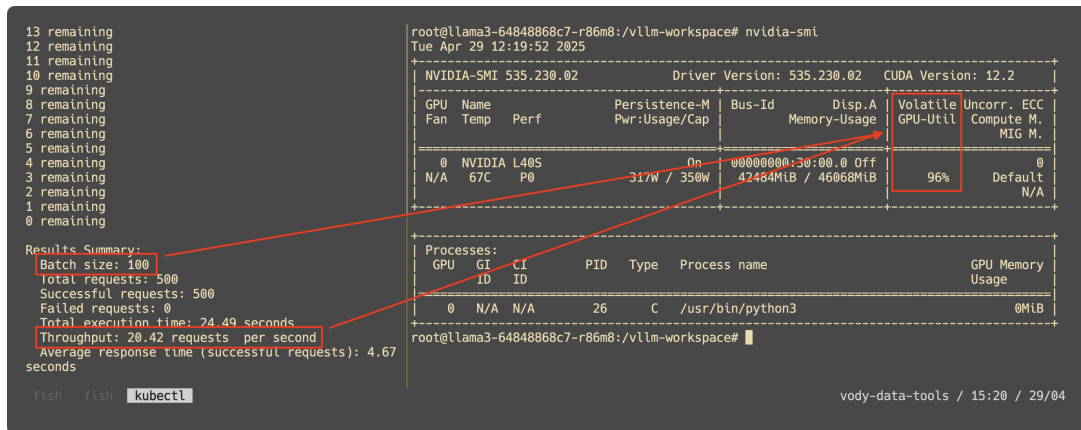
Throughout the development of our system, we encountered bugs across the entire stack, including in the LLM models themselves, as well as in model serving frameworks. In the course of addressing these issues, we actively contributed bug reports and patches to open-source projects such as LangChain and LMDeploy, helping to improve the broader ecosystem, and we encourage other teams to do the same.

<sup>3</sup><https://cloud.google.com/kubernetes-engine/docs/tutorials/serve-gemma-gpu-vllm>





(a) vLLM GPU utilization with batch size 2



(b) vLLM GPU utilization with batch size 100

**Figure 3:** The GPU utilization metric in the nvidia-smi tool can misleadingly report 100% utilization when more capacity is available.

## 4. Conclusion

Efficient large-scale inference of LLMs in production is still a developing discipline. Through our work at Vody, we have found that achieving scale, reliability, and maintainability requires careful system design choices that prioritize flexibility and ease of debugging. Our architecture, based on well-established tools such as Airflow, Kubernetes, and Django, has allowed us to meet demanding client requirements while maintaining a small, agile engineering team. However, many open-source projects in the LLM ecosystem remain immature, and unexpected failures are common. We emphasize the importance of simplifying system components, following established patterns, and maintaining a critical perspective on tooling. By sharing our architecture and key lessons, we hope to contribute to a growing body of knowledge on operationalizing LLMs and support other teams on similar paths.

## Declaration on Generative AI

During the preparation of this work, the author(s) used Writefull to: Grammar and spelling check. ChatGPT and Claude were used to: Paraphrase and reword, Improve writing style, Abstract drafting, and Drafting content. After using these tool(s)/service(s), the author(s) reviewed and edited the content as needed and take(s) full responsibility for the publication's content.

## References

- [1] R. Zhen, J. Li, Y. Ji, Z. Yang, T. Liu, Q. Xia, X. Duan, Z. Wang, B. Huai, M. Zhang, Taming the Titans: A Survey of Efficient LLM Inference Serving, 2025. URL: <http://arxiv.org/abs/2504.19720>. doi:10.48550/arXiv.2504.19720, arXiv:2504.19720 [cs].
- [2] Z. Zhou, X. Ning, K. Hong, T. Fu, J. Xu, S. Li, Y. Lou, L. Wang, Z. Yuan, X. Li, S. Yan, G. Dai, X.-P. Zhang, Y. Dong, Y. Wang, A Survey on Efficient Inference for Large Language Models, 2024. URL: <http://arxiv.org/abs/2404.14294>. doi:10.48550/arXiv.2404.14294, arXiv:2404.14294 [cs].
- [3] L. Contributors, Lmdeploy: A toolkit for compressing, deploying, and serving llm, <https://github.com/InternLM/lmdeploy>, 2023.
- [4] W. Kwon, Z. Li, S. Zhuang, Y. Sheng, L. Zheng, C. H. Yu, J. E. Gonzalez, H. Zhang, I. Stoica, Efficient Memory Management for Large Language Model Serving with PagedAttention, 2023. URL: <http://arxiv.org/abs/2309.06180>. doi:10.48550/arXiv.2309.06180, arXiv:2309.06180 [cs].
- [5] A. Ganiev, C. Chapin, A. De Andrade, C. Liu, An Architecture for Accelerated Large-Scale Inference of Transformer-Based Language Models, in: Y.-b. Kim, Y. Li, O. Rambow (Eds.), Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies: Industry Papers, Association for Computational Linguistics, Online, 2021, pp. 163–169. URL: <https://aclanthology.org/2021.naacl-industry.21/>. doi:10.18653/v1/2021.naacl-industry.21.
- [6] A. Mailach, S. Simon, J. Dorn, N. Siegmund, Themes of Building LLM-based Applications for Production: A Practitioner’s View, 2025. URL: <http://arxiv.org/abs/2411.08574>. doi:10.48550/arXiv.2411.08574, arXiv:2411.08574 [cs].
- [7] C. Parnin, G. Soares, R. Pandita, S. Gulwani, J. Rich, A. Z. Henley, Building Your Own Product Copilot: Challenges, Opportunities, and Needs, 2023. URL: <http://arxiv.org/abs/2312.14231>. doi:10.48550/arXiv.2312.14231, arXiv:2312.14231 [cs].
- [8] E. J. Hu, Y. Shen, P. Wallis, Z. Allen-Zhu, Y. Li, S. Wang, L. Wang, W. Chen, Lora: Low-rank adaptation of large language models, in: Proceedings of the Tenth International Conference on Learning Representations (ICLR), OpenReview.net, 2022. URL: <https://openreview.net/forum?id=nZeVKeeFYf9>.
- [9] H. Touvron, L. Martin, K. Stone, P. Albert, A. Almahairi, Y. Babaei, N. Bashlykov, S. Batra, P. Bhargava, S. Bhosale, D. Bikel, L. Blecher, C. C. Ferrer, M. Chen, G. Cucurull, D. Esiobu, J. Fernandes, J. Fu, W. Fu, B. Fuller, C. Gao, V. Goswami, N. Goyal, A. Hartshorn, S. Hosseini, R. Hou, H. Inan, M. Kardas, V. Kerkez, M. Khabsa, I. Kloumann, A. Korenev, P. S. Koura, M.-A. Lachaux, T. Lavril, J. Lee, D. Liskovich, Y. Lu, Y. Mao, X. Martinet, T. Mihaylov, P. Mishra, I. Molybog, Y. Nie, A. Poulton, J. Reizenstein, R. Rungta, K. Saladi, A. Schelten, R. Silva, E. M. Smith, R. Subramanian, X. E. Tan, B. Tang, R. Taylor, A. Williams, J. X. Kuan, P. Xu, Z. Yan, I. Zarov, Y. Zhang, A. Fan, M. Kambadur, S. Narang, A. Rodriguez, R. Stojnic, S. Edunov, T. Scialom, Llama 2: Open Foundation and Fine-Tuned Chat Models, 2023. URL: <http://arxiv.org/abs/2307.09288>. doi:10.48550/arXiv.2307.09288, arXiv:2307.09288 [cs].
- [10] A. Grattafiori, A. Dubey, A. Jauhri, A. Pandey, A. Kadian, A. Al-Dahle, A. Letman, A. Mathur, A. Schelten, A. Vaughan, A. Yang, A. Fan, A. Goyal, A. Hartshorn, A. Yang, A. Mitra, A. Srivankumar, A. Korenev, A. Hinsvark, A. Rao, A. Zhang, A. Rodriguez, A. Gregerson, A. Spataru, B. Roziere, B. Biron, B. Tang, B. Chern, C. Caucheteux, C. Nayak, C. Bi, C. Marra, C. McConnell, C. Keller, C. Touret, C. Wu, C. Wong, C. C. Ferrer, C. Nikolaidis, D. Allonsius, D. Song, D. Pintz, D. Livshits, D. Wyatt, D. Esiobu, D. Choudhary, D. Mahajan, D. Garcia-Olano, D. Perino, D. Hupkes, E. Lakomkin, E. AlBadawy, E. Lobanova, E. Dinan, E. M. Smith, F. Radenovic, F. Guzmán, F. Zhang, G. Synnaeve, G. Lee, G. L. Anderson, G. Thattai, G. Nail, G. Mialon, G. Pang, G. Cucurell, H. Nguyen, H. Korevaar, H. Xu, H. Touvron, I. Zarov, I. A. Ibarra, I. Kloumann, I. Misra, I. Evtimov, J. Zhang, J. Copet, J. Lee, J. Geffert, J. Vranes, J. Park, J. Mahadeokar, J. Shah, J. v. d. Linde, J. Billock, J. Hong, J. Lee, J. Fu, J. Chi, J. Huang, J. Liu, J. Wang, J. Yu, J. Bitton, J. Spisak, J. Park, J. Rocca, J. Johnstun, J. Saxe, J. Jia, K. V. Alwala, K. Prasad, K. Upasani, K. Plawiak, K. Li, K. Heafield, K. Stone, K. El-Arini, K. Iyer, K. Malik, K. Chiu, K. Bhalla, K. Lakhotia, L. Rantala-Young, L. v. d.



Maaten, L. Chen, L. Tan, L. Jenkins, L. Martin, L. Madaan, L. Malo, L. Blecher, L. Landzaat, L. d. Oliveira, M. Muzzi, M. Pasupuleti, M. Singh, M. Paluri, M. Kardas, M. Tsimpoukelli, M. Oldham, M. Rita, M. Pavlova, M. Kambadur, M. Lewis, M. Si, M. K. Singh, M. Hassan, N. Goyal, N. Torabi, N. Bashlykov, N. Bogoychev, N. Chatterji, N. Zhang, O. Duchenne, O. Çelebi, P. Alrassy, P. Zhang, P. Li, P. Vasic, P. Weng, P. Bhargava, P. Dubal, P. Krishnan, P. S. Koura, P. Xu, Q. He, Q. Dong, R. Srinivasan, R. Ganapathy, R. Calderer, R. S. Cabral, R. Stojnic, R. Raileanu, R. Maheswari, R. Girdhar, R. Patel, R. Sauvestre, R. Polidoro, R. Sumbaly, R. Taylor, R. Silva, R. Hou, R. Wang, S. Hosseini, S. Chennabasappa, S. Singh, S. Bell, S. S. Kim, S. Edunov, S. Nie, S. Narang, S. Raparthy, S. Shen, S. Wan, S. Bhosale, S. Zhang, S. Vandenhennde, S. Batra, S. Whitman, S. Sootla, S. Collot, S. Gururangan, S. Borodinsky, T. Herman, T. Fowler, T. Sheasha, T. Georgiou, T. Scialom, T. Speckbacher, T. Mihaylov, T. Xiao, U. Karn, V. Goswami, V. Gupta, V. Ramanathan, V. Kerkez, V. Gonguet, V. Do, V. Vogeti, V. Albiero, V. Petrovic, W. Chu, W. Xiong, W. Fu, W. Meers, X. Martinet, X. Wang, X. Wang, X. E. Tan, X. Xia, X. Xie, X. Jia, X. Wang, Y. Goldschlag, Y. Gaur, Y. Babaei, Y. Wen, Y. Song, Y. Zhang, Y. Li, Y. Mao, Z. D. Coudert, Z. Yan, Z. Chen, Z. Papakipos, A. Singh, A. Srivastava, A. Jain, A. Kelsey, A. Shajnfeld, A. Gangidi, A. Victoria, A. Goldstand, A. Menon, A. Sharma, A. Boesenberg, A. Baevski, A. Feinstein, A. Kallet, A. Sangani, A. Teo, A. Yunus, A. Lupu, A. Alvarado, A. Caples, A. Gu, A. Ho, A. Poulton, A. Ryan, A. Ramchandani, A. Dong, A. Franco, A. Goyal, A. Saraf, A. Chowdhury, A. Gabriel, A. Bharambe, A. Eisenman, A. Yazdan, B. James, B. Maurer, B. Leonhardi, B. Huang, B. Loyd, B. D. Paola, B. Paranjape, B. Liu, B. Wu, B. Ni, B. Hancock, B. Wasti, B. Spence, B. Stojkovic, B. Gamido, B. Montalvo, C. Parker, C. Burton, C. Mejia, C. Liu, C. Wang, C. Kim, C. Zhou, C. Hu, C.-H. Chu, C. Cai, C. Tindal, C. Feichtenhofer, C. Gao, D. Civin, D. Beaty, D. Kreymer, D. Li, D. Adkins, D. Xu, D. Testuggine, D. David, D. Parikh, D. Liskovich, D. Foss, D. Wang, D. Le, D. Holland, E. Dowling, E. Jamil, E. Montgomery, E. Presani, E. Hahn, E. Wood, E.-T. Le, E. Brinkman, E. Arcaute, E. Dunbar, E. Smothers, F. Sun, F. Kreuk, F. Tian, F. Kokkinos, F. Ozgenel, F. Caggioni, F. Kanayet, F. Seide, G. M. Florez, G. Schwarz, G. Badeer, G. Swee, G. Halpern, G. Herman, G. Sizov, Guangyi, Zhang, G. Lakshminarayanan, H. Inan, H. Shojanazeri, H. Zou, H. Wang, H. Zha, H. Habeeb, H. Rudolph, H. Suk, H. Aspegren, H. Goldman, H. Zhan, I. Damlaj, I. Molybog, I. Tufanov, I. Leontiadis, I.-E. Veliche, I. Gat, J. Weissman, J. Geboski, J. Kohli, J. Lam, J. Asher, J.-B. Gaya, J. Marcus, J. Tang, J. Chan, J. Zhen, J. Reizenstein, J. Teboul, J. Zhong, J. Jin, J. Yang, J. Cummings, J. Carvill, J. Shepard, J. McPhie, J. Torres, J. Ginsburg, J. Wang, K. Wu, K. H. U, K. Saxena, K. Khandelwal, K. Zand, K. Matosich, K. Veeraraghavan, K. Michelena, K. Li, K. Jagadeesh, K. Huang, K. Chawla, K. Huang, L. Chen, L. Garg, L. A, L. Silva, L. Bell, L. Zhang, L. Guo, L. Yu, L. Moshkovich, L. Wehrstedt, M. Khabsa, M. Avalani, M. Bhatt, M. Mankus, M. Hasson, M. Lennie, M. Reso, M. Groshev, M. Naumov, M. Lathi, M. Keneally, M. Liu, M. L. Seltzer, M. Valko, M. Restrepo, M. Patel, M. Vyatskov, M. Samvelyan, M. Clark, M. Macey, M. Wang, M. J. Hermoso, M. Metanat, M. Rastegari, M. Bansal, N. Santhanam, N. Parks, N. White, N. Bawa, N. Singhal, N. Egebo, N. Usunier, N. Mehta, N. P. Laptev, N. Dong, N. Cheng, O. Chernoguz, O. Hart, O. Salpekar, O. Kalinli, P. Kent, P. Parekh, P. Saab, P. Balaji, P. Rittner, P. Bontrager, P. Roux, P. Dollar, P. Zvyagina, P. Ratanchandani, P. Yuvraj, Q. Liang, R. Alao, R. Rodriguez, R. Ayub, R. Murthy, R. Nayani, R. Mitra, R. Parthasarathy, R. Li, R. Hogan, R. Battey, R. Wang, R. Howes, R. Rinott, S. Mehta, S. Siby, S. J. Bondu, S. Datta, S. Chugh, S. Hunt, S. Dhillon, S. Sidorov, S. Pan, S. Mahajan, S. Verma, S. Yamamoto, S. Ramaswamy, S. Lindsay, S. Lindsay, S. Feng, S. Lin, S. C. Zha, S. Patil, S. Shankar, S. Zhang, S. Zhang, S. Wang, S. Agarwal, S. Sajuyigbe, S. Chintala, S. Max, S. Chen, S. Kehoe, S. Satterfield, S. Govindaprasad, S. Gupta, S. Deng, S. Cho, S. Virk, S. Subramanian, S. Choudhury, S. Goldman, T. Remez, T. Glaser, T. Best, T. Koehler, T. Robinson, T. Li, T. Zhang, T. Matthews, T. Chou, T. Shaked, V. Vontimitta, V. Ajayi, V. Montanez, V. Mohan, V. S. Kumar, V. Mangla, V. Ionescu, V. Poenaru, V. T. Mihailescu, V. Ivanov, W. Li, W. Wang, W. Jiang, W. Bouaziz, W. Constable, X. Tang, X. Wu, X. Wang, X. Wu, X. Gao, Y. Kleinman, Y. Chen, Y. Hu, Y. Jia, Y. Qi, Y. Li, Y. Zhang, Y. Zhang, Y. Adi, Y. Nam, Yu, Wang, Y. Zhao, Y. Hao, Y. Qian, Y. Li, Y. He, Z. Rait, Z. DeVito, Z. Rosnbrick, Z. Wen, Z. Yang, Z. Zhao, Z. Ma, The Llama 3 Herd of Models, 2024. URL: <http://arxiv.org/abs/2407.21783>. doi:10.48550/arXiv.2407.21783, arXiv:2407.21783 [cs].

- [11] J. Bai, S. Bai, Y. Chu, Z. Cui, K. Dang, X. Deng, Y. Fan, W. Ge, Y. Han, F. Huang, B. Hui, L. Ji, M. Li,

- J. Lin, R. Lin, D. Liu, G. Liu, C. Lu, K. Lu, J. Ma, R. Men, X. Ren, X. Ren, C. Tan, S. Tan, J. Tu, P. Wang, S. Wang, W. Wang, S. Wu, B. Xu, J. Xu, A. Yang, H. Yang, J. Yang, S. Yang, Y. Yao, B. Yu, H. Yuan, Z. Yuan, J. Zhang, X. Zhang, Y. Zhang, Z. Zhang, C. Zhou, J. Zhou, X. Zhou, T. Zhu, Qwen Technical Report, 2023. URL: <http://arxiv.org/abs/2309.16609>. doi:10.48550/arXiv.2309.16609, arXiv:2309.16609 [cs].
- [12] S. Bai, K. Chen, X. Liu, J. Wang, W. Ge, S. Song, K. Dang, P. Wang, S. Wang, J. Tang, H. Zhong, Y. Zhu, M. Yang, Z. Li, J. Wan, P. Wang, W. Ding, Z. Fu, Y. Xu, J. Ye, X. Zhang, T. Xie, Z. Cheng, H. Zhang, Z. Yang, H. Xu, J. Lin, Qwen2.5-VL Technical Report, 2025. URL: <http://arxiv.org/abs/2502.13923>. doi:10.48550/arXiv.2502.13923, arXiv:2502.13923 [cs].
- [13] A. Q. Jiang, A. Sablayrolles, A. Mensch, C. Bamford, D. S. Chaplot, D. de las Casas, F. Bressand, G. Lengyel, G. Lample, L. Saulnier, L. R. Lavaud, M.-A. Lachaux, P. Stock, T. L. Scao, T. Lavril, T. Wang, T. Lacroix, W. E. Sayed, Mistral 7b, 2023. URL: <https://arxiv.org/abs/2310.06825>. arXiv:2310.06825.