# Towards Resource-Efficient Compound AI Systems

Gohar Irfan Chaudhry[1], Esha Choukse[2], Íñigo Goiri[2], Rodrigo Fonseca[2], Adam Belay[1], Ricardo Bianchini[3]
[1]MIT CSAIL [2]Microsoft Azure Research – Systems [3] Microsoft Azure

## ABSTRACT

Compound AI Systems, integrating multiple interacting components like models, retrievers, and external tools, have emerged as essential for addressing complex AI tasks. However, current implementations suffer from inefficient resource utilization due to tight coupling between application logic and execution details, a disconnect between orchestration and resource management layers, and the perceived exclusiveness between efficiency and quality.

We propose a vision for resource-efficient Compound AI Systems through a *declarative workflow programming model* and an *adaptive runtime system* for dynamic scheduling and resource-aware decision-making. Decoupling application logic from low-level details exposes levers for the runtime to flexibly configure the execution environment and resources, without compromising on quality. Enabling collaboration between the workflow orchestration and cluster manager enables higher efficiency through better scheduling and resource management.

We are building a prototype system, called ***Murakkab***, to realize this vision. Our preliminary evaluation demonstrates speedups up to ∼ 3.4× in workflow completion times while delivering ∼ 4.5× higher energy efficiency, showing promise in optimizing resources and advancing AI system design.

**ACM Reference Format:**
Gohar Irfan Chaudhry, Esha Choukse, Íñigo Goiri, Rodrigo Fonseca, Adam Belay, Ricardo Bianchini. 2025. Towards Resource-Efficient Compound AI Systems. In *Workshop in Hot Topics in Operating Systems (HOTOS 25), May 14–16, 2025, Banff, AB, Canada.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3713082.3730377

## 1 INTRODUCTION

A Compound AI System is "a system that tackles complex tasks using multiple interacting components, including multiple calls to different AI models, retrievers, or external tools" [42]. With increasing task complexity and model capabilities, the workflows promise to grow deeper, more complex, and self-morphing with self-improving agents. Today, workflows are designed by explicitly defining the components, their interactions, and the allocation of resources.

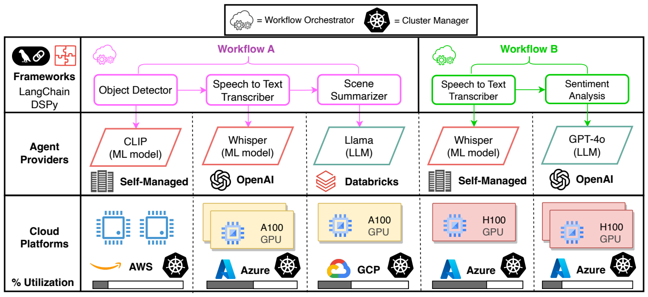

**Figure 1: Today programmers use frameworks to call agents from *different* providers hosted on *multiple* cloud platforms. The rigid coupling between all layers of the system results in inefficiencies.**

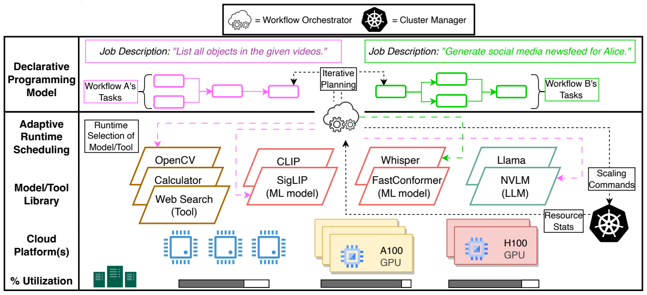

**Figure 2: We envision *fungible workflows* with high-level descriptions, managed jointly by the Workflow Orchestrator and Cluster Manager. This allows higher resource multiplexing between independent workflows to improve efficiency.**

While effective for many tasks, these workflows in practice frequently suffer from inefficient resource utilization. Figure 1 shows a typical Compound AI workflow deployment, with multiple stages. Each stage in the workflow comprises of three key entities:

*Programming Frameworks* to create workflows by composing *agents* including LLMs, ML models, and tools. They have *workflow orchestrators* that may decide agent execution order, optimize prompts for LLMs, process intermediate outputs and provide memory to stateless models etc. Examples include LangChain [23], LlamaIndex [25] and DSPy [38].

*Agent Providers* offer specific models, tools, or vector databases etc. typically through REST APIs, to invoke from workflows. These could be proprietary (*e.g.,* GPT 4o [19])

*HOTOS 25, May 14–16, 2025, Banff, AB, Canada*

ACM ISBN 979-8-4007-1475-7/2025/05.
https://doi.org/10.1145/3713082.3730377

or open-source (*e.g.,* Llama [18]) models. They may also provide additional features like analytics, data storage and management etc. Example providers include OpenAI [31] and Databricks [14].

*Cloud Platforms* rent out hardware infrastructure like GPUs, CPUs, and storage for running models, tools and vector databases etc. They may individually run *cluster managers* to monitor reasource utilization, take scaling decisions and perform load balancing among instances etc. Example platforms include AWS [1], Azure [27], and GCP [17].

Entities in the stack have differing efficiency objectives: (a) programmers prioritize result quality, (b) agent providers aim to offer diverse tools at lower costs, and (c) cloud platforms focus on maximizing utilization and profitability. Workflows spanning multiple providers and platforms add complexity.

**Challenges.** State-of-the-art AI workflows raise:

(1) Tight coupling of application logic with execution configurations (*e.g.,* model and hardware) restricts efficient alternatives.
(2) Disconnects between workflow orchestration and cluster management (often separately owned) result in suboptimal scheduling.
(3) Balancing resource efficiency (*e.g.,* cost, power) with end-to-end result quality (model accuracy) is difficult, as over-provisioning fragments resources and under-provisioning degrades performance.

These inefficiencies affect all entities, as workflow users and agent providers either pay for unused resources or experience degraded performance, while cloud platforms suffer from poor resource utilization.

**Our Work.** We believe that future Compound AI Systems will become increasingly complex. These systems are likely to integrate self-improving agents, dynamic workflows for open-ended tasks, unpredictable execution flows, and an ever-expanding library of models and tools. Managing such systems efficiently requires rethinking the entire stack and redefining the roles of its layers.

We present Murakkab, a system that enhances resource efficiency through *fungible workflows* and dynamic scheduling (Figure 2). Key components include: (a) a *declarative workflow programming model* that abstracts model, tool, and hardware choices, simplifying development and enabling flexibility, and (b) an *adaptive runtime system* that integrates workflow orchestration and cluster management for resource-aware scheduling and proactive resource management. Preliminary evaluation of Murakkab shows speedups of ∼ 3.4× in workflow completion times with ∼ 4.5× higher energy efficiency.

## 2 TODAY'S IMPERATIVE WORKFLOWS

Workflows in Compound AI Systems today are typically expressed through imperative programs that contain: (1) *the system flow* specifying the components and their interaction, (2) *model types and configuration details* to implement each component and any model/tool specific parameters, (3) *resources for each component* in terms of hardware configuration, and (4) *pricing tiers* in terms of performance guarantees (*e.g.,* token generation throuhgput etc.)

Listing 1 shows such an example of a Video Understanding workflow, based on OmAgent [43]. It defines components (lines 2 to 8) to perform various tasks and their execution flow (line 12). For example, it has a frame extraction (`frame_ext`) component that uses OpenCV [9] and has task-specific parameters like sampling rate. For audio processing, it has a speech-to-text transcription agent (`stt`) implemented using Whisper [33]. It uses an LLM, in this case Llama [18], for summarizing scenes and specifies a context length. For each of these components, there is either a hardware configuration (*e.g.,* 1 NVIDIA H100 [12] GPU) or pricing tier (*e.g.,* 4 Provisioned Throughput Units or PTUs [6]).

This workflow tightly integrates the application logic (*e.g.,* "List objects shown/mentioned in the videos"), the specific models to use (*e.g.,* Llama), and the resources to allocate (*e.g.,* GPUs: 1 or PTUs: 4). In addition to developing application logic, the developer has added burden to configure many hyperparameters and resource specifications. Often, these selections are suboptimal—there could either be resource stranding or underprovisionig leading to suboptimal performance. As a result we end up in a situation similar to Figure 1, with rigid cross-layer coupling, that makes it challenging to improve efficiency of such systems.

## 3 EFFICIENCY THROUGH FUNGIBILITY

Figure 2 shows our vision for Compound AI Systems, where application logic is decoupled from execution details. Developers focus on application logic, without managing model selection, updates, or resource demands. The runtime dynamically generates task graphs from high-level descriptions, mapping tasks to models and tools for efficient resource multiplexing while maintaining quality. It leverages application fungibility to optimize for efficiency at runtime [36].

We draw inspiration from the evolution of SQL and the query optimizations that are enabled by its declarative nature [10, 20]. Recent work has taken initial steps in exploring this for AI systems [3, 24, 26], albeit for narrow use cases and with limited focus on resource efficiency and multi-tenancy. We aim to bridge this gap with Murakkab.

```
# Define the components (models/tools), hyperparameters, resource specifications and pricing tiers
frame_ext = Tool(name="OpenCV", params={sampling_rate: 15}, key=ON_PREM_SSH_KEY, resources={CPUs: 1})
stt       = MLModel(name="Whisper", key=OPENAI_API_KEY, resources={PTUs: 1})
obj_det   = MLModel(name="CLIP", key=AWS_SSH_KEY, resources={CPUs: 2})
summarize = LLM(name="llama", key=DATABRICKS_API_KEY, params={context_len: 4096},
                resources={GPUs: 1, GPU_Type: H100},
                system_prompt="You are an agent that can describe images in detail.",
                user_prompt="Summarize the scenes using frames, detected objects and transcripts.")
# Inputs
videos = ["cats.mov", "formula_1.mov"]
# Describe the data flow between components
result = Workflow(frame_ext -> stt -> obj_det -> summarize).execute()
```

**Listing 1: Video Understanding workflow defined for today's Compound AI Systems. This requires explicit selection of models/tools and details of the providers offering them (*i.e.,* API keys). For clarity, we show the resource configuration for each agent in-line, although these details are typically specified when signing up with agent providers and/or cloud platforms.**

```
1  # Define the job in natural language
2  desc ="List objects shown/mentioned in the videos"
3  # Optional: Specify sub-tasks in the job
4  t1 = "Extract frames from each video"
5  t2 = "Run speech-to-text on all scenes"
6  t3 = "Detect objects in the frames"
7  # Inputs
8  videos = ["cats.mov", "formula_1.mov"]
9  # Execute
10 result = Job(description=desc, inputs=videos,
11              tasks=[t1, t2, t3],
12              constraints=MIN_COST).execute()
```

**Listing 2: Video Understanding workflow defined for Murakkab. This only requires declaring a high-level job description and optional hints/constraints.**

### 3.1 Declarative Workflow Programming

Murakkab promotes high-level declarative workflow specifications for two reasons: (1) it frees developers from managing low-level implementation details, and (2) it enhances workflow flexibility by allowing dynamic selection of models, tools, and resources at runtime to improve efficiency.

Listing 2 shows the same Video Understanding workflow defined for Murakkab. The programmer provides a natural language description of the job (`desc`) and the required inputs for the job (`inputs`). The programmer may optionally assist the system by specifying sub-tasks (lines 4 to 6) that need to be performed to accomplish the job. However, if these tasks are not provided or are insufficient, an orchestrator LLM decomposes the job into smaller tasks based on the provided job description. It also identifies the relationship between tasks and generates the corresponding internal representation as a directed acyclic graph (DAG) where the nodes represent agents, and edges represent dataflow between them. Moreover, the programmer can also specify high-level constraints for performance or quality (*e.g.,* `MIN_COST` would let the system decide an execution strategy that minimizes execution cost of the workflow, potentially in exchange for latency.) In the future, we plan to support multiple constraints with a priority ordering.

Note that specific models, tools, and hardware resources are abstracted from the developer while allowing Murakkab to dynamically select them at runtime.

### 3.2 Adaptive Runtime Scheduling

Table 1 outlines the parameters Murakkab adjusts to optimize monetary cost, power consumption, and execution latency while measuring their impact on workflow result quality. Below, we describe Murakkab's adaptive workflow execution and how these parameters guide scheduling decisions.

**Job Decomposition.** Using a declarative programming model requires *lowering* the high-level job specification into actionable tasks. For dynamic workflows, Murakkab uses LLMs to decompose a job description into a set of tasks, following the ReAct [41] approach. The LLM orchestrates the execution order of tasks in the workflow and outputs a DAG.

**Task-to-Agent Mapping.** Murakkab maintains a flexible library of agents, detailing their names, functionalities, and schemas (*e.g.,* function arguments). The orchestrator uses this library and task descriptions to map tasks to suitable agents. For example, with NVLM [13] as the orchestrator LLM, Murakkab provides the agent library via the *system prompt* [30] and task descriptions via the *user prompt*. This enables the LLM to assign agents to tasks. Murakkab then supplies task metadata and input details to the LLM, requesting a *tool call* [29] for the selected agent. The LLM generates an executable code snippet with the necessary arguments to invoke the agent directly. For example, given the task `"Extract frames from each video"` and appropriate metadata, the LLM may generate the following tool call: `FrameExtractor(start_time=0, end_time=60s, num_frames=10, file="cats.mov")`.

**Model/Tool Selection.** The library may contain multiple models or tools that support a given agent interface. For instance, the `Speech-to-Text` agent can be implemented using Whisper, DeepSpeech, Fast Conformer [34], and others.

| Parameter | Category | Selection | $ Cost | Power | Latency | Quality |
|---|---|---|---|---|---|---|
| GPU Generation | Hardware Type | Newer | Higher | Higher | Lower/No Change | No Change |
| CPU vs GPU | Hardware Type | CPU | Lower | Lower | Lower | No Change |
| Task Parallelism | Resource Amount | More Fan Out | Higher | Higher | Lower | No Change |
| Execution Paths | Resource Amount | More Paths | Higher | Higher | Higher/No Change | Higher/No Change |
| Model/Tool | Agent Implementation | More Parameters | Higher | Higher | Higher | Higher/No Change |

**Table 1: Optimization parameters and their impact on efficiency and quality. For simplicity we show a single selection.**

Each differs in response quality, performance and resource requirements. Murakkab generates an *execution profile* for each model/tool and hardware resource pair when a new one is added to the library—the profile captures an efficiency vs quality tradeoff. Efficiency metrics include cost, power consumption, and latency. At runtime, Murakkab selects the model/tool and resources that maximize efficiency while meeting the target quality.

**Resource Allocation.** Cloud platforms provide various hardware SKUs, including different GPU and CPU types with dynamic availability (*e.g.,* Spot VMs [7], Harvest VMs [2]). Models and tools can run on a range of these hardware types. For instance, some models perform better on newer GPUs (*e.g.,* NVIDIA H100 [12]) with higher FLOPS, while others see no significant benefit. Similarly, certain models run efficiently on CPUs, whereas others may be too slow to execute practically. Murakkab dynamically allocates resources to the models and tools by using their execution profiles and real-time resource availability metrics from the Cluster Manager.

**Execution Paths.** The Orchestrator leverages parallelism within the workflow, assigning additional resources to agents for improved performance by breaking tasks into sub-tasks and executing them in parallel. For example, `FrameExtractor` can split a video into smaller chunks for parallel extraction when resources are available. In some cases, the Orchestrator may explore multiple execution paths in parallel to enhance result quality. For example, in Chain-of-Thought [40] workflows, allocating more resources allows exploration of additional reasoning paths, with the final result determined by top-k outputs. These decisions are based on cost constraints and real-time resource availability.

**Workflow-Aware Cluster Management.** Typically used cluster management systems (*e.g.,* Kubernetes [4] for microservices) are not well-suited for Compound AI Systems as they lack the necessary insights for workflow scheduling and orchestration, which are integral to such systems (Figure 1). Murakkab bridges this gap by facilitating information exchange between workflow scheduling and cluster management (Figure 2). It exposes workflow DAGs to the Cluster Manager, providing visibility into completed and upcoming tasks. This enables the Cluster Manager to rebalance resources across models and tools more effectively. With this enhanced visibility, the Cluster Manager can make better scaling and resource allocation decisions. For example, if no workflows are expected to require a `Speech-To-Text` agent soon, it can reallocate GPU resources from Whisper to Llama in anticipation of increased demand.

We wish to incorporate learning from prior cluster management research [15, 16, 35] to efficiently use heterogeneous hardware, offer QoS and perform online scheduling.

**Resource-Aware Worfklow Orchestration.** The Workflow Orchestrator continuously receives stats from the Cluster Manager including idle resources, per-model or tool resource consumption and any harvestable resources like Spot Instances [2, 7]. The Orchestrator prefers selecting models/-tools that are already running or for which there are enough resources available to handle incoming requests. Resource efficiency is improved by maximizing resource multiplexing and minimizing fragmentation.

Thus, integrating the Workflow Orchestrator and Cluster Manager is crucial to unlocking efficient Compound AI Systems. Murakkab leverages this design to the fullest.

### 3.3 Murakkab Overheads

Murakkab has several overheads associated with it. (a) *Profiling*: To be able to offer different resource configurations, we need to profile the agents and tools on different hardware and configurations. However, this profiling is amortized over the lifetime of all the workflows that use a particular agent or tool. (b) *DAG Creation*: Task understanding from a natural language prompt, and DAG creation requires LLM queries. However, these are short input and short output queries, that take less than 1% of the execution time of the target AI workflows. (c) *Configuration Search*: The search space across the levers mentioned in Table 1 can easily explode. Therefore, we are working on strategies to prune the space with greedy search using hierarchy of optimization functions.

## 4 EVALUATION

Our evaluation examines whether Murakkab can take advantage of the fungibility of the declarative workflow to identify different execution configurations using the levers in Table 1 and make a selection based on their efficiency/performance trade-off. We run the Video Understanding workflow derived from OmAgent [43] as shown in Listing 1 as the baseline and Listing 2 on Murakkab. The execution output and accuracy are the same in all comparisons.

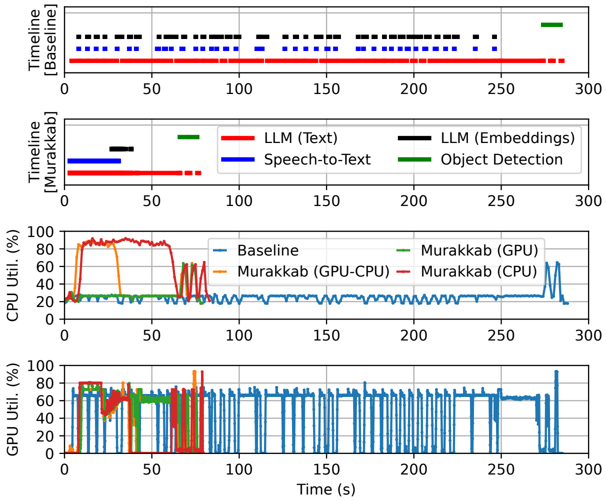

**Figure 3: Execution traces of the Video Understanding workflow. Murakkab can adjust between multiple configurations and deliver a** ∼ 3.4× **speedup with higher resource efficiency.**

**Setup.** We run our experiments on two Azure VMs (Standard_ND96amsr_A100_v4 [8]) each with 96 AMD EPYC 7V12 vCPUs and 8 NVIDIA A100 (80GB) GPUs [28]. We use an OpenCV [9]-based frame extractor (CPUs), NVLM [13] as the LLM for frame summarization (8 GPUs for text completion and 2 GPUs for embeddings to insert in a VectorDB for question/answering), CLIP [32] for Object Detection (CPUs) and Whisper [33] for Speech-to-Text transcription (1 GPU).

**Baseline.** Derived from OmAgent [43], the baseline workflow specifies a fixed execution without any intra-task parallelism or opportunity to utilize idle resources. Each scene and its constituent frames are processed sequentially. Figure 3 shows the results—this workflow completes in 283*s* and severely underutilizes resources.

**Murakkab.** We use NVLM as the orchestrator, and generate a DAG for the workflow to identify parallelism opportunities. Speech-to-Text (STT) is identified as the main dependency for the later stages. Based on STT's internal tasks and DAG, Murakkab is able to perform three optimizations. It: (a) executes STT transcription for multiple scenes in parallel (leveraging dataflow structure from the DAG), (b) parallelizes intra-scene frame summarization (leveraging underutilized GPUs), and (c) adjusts the resource configuration for STT (using execution profiles for Whisper). We show execution traces from the various resource configurations that Murakkab can choose for STT—1 GPU (similar to the baseline), 64 CPU cores, and a combination of GPU and CPUs. Figure 3 shows the results—Murakkab can complete the workflow between 77–83*s*, a ∼ 3.4× speedup over the baseline.

Table 2 shows the energy consumption and execution times of each configuration. For simplicity we only measure the GPU energy consumption since that is the dominant source in the system (rated 16× higher than the CPU power). Executing STT entirely on CPUs results in the lowest energy consumption (34$Wh$), while executing it entirely on GPUs results in the fastest execution (77$s$). Murakkab selects the CPU configuration to satisfy the `MIN_COST` constraint (Listing 2), resulting in ∼ 4.5× higher energy efficiency.

| Speech-to-Text Config. | Energy (Wh) | Time (s) |
|---|---|---|
| Baseline | 155 | 285 |
| Murakkab CPU | 34 | 83 |
| Murakkab GPU | 43 | 77 |
| Murakkab GPU + CPU | 42 | 77 |

**Table 2: Energy and execution time of each configuration.**

## 5 DISCUSSION

**AI Workflows-as-a-Service (AIWaaS).** We envision a future for Compound AI Systems where developers focus solely on application logic, without needing to manage model or resource details. Similar to Functions-as-a-Service (FaaS) [21], where the runtime system handles resource allocation and load balancing, we propose an AI Workflows-as-a-Service (AIWaaS) model with similar capabilities. This can improve efficiency, lower operational costs, and make AI systems more accessible and easier to maintain. Applications will not need rewriting (*e.g.,* prompt engineering, workflow recreation) when new models or tools are available—the runtime system will transparently adopt newer implementations and resources as needed.

**Quantifying and Controlling Quality.** Cost-quality tradeoffs are well-studied for single-model queries (e.g., FrugalGPT [11]), but end-to-end workflows pose unique challenges. Model interactions cause cascading effects, making it costly and impractical to evaluate all combinations. We explore methods to narrow the search space by identifying stages with the greatest impact on cost and accuracy. Additionally, *hallucinations* in early stages can derail workflows, highlighting the need for more correctness checkpoints and tools for quality control.

**Proprietary Models and Agents.** Integrating agent providers and cloud platforms into a unified entity can improve resource efficiency in Compound AI Systems. However, proprietary models often cannot be exported, requiring external API calls to third-party models. This may reduce resource efficiency due to limited visibility into third-party resource usage. Further research is needed to determine when offloading tasks to third-party providers is more beneficial than using local models/tools, albeit with lower quality.

**Multi-cloud Compound AI Systems.** Greater control over hardware resources is easier when the cloud platform and workflow execution service are managed by the same entity (*e.g.,* Azure ML [5], AWS SageMaker [37]). However, using multiple cloud platforms [39] can reduce costs and offer a wider variety of hardware (*e.g.,* Google TPUs [22]). This is possible if each platform exposes resource utilization metrics, allowing systems like Murakkab to manage resources across clouds and schedule tasks efficiently.

## 6 CONCLUSION

This work highlights inefficiencies in existing Compound AI Systems and identifies key challenges limiting resource optimization. To overcome these, we propose a reimagined architecture featuring fungible workflows, dynamic scheduling, and adaptive resource management. By unifying workflow orchestration with cluster management, our system enhances resource utilization, reduces operational costs, and maintains or improves result quality. Our preliminary evaluations show significant gains in efficiency, validating our approach. Looking ahead, our AIWaaS vision aims to simplify AI application development and make AI systems more accessible and sustainable across diverse use cases.

## ACKNOWLEDGMENTS

This work was funded in part by ACE, one of seven centers in JUMP 2.0, a Semiconductor Research Corporation (SRC) program sponsored by DARPA.

## REFERENCES

[1] Amazon Web Services. 2025. Amazon Web Services (AWS). https://aws.amazon.com/.

[2] Pradeep Ambati, Íñigo Goiri, Felipe Frujeri, Alper Gun, Ke Wang, Brian Dolan, Brian Corell, Sekhar Pasupuleti, Thomas Moscibroda, Sameh Elnikety, et al. 2020. Providing SLOs for Resource-Harvesting VMs in cloud platforms. In *OSDI*.

[3] Eric Anderson, Jonathan Fritz, Austin Lee, Bohou Li, Mark Lindblad, Henry Lindeman, Alex Meyer, Parth Parmar, Tanvi Ranade, Mehul A. Shah, Benjamin Sowell, Dan Tecuci, Vinayak Thapliyal, and Matt Welsh. 2024. The Design of an LLM-powered Unstructured Analytics System. arXiv:2409.00847 [cs.DB] https://arxiv.org/abs/2409.00847

[4] The Kubernetes Authors. 2024. Kubernetes Documentation. https://kubernetes.io/

[5] Microsoft Azure. 2024. Azure Machine Learning. https://azure.microsoft.com/en-us/products/machine-learning

[6] Microsoft Azure. 2024. What is provisioned throughput? https://learn.microsoft.com/en-us/azure/ai-services/openai/concepts/provisioned-throughput

[7] Microsoft Azure. 2024. What is provisioned throughput? https://azure.microsoft.com/en-us/products/virtual-machines/spot/

[8] Microsoft Azure. 2025. GPU Accelerated Virtual Machines: NDm A100 v4-Series. https://learn.microsoft.com/en-us/azure/virtual-machines/sizes/gpu-accelerated/ndma100v4-series.

[9] G. Bradski. 2000. The OpenCV Library. *Dr. Dobb's Journal of Software Tools* (2000).

[10] Surajit Chaudhuri. 1998. An overview of query optimization in relational systems. In *Proceedings of the Seventeenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems* (Seattle, Washington, USA) *(PODS '98)*. Association for Computing Machinery, New York, NY, USA, 34–43. https://doi.org/10.1145/275487.275492

[11] Lingjiao Chen, Matei Zaharia, and James Zou. 2023. FrugalGPT: How to Use Large Language Models While Reducing Cost and Improving Performance. arXiv:2305.05176 [cs.LG] https://arxiv.org/abs/2305.05176

[12] NVIDIA Corporation. 2022. NVIDIA H100 Tensor Core GPU Datasheet. https://resources.nvidia.com/en-us-tensor-core/nvidia-tensor-core-gpu-datasheet

[13] Wenliang Dai, Nayeon Lee, Boxin Wang, Zhuolin Yang, Zihan Liu, Jon Barker, Tuomas Rintamaki, Mohammad Shoeybi, Bryan Catanzaro, and Wei Ping. 2024. NVLM: Open Frontier-Class Multimodal LLMs. arXiv:2409.11402 https://arxiv.org/abs/2409.11402

[14] Databricks. 2025. Databricks Large Language Model Serving. https://docs.databricks.com/en/large-language-models/index.html.

[15] Christina Delimitrou and Christos Kozyrakis. 2013. Paragon: QoS-aware scheduling for heterogeneous datacenters. In *Proceedings of the Eighteenth International Conference on Architectural Support for Programming Languages and Operating Systems* (Houston, Texas, USA) *(ASPLOS '13)*. Association for Computing Machinery, New York, NY, USA, 77–88. https://doi.org/10.1145/2451116.2451125

[16] Christina Delimitrou and Christos Kozyrakis. 2014. Quasar: resource-efficient and QoS-aware cluster management. In *Proceedings of the 19th International Conference on Architectural Support for Programming Languages and Operating Systems* (Salt Lake City, Utah, USA) *(ASPLOS '14)*. Association for Computing Machinery, New York, NY, USA, 127–144. https://doi.org/10.1145/2541940.2541941

[17] Google Cloud. 2025. Google Cloud Platform. https://cloud.google.com/.

[18] Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, et al. 2024. The Llama 3 Herd of Models. arXiv:2407.21783 [cs.CL] https://arxiv.org/abs/2407.21783

[19] Aaron Hurst, Adam Lerer, Adam P. Goucher, Adam Perelman, Aditya Ramesh, et al. 2024. GPT-4o System Card. arXiv:2410.21276 [cs.CL] https://arxiv.org/abs/2410.21276

[20] Matthias Jarke and Jurgen Koch. 1984. Query Optimization in Database Systems. *ACM Comput. Surv.* 16, 2 (June 1984), 111–152. https://doi.org/10.1145/356924.356928

[21] Eric Jonas, Johann Schleier-Smith, Vikram Sreekanti, Chia-Che Tsai, Anurag Khandelwal, Qifan Pu, Vaishaal Shankar, Joao Carreira, Karl Krauth, Neeraja Yadwadkar, et al. 2019. Cloud programming simplified: A berkeley view on serverless computing. *arXiv preprint arXiv:1902.03383* (2019).

[22] Norman P. Jouppi, George Kurian, Sheng Li, Peter Ma, Rahul Nagarajan, Lifeng Nai, Nishant Patil, Suvinay Subramanian, Andy Swing, Brian Towles, Cliff Young, Xiang Zhou, Zongwei Zhou, and David Patterson. 2023. TPU v4: An Optically Reconfigurable Supercomputer for Machine Learning with Hardware Support for Embeddings. arXiv:2304.01433 [cs.AR] https://arxiv.org/abs/2304.01433

[23] LangChain. 2024. LangChain. https://github.com/langchain-ai/langchain.

[24] Chunwei Liu, Matthew Russo, Michael Cafarella, Lei Cao, Peter Baille Chen, Zui Chen, Michael Franklin, Tim Kraska, Samuel Madden, and Gerardo Vitagliano. 2024. A Declarative System for Optimizing AI Workloads. arXiv:2405.14696 [cs.CL]

[25] Jerry Liu. 2022. LlamaIndex. https://github.com/jerryjliu/llama_index.

[26] Samuel Madden, Michael Cafarella, Michael Franklin, and Tim Kraska. 2024. Databases Unbound: Querying All of the World's Bytes with AI. *Proc. VLDB Endow.* 17, 12 (Nov. 2024), 4546–4554. https://doi.org/10.14778/3685800.3685916

[27] Microsoft. 2025. Microsoft Azure. https://azure.microsoft.com/.

[28] NVIDIA. 2024. NVIDIA A100 Tensor Core GPU Datasheet. https://www.nvidia.com/content/dam/en-zz/Solutions/Data-Center/a100/pdf/nvidia-a100-datasheet-us-nvidia-1758950-r4-web.pdf

[29] OpenAI. 2023. Function Calling Guide. https://platform.openai.com/docs/guides/function-calling

[30] OpenAI. 2023. OpenAI API Reference. https://platform.openai.com/docs/api-reference/introduction

[31] OpenAI. 2025. OpenAI Large Language Models and API. https://platform.openai.com/docs/.

[32] Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Saurabh Sastry, Amanda Askell, Pam Mishkin, Jack Clark, Gretchen Krueger, and Ilya Sutskever. 2021. Learning Transferable Visual Models From Natural Language Supervision. https://openai.com/research/clip. OpenAI.

[33] Alec Radford, Jong Wook Kim, Tao Xu, Greg Brockman, Christine McLeavey, and Ilya Sutskever. 2022. Robust Speech Recognition via Large-Scale Weak Supervision. https://openai.com/research/whisper. OpenAI.

[34] Dima Rekesh, Nithin Rao Koluguri, Samuel Kriman, Somshubra Majumdar, Vahid Noroozi, He Huang, Oleksii Hrinchuk, Krishna Puvvada, Ankur Kumar, Jagadeesh Balam, and Boris Ginsburg. 2023. Fast Conformer with Linearly Scalable Attention for Efficient Speech Recognition. arXiv:2305.05084 [eess.AS] https://arxiv.org/abs/2305.05084

[35] Francisco Romero and Christina Delimitrou. 2018. Mage: online and interference-aware scheduling for multi-scale heterogeneous systems. In *Proceedings of the 27th International Conference on Parallel Architectures and Compilation Techniques* (Limassol, Cyprus) *(PACT '18)*. Association for Computing Machinery, New York, NY, USA, Article 19, 13 pages. https://doi.org/10.1145/3243176.3243183

[36] Zhenyuan Ruan, Shihang Li, Kaiyan Fan, Marcos K. Aguilera, Adam Belay, Seo Jin Park, and Malte Schwarzkopf. 2023. Unleashing True Utility Computing with Quicksand. In *Proceedings of the 19th Workshop on Hot Topics in Operating Systems* (Providence, RI, USA) *(HOTOS '23)*. Association for Computing Machinery, New York, NY, USA, 196–205. https://doi.org/10.1145/3593856.3595893

[37] Amazon Web Services. 2024. Amazon SageMaker: Build, Train, and Deploy Machine Learning Models at Scale. https://aws.amazon.com/sagemaker/

[38] Stanford NLP Group. 2023. DSPy: The Framework for Programming—Not Prompting—Language Models. https://github.com/stanfordnlp/dspy.

[39] Ion Stoica and Scott Shenker. 2021. From Cloud Computing to Sky Computing. In *HotOS*. https://doi.org/10.1145/3458336.3465301

[40] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed Chi, Quoc Le, and Denny Zhou. 2023. Chain-of-Thought Prompting Elicits Reasoning in Large Language Models. arXiv:2201.11903 [cs.CL] https://arxiv.org/abs/2201.11903

[41] Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. 2023. ReAct: Synergizing Reasoning and Acting in Language Models. arXiv:2210.03629 [cs.CL] https://arxiv.org/abs/2210.03629

[42] Matei Zaharia, Omar Khattab, Lingjiao Chen, Jared Quincy Davis, Heather Miller, Chris Potts, James Zou, Michael Carbin, Jonathan Frankle, Naveen Rao, and Ali Ghodsi. 2024. The Shift from Models to Compound AI Systems. https://bair.berkeley.edu/blog/2024/02/18/compound-ai-systems/.

[43] Lu Zhang, Tiancheng Zhao, Heting Ying, Yibo Ma, and Kyusong Lee. 2024. OmAgent: A Multi-modal Agent Framework for Complex Video Understanding with Task Divide-and-Conquer. arXiv:2406.16620 [cs.CL] https://arxiv.org/abs/2406.16620