Caltech Center for Technology & Management Education

The New AI Team: A Departure from ML Teams

Unlocking New Possibilities for Innovation and Impact

*Nicholas Beaudoin, Caltech CTME **

October 2024

The ML/AI Shift

Over the past two years, we have observed a significant transformation in companies' deployment of machine learning products. What was once known as machine learning (ML) is now frequently termed artificial intelligence (AI), a shift that has not gone unnoticed by engineering teams. Almost overnight, data scientists and ML engineers updated their LinkedIn profiles to reflect their new titles as "AI engineers."[1] Executives and hiring managers are increasingly grappling with the rapid evolution of AI development, particularly generative AI (gen AI) and the specialized skills it demands. This article aims to unravel these complexities, equipping decision-makers with the necessary insights to make strategic, informed AI hires.

What is the Difference Between ML and AI Teams?

The transition from ML team to AI team has been bubbling under the surface for some time. The release of ChatGPT instigated a coup in budget allocation between traditional ML engagements and investigative AI projects. The impact was immediate and far-reaching. Companies began to run POCs/POVs (proof of concept/value) to demonstrate the feasibility of harnessing pre-trained OpenAI GPT API model calls into their enterprise systems. However, as these POCs and demos progressed, it quickly became apparent that a new set of technical skills was required to bring these models to deployment.

The pivotal moment occurred following Google's 2017 seminal paper, *Attention Is All You Need.* After 2017, large language models became accessible via API calls, enabling the processing of vast amounts of data through transformer models at unprecedented computational speeds. The drive towards APIs as the central point of algorithmic magic was a paradigm shift for ML engineers. Before ChatGPT and Open AI's GPT 3.5 model release, ML engineers were guided by traditional ML practices. These typically involved rigorous data preprocessing, model creation through algorithmic selection, parameter initialization, hyperparameter tuning, and performance evaluation (outside computer vision use cases). After November 2022, this all changed.

Winter 2022 was a curious time for ML teams. Most teams tried to deconstruct the ChatGPT interface for their use cases and make sense of the GPT API calls. Others binged on YouTube tutorials to understand the Transformer model and attention-based modeling approach. Others read new gen AI model development releases on Arxiv.org and played with various LangChain and OpenAI tutorials. Outside the ML community, only some individuals in management took heed of what was about to happen.

Enter the Hype Machine

In the spring of 2023, the marketing machine took over. Companies like Google began to espouse their managed gen AI solutions in Vertex AI. Amazon soon followed suit by implementing its own gen AI management platform, Bedrock. Azure decided to buy their way in with a partnership with Open AI. Suddenly, every company asked themselves one question: "How can I leverage large language models (LLMs) to increase revenue?" This was followed by the familiar chime of consultant pitches offering slide decks of humanoid robots to display what customers wanted to see. Every way you looked, the title "Data Scientist" was replaced with "Gen AI" on LinkedIn.

Behind the scenes, hiring teams scrambled. Strategic AI hires moved budgets away from ML projects into anything that said gen AI. The core focal point

1 This article defines ML teams in the context of traditional ML projects. These may include fraud detection, forecasting, anomaly detection, basic NLP tasks, and derivatives of deep neural networks. However, the scope is more comprehensive than these projects alone. References to transfer learning on an ML team imply downloading the weights of a pre-trained model for a new domain use case. In contrast, the AI team is represented as those who work with pre-trained algorithms via API connections to the hosting platform.

moved from "How do I create an algorithm that can predict this thing?" to "How can I borrow an algorithm that can *generate* this thing?" The paradigm shifted, and AI teams became the champions of executive hiring decision-making.

How Have Things Worked Historically?

So, what is involved in an AI team, and how does it differ from how we used to hire ML teams? As previously stated, the ML team is structured around creating a well-tuned algorithm that can generalize about future scenarios, be it making predictions about revenue, movie recommendations, or whether this image is a hot dog. ML engineering has been the purview of deep expertise in creating an algorithm with a low error rate.

Traditionally, Data Engineers support ML teams and move data from one place to another, ensure its validity and structure, and help maintain a steady flow of ones and zeros into the algorithm. The Data Scientist creates data narratives based on statistical principles to inform the model-building process, engineer new informative features, and work hand-in-hand with the ML engineer. The ML engineer takes those features and builds a generalized model of the specified environment. Finally, the ML Operations (MLOps) Engineer ensures that the model is deployable and scalable and has the necessary pipelines to automate any changes that need to happen in production.[2]

However, not all companies have this level of corporate drive or engineering community cohesion to adopt new technologies. At the same time, many companies don't have the data foundations even to begin looking at AI solutions. The main point is companies are at different levels of an AI/ML maturity curve. Some are merely leveraging Jupyter Notebooks on a local machine. Others are spinning up a managed ML environment. Some are making these managed AI services accessible to the entire team on AWS SageMaker or Google's Vertex AI. Some have a rough implementation of MLOps where proper CI/CD practices are enforced to reduce the clutter around GitHub PRs. Even in the MLOps world, there are various levels of maturity. (Cloud Architecture Center: Automation pipelines in machine learning)

How Have Things Changed?

Modern AI systems, notably gen AI, rely on pre-trained algorithms called "foundation models" accessed via an API. Unless your company has tens of millions of dollars and access to a world-class collection of documents, **cough cough** Bloomberg **cough cough**, you aren't creating an in-house foundation model. This is why the industry has leaned so heavily on API calls to foundation models. However, after the API call, companies are flummoxed by an inability to provide the right tools to integrate the API into a deployable use case.

Scalable ML deployments are centered around the ML engineer and coordination with the MLOps solutions architect (or engineer). In contrast, AI system deployment in computer vision or gen AI focuses on individuals who can deploy and scale an API connection. Rather than building ML algorithms, these new AI teams can focus on building applications and workflows powered by the API. These new AI engineers must know the ins and outs of open-source tools such as LangChain, LlamaIndex, or Semantic Kernel. They are called to orchestrate a dance between prompts, agents, and service calls, wrapping them into a Dockerized package ready for shipment.

The AI engineer's new domain focuses on software engineering, specifically how to integrate various open and closed-source systems. Whereas the ML engineer works with structured data (mostly) and relies on the data engineer to query and pull data, the AI engineer needs to leverage a storage

2 Not all companies have the resources to deploy open-source solutions tailored to their specific use case. In my experience, I have seen companies with data science teams in the double digits effectively coordinate shared features with open-source feature stores, create reproducible model registries while leveraging multiple cloud services, and create adoption frameworks that drew these various data science and ML teams to follow this new tool. Uber's Michelangelo 3.0 is probably the most valiant effort publicly written about on their engineering blog.

space for unstructured data. The AI engineer often pulls this data rather than the Data Engineer in this new AI team environment. Building on this, the AI engineer needs to be proficient in vector databases where retrieval of unstructured data is influenced not by an index but by an algorithm.[3] Again, paradigm shift alert!

What about infrastructure?

The ML team relies on the role of the MLOps engineer to orchestrate and define the pipelining and deployment along DevOps best practices. There has been some overlap in the gen AI world, particularly around LLMs. Whereas the MLOps community has set standards followed by traditional ML deployments, the AI engineering community does not. This practice has numerous names, from LLMOps to GenOps to AIOps. From firm to firm, these structured methods of AI deployment vary and need to be in sync. More often than not, these terms are veneers for a practice that the AI community is still figuring out. While there are core principles of the DevOps community that can be practiced, the paradigm shift from ML to AI teams encapsulates how these systems are deployed.

So, is an AI engineer just a glorified API call machine?

The AI and ML community is familiar with the famous screwups of treating an AI team as a lone software engineer. Chevrolet of Watsonville selling Chevy Tahoes for a dollar or consulting chatbots telling the client what a terrible product they sell is. Many of these are not far from the consequences of Microsoft's 2016 flop in their Tay chatbot release. Proper engineering of the AI system will require a profound understanding of various open-source platforms, guardrails to reduce incorrect or misleading information and manipulation of output, storage mechanisms that can query unstructured data, and means to version control changes to the platform.

What skills do AI teams use?

The software engineer toolkit is the real hero for AI engineers, which is why most AI engineers come from a heavy software engineering background. As discussed, the switch from building ML models to calling an API for a pre-trained model has driven AI teams to adopt software engineering methodologies. Principally, how do you integrate an API into a legacy system? Software engineering methods such as building self-contained, modular code, creating unit and integration tests, and abstraction to hide complexity from the user are all in demand when building AI systems. This shift is evident based on hiring posts requiring a system integration background and how to use open-source tools such as LangChain and LlamaIndex to abstract complex prompts and agent calls.

Approximately 80 percent of industry gen AI use cases rely on semantic search, which involves searching an extensive collection of documents. A database system that can handle unstructured data is needed to support this. Not only that, but they need a database that can use algorithmic search methods to find documents efficiently. The ability to use vector databases to search this semantic space makes open-source tools such as Meta's FAISS, ChromaDB, Pinecone, and Weaviate so powerful.[4] On the other hand, there are proprietary systems such as Postgres pgvector, Amazon's Kendra, and whatever Google has renamed Vertex AI's Matching Engine.

Since software engineering tries to abstract much of a system's complexity out of a user's hands, AI engineers try to copy this using tools like open-source LangChain, LlamaIndex, or proprietary Microsoft Semantic Kernel. These tools allow AI engineers to use prompt templating to obfuscate the cumbersome prompting away from the user, enforce guardrails on the post-processing of the API model calls, use agents to create tasks to communicate with other systems and leverage built-in frameworks that help with document ingestion.

As we move to 2025, the AI engineer toolkit is still changing. While many companies have implemented internal products leveraging gen AI (e.g. document Q/A, chatbots, etc), few have deployed these to their external users at scale.

This is primarily limited by a lack of trust by the company in the model's output. Having a grocery store chatbot release medical information when being asked about a recipe involving a hot stove is just too big of a risk for many companies to swallow.

To combat this, many AI engineering teams are hiring AI engineers with a background in model evaluation. AI engineers have found that relying on foundation model evaluation benchmarks released by Open AI, Anthropic, AI21, Meta, etc, does little once that data is "trained" on their internal documents. In fact, most evaluation benchmarks go out the window when tailored to a company's data.

So, what does the future outlook look like?
As the initial hype surrounding gen AI subsides, AI teams will increasingly prioritize model evaluation, validation, and product management over simple use case identification. Many companies have developed extensive lists of gen AI use cases; those efforts will only be worthwhile if pushed to a deployable infrastructure. Rigorous model evaluation will be essential to earn client trust to integrate gen AI systems into byzantine enterprise infrastructure (medical providers, hospitals, law firms, etc). Proper channels and orchestration of the AI engineering team will rely on product managers who live and breathe the AI engineer toolkit. 2025 will be an exciting time as companies face board resistance to new gen AI initiatives, demanding a return on the initial investment many of their vendors have promised. Given these challenges, the latest AI team will further separate from their legacy ML teams. •

ABOUT CALTECH CTME
Caltech's Center for Technology and Management Education (CTME) partners with technology-driven organizations to address their most critical challenges and harness their most significant opportunities through executive education and professional development. As part of the California Institute of Technology, CTME combines the power of world-class research with applied learning to develop custom professional development programs that drive innovation and leadership at the intersection of technology and business.

Our programs are designed to equip professionals with cutting-edge skills and a forward-thinking mindset, empowering them to solve today's problems and define the future. CTME's global team of expert instructors and industry practitioners deliver a uniquely collaborative approach to education, leveraging Caltech's pioneering spirit to help clients transform their organizations and achieve lasting competitive advantage.

CTME offers a broad portfolio of professional education, including leadership development, digital transformation, systems engineering, and data science. Our customizable programs ensure that learning is not only highly relevant and impactful but also accessible and scalable for organizations around the world.

* *Special thanks to Scott Tarlow and Kevin Coyle, contributing editors to this article.*

3 Most vector databases use an algorithm in two places: before ingestion and for semantic retrieval. To convert text or images into searchable storage, they need to go through an embedding process. This is where an algorithm (shallow neural network based such as Word2Vec) converts inputs to an array of numbers. These numbers are then ingested into the vector database. When a query is issued, an algorithm (ie, ANNOY) searches the database to retrieve the embeddings.

4 The vector database is where the AI team will be pointing their queries. CV and Gen AI require algorithms to assist in storing unstructured data (text, images, sound, videos) into an n-dimensional space that can hold data as a representation rather than an index, as we see in structured data storage. These representations are created using algorithms called embeddings, which use a shallow neural network to encode and map data to vector space. Once in vector space, these embeddings represent the location of the data. At the moment of query, the AI engineer can create a search process, typically with a lightweight algorithm or trigonometric principles like Cosine Similarity, to retrieve the embeddings and associate it with its true data. Open-source tools like ChromasDB, Pinecone, and FAISS give AI engineers ample resources to experiment and build.