



Comprehensive Summary and Notes on Google Whitepapers


by: Erwin R. Pasia



Google x kaggle

5-Day Gen AI Intensive with Google

Monday, March 31 - Friday, April 4
2025



5-Day Generative AI Intensive with Google 2025

Table of Contents

Day 1: Foundational LLMs and Text Generation

- Deep Dive into Large Language Models (LLMs) and Text Generation
- Introduction
- Transformer Architecture Foundation
- Input Processing for Transformers
- Self-Attention Mechanism
- Multi-Head Attention
- Layer Normalization & Residual Connections
- Feed-Forward Network Layer
- Decoder-Only Architecture
- Mixture of Experts (MoE)
- Evolution of LLMs - Timeline & Key Models
- LLM Training Overview
- Fine-tuning Techniques
- Parameter-Efficient Fine-Tuning (PEFT)
- Prompt Engineering
- Sampling Techniques
- Evaluating LLMs
- Inference Acceleration
- Applications of LLMs

- Conclusion and Future

Day 1: Prompt Engineering Techniques

- Prompt Engineering Techniques for Large Language Models
- Introduction to Prompt Engineering
- Configuring Model Output
- Prompt Engineering Techniques
- Advanced Reasoning Techniques
- Code Prompting Applications
- Best Practices for Prompt Engineering
- Conclusion and Future Considerations

Day 2: Embeddings and Vector Stores

- Introduction to Embeddings and Vector Stores
- Importance of Embeddings
- Applications of Embeddings
- Evaluating Embedding Effectiveness
- Retrieval Augmented Generation (RAG)
- Types of Embeddings
- Training Embedding Models
- Vector Search
- Vector Databases
- Applications of Embeddings and Vector Stores

Day 3: Agents

- Generative AI Agents: A Deep Dive
- Introduction

- What is a Generative AI Agent?
- Cognitive Architecture of Agents
- Agents vs. Models
- How Agents Operate: Cognitive Architectures
- Reasoning and Planning Frameworks
- ReAct in Practice: Example
- Tools: Keys to the Outside World
- Retrieval Augmented Generation (RAG)
- Tools Recap
- Enhancing Model Performance
- Agent Quick Start with LangChain
- Vertex AI Agents
- Main Takeaways

Day 3: Agents Companion

- AI Agents White Paper Companion: A Deep Dive
- Introduction to Generative AI Agents
- Advanced Agent Concepts for Developers
- Agent Ops: Tailored DevOps for AI Agents
- Measuring Success with Business KPIs
- The Importance of Human Feedback
- Automated Evaluation
- Analyzing Agent Trajectory
- Techniques for Measuring Trajectory
- Judging the Quality of Final Response

- The Role of Humans in Evaluation
- Multi-Agent Systems: Divide and Conquer
- Benefits of Multi-Agent Systems
- Structuring Multi-Agent Systems: Design Patterns
- Challenges in Building Multi-Agent Systems
- Evaluating Multi-Agent Systems
- Agentic RAG: Retrieval Augmented Generation with Agents
- Optimizing Basic Search Engine
- Google Cloud Tools for Search
- Real-World Example: Google's Co-scientist
- Automotive AI: Multi-Agent Systems in Cars
- Patterns in Automotive AI
- Benefits of Multi-Agent Approach in Automotive AI
- Agent Builder: Google Cloud's Toolkit for Agent Developers
- Agents as Contractors: A Conceptual Shift
- Agentic RAG Deep Dive
- Optimizing Search for Agentic RAG
- Conceptual Shifts: Trust and Reliability
- Transparency
- Accountability
- Robustness
- Fairness

Day 4: Solving Domain-Specific Problems Using LLMs: Cybersecurity and Medicine

- Introduction to LLMs and Domain Specialization

- Cybersecurity Challenges and SecLM
- Pressures in Cybersecurity
- LLMs as AI Assistants in Cybersecurity
- Layered Approach in Cybersecurity
- SecLM as a Central Resource
- Standards for SecLM
- Reasons General-Purpose LLMs Fall Short
- Creating Specialized SecLMs: Targeted Training Approach
- Evaluating Performance of Specialized Models
- Techniques to Help Models
- Flexible Planning and Reasoning Framework
- SecLM Applications
- Ultimate Goal for SecLM
- Healthcare and Med-PaLM
- Potential Uses of GenAI in Healthcare
- Responsible Innovation in Medicine
- Shift in Scientific Approach
- Med-PaLM Progress
- Measuring AI's Medical Knowledge: Evaluation Strategy
- Human Evaluations
- Areas for Improvement
- Task-Specific vs. Broad Domain Models
- Applications Beyond Patient Care
- Med-PaLM as a Suite of Commercially Available Models
- Med-PaLM 2 Training

- Conclusion

Day 5: Operationalizing Generative AI on Vertex AI using MLOps

- Introduction
- Foundation Models vs. Traditional ML Models
- The Generative AI Lifecycle
- Discovery Phase
- Development and Experimentation Phase
- Data Practices for Generative AI
- Evaluation
- Deployment
- Monitoring and Logging
- Governance
- Agent Ops: The Next Frontier
- The Changing MLOps Landscape
- Conclusion

Day 1: Foundational LLMs and Text Generation

Deep Dive into Large Language Models (LLMs) and Text Generation

Introduction

Large Language Models (LLMs) represent a significant transformation in artificial intelligence, fundamentally altering how we interact with information and technology. These advanced AI systems, typically implemented as deep neural networks, excel at processing, understanding, and generating text that closely resembles human language. LLMs are trained on vast datasets of text, enabling them to discern complex language patterns and perform diverse tasks such as machine translation, creative writing, question answering, text summarization, and reasoning. This chapter delves into the architectural history of LLMs, fine-tuning methodologies, efficient training techniques, inference acceleration strategies, diverse applications, and illustrative code examples, providing a comprehensive understanding up to early 2025.

Transformer Architecture Foundation

The cornerstone of modern LLMs, the Transformer architecture, emerged from a Google translation project in 2017. Its original design featured an encoder-decoder structure. The encoder processed the input text (e.g., a sentence in French) to create a meaningful representation, which the decoder then used to generate the output (e.g., the English translation) sequentially, token by token. A token can represent a complete word (like "cat") or a sub-word unit (like "pre" in "prefix").

Input Processing for Transformers

Before processing, input text undergoes tokenization, where it is broken down into tokens based on a predefined vocabulary. Each token is then converted into a dense vector, known as an embedding, which captures its semantic meaning. Since Transformers process tokens simultaneously rather than sequentially, positional encoding is added to the embeddings. This mechanism, which can be sinusoidal or learned, preserves the original sequence information of the tokens. The choice of positional encoding can influence the model's ability to comprehend longer sentences effectively.

Self-Attention Mechanism

The self-attention mechanism is a critical component that allows the Transformer model to weigh the importance of different words within a sentence when processing a specific word. It enables the model to understand contextual relationships, such as identifying the antecedent of a pronoun (e.g., recognizing that "it" refers to "tiger" in "The thirsty tiger drank because it was hot").

This mechanism operates using three types of vectors derived from each token's embedding:

- **Query (Q):** Represents the current word seeking relevant context.
- **Key (K):** Acts as a label for each word, indicating its semantic content.
- **Value (V):** Contains the actual information or meaning the word carries.

The process involves calculating scores based on the similarity between the Query vector of one word and the Key vectors of all other words in the sentence. These scores are normalized into attention weights, signifying how much focus the current word should place on other words. Finally, a weighted sum of all Value vectors, guided by the attention weights, produces a contextually rich representation for each word. This entire comparison and calculation process is efficiently parallelized using matrix operations for Q, K, and V. The ability to simultaneously process these relationships across the entire input sequence is a key factor in the Transformer's success in capturing nuanced meaning, particularly over long distances within the text.

Multi-Head Attention

To further enhance the model's understanding, multi-head attention executes the self-attention process multiple times in parallel, each time using different, learned Q, K, and V matrices. Each "head" can focus on distinct types of relationships within the text, such as grammatical dependencies or semantic similarities. By integrating the perspectives from these multiple heads, the model achieves a more comprehensive and deeper understanding of the input text.

Layer Normalization & Residual Connections

Training deep neural networks like Transformers can be challenging. Layer normalization helps stabilize the learning process by normalizing the inputs across the features for each layer, leading to faster training and improved performance. Residual connections, or skip connections, provide another crucial stabilization technique. They allow the original input of a layer to bypass the layer's transformations and be added directly to its output. This helps mitigate the vanishing gradient problem, enabling the network to train deeper architectures effectively and retain information learned in earlier layers.

Feed-Forward Network Layer

Following the attention layers, each token's representation is processed independently by a feed-forward network. This network typically consists of two linear transformations separated by a non-linear activation function, such as ReLU (Rectified Linear Unit) or GeLU (Gaussian Error Linear Unit). This component further refines the token representations and enhances the model's overall capacity to learn complex patterns.

Decoder-Only Architecture

While the original Transformer had both an encoder and a decoder, many modern LLMs, particularly those focused on text generation tasks like writing articles or engaging in conversation, employ a decoder-only architecture. These models utilize masked self-attention, where each token can only attend to preceding tokens in the sequence. This ensures that the model generates text sequentially, predicting the next token based solely on the context that comes before it. This simpler design is highly effective for generative tasks as it focuses directly on producing coherent and contextually appropriate output token by token.

Mixture of Experts (MoE)

The Mixture of Experts (MoE) architecture offers a strategy for building significantly larger and more capable models while managing computational costs efficiently. Instead of having one massive, dense network, an MoE model comprises multiple smaller, specialized "expert" submodels. A "gating network" directs the input tokens to only a relevant subset of these

experts for processing. This selective activation means that only a fraction of the model's total parameters are used for any given input, drastically reducing the computational load during both training and inference compared to a dense model of equivalent size.

Evolution of LLMs - Timeline & Key Models

The field of LLMs has evolved rapidly since the introduction of the Transformer:

- **GPT-1 (2018):** A decoder-only model pioneering unsupervised pre-training but sometimes producing repetitive text.
- **BERT (2018):** An encoder-only model focused on language understanding, trained using masked language modeling and next sentence prediction.
- **GPT-2 (2019):** A scaled-up version of GPT-1 demonstrating improved coherence and early zero-shot learning capabilities (performing tasks without specific examples).
- **GPT-3 Family (2020+):** Marked by massive scale (billions of parameters), few-shot learning (learning from a small number of examples), instruction tuning (InstructGPT), code generation (GPT-3.5), multimodality (GPT-4 handling images and text), and large context windows.
- **Lambda (2021):** Focused on improving natural conversation capabilities.
- **Gopher (2021):** Emphasized the importance of high-quality data and optimization, highlighting that simply increasing model size doesn't guarantee better performance across all tasks.
- **GLAM:** Utilized the Mixture of Experts (MoE) architecture.
- **Chinchilla (2022):** Demonstrated the concept of compute-optimal scaling, showing the crucial relationship between model size and the amount of training data.
- **PaLM & PaLM 2 (2022/2023):** Achieved strong benchmark performance, featured the Pathway system, and showed improved reasoning, coding, and mathematical abilities.
- **Gemini:** Natively multimodal, optimized for TPUs, utilized MoE, offered in various sizes (Ultra, Pro, Nano, Flash), and featured very large context windows (1.5 Pro).

- **Open Source Models:** A growing ecosystem including Gemma/Gemma 2, the Llama family (1, 2, 3, 3.1), Mixtral (MoE), O1 (reasoning-focused), DeepSeek-R1 (reasoning), Qwen, Yi, and Grok, each with varying capabilities and licensing considerations.

LLM Training Overview

Training LLMs typically involves two main stages:

- **Pre-training:** An unsupervised phase where the model learns general language patterns, grammar, and world knowledge by processing massive amounts of unlabeled text data from the internet and digitized books.
- **Fine-tuning:** A subsequent phase where the pre-trained model is further trained on smaller, more specific datasets, often labeled, to adapt its capabilities for particular tasks or domains (e.g., medical text analysis, customer service).

Fine-tuning Techniques

Several techniques exist to adapt pre-trained models:

- **Supervised Fine-Tuning (SFT):** Training the model on curated datasets consisting of prompt-response pairs relevant to the target task.
- **Reinforcement Learning from Human Feedback (RLHF):** A multi-step process involving training a separate "reward model" based on human preferences for different model outputs, and then using reinforcement learning algorithms to optimize the LLM to generate outputs that maximize the reward score, aligning it better with human expectations. Related techniques like RLAI (RL from AI Feedback) and DPO (Direct Preference Optimization) offer alternative approaches to alignment.

Parameter-Efficient Fine-Tuning (PEFT)

Training all the parameters of extremely large LLMs can be computationally expensive. PEFT methods address this by fine-tuning only a small subset of the model's parameters while keeping the majority of the pre-trained weights frozen. This significantly reduces the computational resources and time required for adaptation. Common PEFT techniques include:

- **Adapters:** Inserting small, trainable modules between existing layers of the frozen pre-trained model.
- **LoRA (Low-Rank Adaptation):** Introducing trainable low-rank matrices into layers to approximate the weight updates.
- **QLoRA:** A memory-efficient version of LoRA that uses quantization.
- **Soft Prompting (Prompt Tuning):** Keeping the model weights frozen and instead learning specific "soft prompt" embeddings that are prepended to the input sequence to guide the model's behavior.

Prompt Engineering

Prompt engineering is the practice of carefully designing the input text (the prompt) given to an LLM to elicit the desired output. The way a prompt is phrased significantly influences the model's response. Key techniques include:

- **Zero-shot prompting:** Providing a direct instruction or question without any examples.
- **Few-shot prompting:** Including a small number of examples (input-output pairs) within the prompt to demonstrate the desired task format or style.
- **Chain-of-Thought (CoT) prompting:** Structuring the prompt to encourage the model to break down a problem into intermediate reasoning steps before providing the final answer, often improving performance on complex reasoning tasks.

Sampling Techniques

LLMs generate text token by token, predicting the probability distribution for the next token at each step. Sampling techniques determine how the next token is selected from this distribution:

- **Greedy Search:** Always selects the single most probable next token. It's fast but can lead to repetitive or deterministic output.
- **Random Sampling (with Temperature):** Introduces randomness by sampling from the probability distribution. A "temperature" parameter controls the degree of randomness:

higher temperatures increase creativity but also the risk of nonsensical output, while lower temperatures make the output more focused and predictable.

- **Top-K Sampling:** Restricts the selection to the K most probable tokens, reducing the chance of picking very unlikely tokens.
- **Top-P (Nucleus) Sampling:** Selects from the smallest set of tokens whose cumulative probability exceeds a threshold P. This provides a dynamic alternative to Top-K.
- **Best-of-N Sampling:** Generates N candidate responses and uses a scoring mechanism (potentially another model or specific criteria) to select the best one.

Evaluating LLMs

Evaluating the performance of LLMs is complex due to the subjective nature of language and the wide range of tasks they can perform. Traditional NLP metrics are often insufficient. A comprehensive evaluation framework considers:

- **Task-specific data:** Using datasets that reflect real-world usage scenarios and user interactions relevant to the intended application.
- **System-level consideration:** Evaluating the LLM as part of the larger system it integrates with (e.g., in a Retrieval Augmented Generation system).
- **Defining "good":** Establishing clear metrics based on desired qualities like accuracy, helpfulness, factual correctness, coherence, fluency, creativity, and adherence to specific styles or constraints.

Methods for evaluation include:

- **Quantitative metrics:** Using metrics like BLEU and ROUGE (often used in translation and summarization) to compare model output against reference "ground truth" answers, although these may not fully capture quality.

- **Human evaluation:** Employing human raters to provide nuanced judgments on aspects like fluency, coherence, relevance, and overall quality, often considered the gold standard but expensive and time-consuming.
- **LLM-powered evaluators (auto-evaluators):** Using another capable LLM to assess the output of the model being evaluated, based on predefined criteria or rubrics. This approach requires careful calibration and validation.
- **Advanced approaches:** Breaking down complex tasks into subtasks and using detailed rubrics with multiple criteria, especially crucial for multimodal models handling diverse data types.

Inference Acceleration

Making LLM inference (the process of generating responses) faster and more computationally efficient is critical for real-world deployment. This involves balancing trade-offs between response quality, speed (latency), cost, and the number of requests handled concurrently (throughput). Techniques include:

- **Output Approximating Methods:**
 - o **Quantization:** Reducing the numerical precision of the model's weights and activations (e.g., from 32-bit floats to 8-bit integers), decreasing memory usage and potentially speeding up computation, sometimes with a minor impact on quality.
 - o **Distillation:** Training a smaller "student" model to mimic the behavior of a larger, more capable "teacher" model.
- **Output Preserving Methods:**
 - o **FlashAttention:** An optimized implementation of the attention mechanism that reduces memory reads/writes, speeding up calculations.
 - o **Prefix Caching (KV Caching):** Reusing the computed Key and Value vectors for the initial part of the input (prompt) when generating subsequent tokens, avoiding redundant calculations.

- o **Speculative Decoding:** Using a smaller, faster "draft" model to generate candidate token sequences, which are then efficiently verified or corrected by the larger main model.
- o **Batching:** Processing multiple user requests simultaneously to improve hardware utilization and throughput.
- o **Parallelization:** Distributing the model's computation across multiple processing units (like GPUs or TPUs).

Applications of LLMs

LLMs have demonstrated remarkable capabilities across a wide spectrum of applications:

- **Code/Math Assistance:** Generating, completing, refactoring, debugging, translating, and documenting code; explaining complex codebases; solving mathematical problems.
- **Machine Translation:** Producing more fluent, accurate, and context-aware translations between languages.
- **Summarization:** Condensing lengthy documents or articles into concise summaries highlighting key information.
- **Question Answering (RAG):** Enhancing question-answering systems by retrieving relevant information from external knowledge bases before generating an answer (Retrieval Augmented Generation).
- **Chatbots & Conversational AI:** Enabling more natural, engaging, and humanlike interactions in customer service, virtual assistants, and entertainment.
- **Content Creation:** Assisting with or autonomously generating various forms of creative text, including articles, marketing copy, scripts, poems, and emails.
- **Natural Language Inference:** Performing tasks like sentiment analysis, analyzing legal documents for specific clauses, assisting in medical diagnosis by processing clinical notes, and understanding customer feedback.

- **Text Classification:** Categorizing text for applications like spam detection, news topic classification, and intent recognition.
- **LLM Evaluation:** Serving as automated evaluators for assessing the quality of other AI models.
- **Text Analysis:** Extracting insights, identifying trends, and summarizing opinions from large volumes of unstructured text data.
- **Multimodal Applications:** Processing and generating content that combines text with other modalities like images, audio, and video.

Conclusion and Future

This exploration covered the foundational Transformer architecture, the rapid evolution of LLMs, techniques for training and fine-tuning, methods for evaluation and optimization, and the expanding range of applications. The field continues to advance at an accelerated pace, prompting questions about the novel applications future LLM generations will enable and the ethical, technical, and societal challenges that need addressing for responsible development and deployment.

Day 1: Prompt Engineering Techniques

Prompt Engineering Techniques for Large Language Models

Introduction to Prompt Engineering

Prompt engineering is the crucial skill of crafting effective inputs to guide Large Language Models (LLMs) toward desired outputs. While basic prompting is accessible to anyone, achieving specific, reliable, and high-quality results, particularly in data-intensive environments like Kaggle competitions, requires a deeper understanding of prompting techniques. This section aims to equip users, especially those on platforms like Kaggle, with practical methods to leverage LLMs

for enhancing coding, data analysis, and problem-solving capabilities. We will cover fundamental concepts through advanced strategies like Chain of Thought and ReAct, tailored for competitive data science challenges.

Configuring Model Output

Successfully utilizing LLMs involves not only crafting the input prompt but also configuring the model's generation parameters, as both influence the final output.

- **Output Length:** The maximum number of tokens the model generates directly impacts processing time and computational cost. This is particularly relevant for platforms with output limits or resource constraints. Precise prompts are needed for concise responses, especially when using iterative techniques like ReAct where multiple model calls might occur.
- **Sampling Controls:** These parameters govern the randomness and creativity of the model's output.
 - o **Temperature:** Controls the randomness of token selection. Lower temperatures (e.g., 0.1) produce more deterministic and focused outputs, ideal for tasks requiring specific formats or factual accuracy like generating syntactically correct code. Higher temperatures (e.g., 0.9) encourage more diversity and creativity, useful for brainstorming novel approaches or generating varied text formats.
 - o **Top-K and Top-P (Nucleus Sampling):** These parameters further refine the token selection process by limiting the pool of candidate tokens considered at each step. Top-K restricts the choice to the K most probable tokens, while Top-P selects from the smallest set of tokens whose cumulative probability exceeds a threshold P. Experimentation is key, as optimal settings vary by task. Combining these controls can fine-tune the balance between coherence and creativity.
- **Repetition Loop Bug:** A potential issue where the model gets stuck repeating words or phrases. Careful tuning of temperature, Top-K, and Top-P can help mitigate this behavior.

Recommendations for Kaggle:

- For coherent yet creative results (e.g., exploring feature engineering ideas): Temperature ~0.2, Top-P ~0.95, Top-K ~30.
- For highly creative output (e.g., generating diverse synthetic data examples): Temperature ~0.9, Top-P ~0.99, Top-K ~40.
- For factual accuracy (e.g., summarizing technical documentation): Temperature ~0.1, Top-P ~0.9, Top-K ~20.
- For tasks with a single correct answer (e.g., specific code translation): Temperature 0.

Prompt Engineering Techniques

Clear and well-structured prompts are fundamental for obtaining accurate and relevant predictions from LLMs.

- **General Prompting (Zero-Shot Prompting):** This involves providing the task description directly without including examples. It relies on the model's pre-trained knowledge and is often effective for straightforward tasks like generating simple code snippets based on natural language descriptions.
- **Documenting Prompts:** Systematically recording prompts and their resulting outputs is crucial, especially in competitive settings. This practice helps track effective strategies, identify patterns in model behavior, and facilitates iterative improvement and debugging.
- **One-Shot and Few-Shot Prompting:** These techniques involve including one (one-shot) or a few (few-shot) examples of the desired input-output behavior directly within the prompt. This guides the model more explicitly on the expected format, style, and task execution. The quality and relevance of the examples are paramount; poorly chosen or misleading examples can confuse the model. Including examples covering potential edge cases can further improve robustness.
- **System, Role, and Contextual Prompting:** These advanced techniques provide meta-instructions to shape the model's behavior more globally.

- o **System Prompting:** Sets the overall context, purpose, or constraints for the interaction (e.g., "You are a helpful assistant specializing in Python data analysis").
- o **Role Prompting:** Assigns a specific persona or identity to the LLM (e.g., "Act as a senior data scientist reviewing code"), influencing the tone, style, and focus of its responses.
- o **Contextual Prompting:** Provides specific background information, data snippets, or situational details relevant to the immediate task.
- **Step-Back Prompting:** Encourages the model to first consider the broader context or underlying principles related to a specific question before providing a direct answer. This can lead to more insightful, well-reasoned, and comprehensive outputs.

Advanced Reasoning Techniques

For complex problems requiring multi-step logic or exploration, advanced prompting frameworks can enhance LLM reasoning capabilities.

- **Chain of Thought (CoT) Prompting:** This technique guides the model to articulate its reasoning process step-by-step before arriving at the final answer. It involves providing examples in the prompt that demonstrate this thinking pattern. CoT improves performance on tasks requiring arithmetic, common sense, or symbolic reasoning, while also increasing the transparency and interpretability of the model's output.
- **Self-Consistency:** An enhancement to CoT where the model generates multiple reasoning paths (multiple CoT outputs) for the same prompt, often using a non-zero temperature. The final answer is determined by a majority vote across the different paths, improving robustness and reliability by mitigating the impact of any single flawed reasoning chain.
- **Tree of Thoughts (ToT):** Extends CoT by allowing the model to explore multiple reasoning paths concurrently, forming a tree structure. The model can evaluate the promise of intermediate thoughts and backtrack or explore different branches as needed. This is particularly suited for complex problems with large search spaces or requiring significant exploration.

- **ReAct (Reason and Act):** A framework that interleaves reasoning steps (thought) with actions (interacting with external tools or environments). The model generates a thought about what to do next, then an action (e.g., calling an API, running code, searching a database), observes the result of the action, and then uses that observation to inform the next thought-action cycle. This enables agents to dynamically gather information and interact with the world to solve problems.
- **Automatic Prompt Engineering (APE):** Techniques where LLMs themselves are used to generate or refine prompts for specific tasks, automating parts of the prompt engineering process.

Code Prompting Applications

LLMs are powerful tools for various coding-related tasks:

- **Code Generation:** Generating code snippets or entire functions based on natural language descriptions. While this can significantly speed up development, the generated code must be carefully reviewed, tested, and potentially refined to ensure correctness, efficiency, and security.
- **Explaining Code:** Providing natural language explanations of complex or unfamiliar code segments, aiding understanding, collaboration, and learning.
- **Translating Code:** Converting code between different programming languages. This can be useful when adapting algorithms or libraries, but the translated code requires thorough verification and testing.
- **Debugging and Reviewing Code:** Identifying potential errors, suggesting improvements, refactoring code for clarity or efficiency, and checking for adherence to style guides.
- **Multimodal Prompting:** An emerging area involving prompts that include non-textual inputs like images or diagrams alongside text instructions, potentially relevant for analyzing visualizations or data presented graphically in future Kaggle challenges.

Best Practices for Prompt Engineering

To maximize the effectiveness of LLMs, consider these best practices:

- **Provide Examples:** Use one-shot or few-shot prompting when possible to clearly demonstrate the desired output format and task execution.
- **Simplicity and Clarity:** Design prompts that are easy for the model (and humans) to understand. Avoid ambiguity.
- **Be Specific:** Clearly define the desired output, including format, length, tone, and any constraints.
- **Use Positive Instructions:** Frame requests positively (e.g., "Generate Python code that does X") rather than negatively (e.g., "Don't use loops").
- **Control Output Length:** Use parameters like `max_tokens` to manage response length, costs, and adhere to platform limits.
- **Dynamic Prompts:** Use variables or placeholders in prompt templates to easily adapt prompts for different inputs, datasets, or tasks, enhancing reusability.
- **Experimentation:** Try different phrasing, formats, examples, and model parameters to discover what works best for your specific task and model.
- **Collaboration:** Share successful prompts and strategies with peers to accelerate learning and innovation within the community.
- **Documentation:** Keep records of prompt attempts, configurations, and results to track progress, facilitate debugging, and build a knowledge base of effective techniques.
- **Adapt to Model Updates:** Be aware that LLM performance and behavior can change with new model versions; prompts may need adjustments.
- **Structured Output Formats:** For Kaggle, experiment with prompts that explicitly request output in specific structured formats (e.g., CSV, JSON) suitable for submission or further processing.

- **Reasoning First, Then Answer:** For logical tasks, structure prompts (especially with CoT) to output the reasoning steps before the final answer. Setting temperature to zero can enhance consistency for the final answer extraction.

Conclusion and Future Considerations

Mastering prompt engineering is becoming an essential skill for effectively leveraging LLMs, offering a significant advantage in competitive environments like Kaggle. As models continue to evolve, staying updated on new capabilities, techniques, and best practices is crucial. A mindset of continuous experimentation, iteration, adaptation, and learning is key to pushing the boundaries of what can be achieved with these powerful tools.

Day 2: Embeddings and Vector Stores

Introduction to Embeddings and Vector Stores

Embeddings are a fundamental concept in modern machine learning, providing a powerful way to represent diverse types of data in a common numerical format. They are essentially low-dimensional vectors designed to capture the underlying meaning, semantics, and relationships within the data. By transforming heterogeneous data types—such as text, images, audio, or structured information—into points in a multi-dimensional vector space, embeddings enable efficient processing, comparison, and pattern recognition across large datasets.

Think of embeddings as a way to map complex, high-dimensional data onto a simpler, more manageable "map," much like latitude and longitude represent locations on Earth. In this vector space, items with similar meanings or characteristics are positioned closer together, while dissimilar items are located farther apart.

Importance of Embeddings

The significance of embeddings stems from several key advantages:

- **Efficiency:** They provide compact representations for various data modalities, making storage and computation more manageable.
- **Pattern Recognition:** By placing similar items close together in the vector space, embeddings facilitate the discovery of patterns, clusters, and relationships that might be obscured in the original data format.
- **Semantic Relationships:** Embeddings are designed to capture semantic meaning. For instance, in a well-trained text embedding space, the vector for "king" would likely be closer to "queen" and "monarch" than to unrelated words like "bicycle" or "cloud".

Applications of Embeddings

Embeddings power a wide array of applications across different domains:

- **Retrieval Systems:** Search engines like Google utilize embeddings extensively. Web pages can be pre-computed into embedding vectors. When a user enters a search query, it is converted into an embedding, and the system retrieves pages whose embeddings are closest (nearest neighbors) in the vector space.
- **Recommendation Systems:** Embeddings represent users and items (e.g., products, movies, articles). Recommendations are generated by finding items whose embeddings are similar to a user's embedding (derived from their past interactions) or to items they have previously liked.
- **Multimodal Applications:** Joint embeddings allow mapping different data types (e.g., images and their corresponding text captions) into a shared embedding space, enabling tasks like cross-modal search (searching images using text queries or vice versa).
- **Classification and Clustering:** Embeddings can serve as features for downstream machine learning tasks, often improving performance by providing semantically rich input representations.
- **Anomaly Detection:** Outliers or unusual data points may appear distant from clusters of normal data points in the embedding space.

Evaluating Embedding Effectiveness

The quality of embeddings is crucial for the performance of downstream applications. Evaluation focuses on how well they capture relevant relationships and distinguish between similar and dissimilar items.

- **Relevance Assessment:** Primarily judged by their performance in retrieval tasks – how effectively they surface relevant items and suppress irrelevant ones.
- **Metrics:** Standard information retrieval metrics are often employed:
 - **Precision:** The proportion of retrieved items that are actually relevant.
 - **Recall:** The proportion of all existing relevant items that were successfully retrieved.
 - **Precision@K & Recall@K:** Focus the evaluation on the top K retrieved results, which is often critical in user-facing applications.
 - **Normalized Discounted Cumulative Gain (NDCG):** A ranked metric that assigns higher scores when highly relevant items appear earlier in the results list.
- **Benchmarks:** Standardized datasets and tasks, such as BEIR (Benchmarking-IR) and MTEB (Massive Text Embedding Benchmark), provide a common ground for comparing the performance of different embedding models across various retrieval scenarios.
- **Libraries:** Established libraries like `trec_eval`, `rank_eval`, or frameworks like PyTerrier offer tools for systematic evaluation using these metrics.
- **Practical Considerations:** Beyond relevance metrics, practical aspects like model size, embedding dimensionality (affecting storage and search speed), inference latency, and computational cost are important factors in choosing an embedding model.

Retrieval Augmented Generation (RAG)

Embeddings are a core component of Retrieval Augmented Generation (RAG), a powerful technique for enhancing Large Language Models (LLMs). RAG systems leverage embeddings to find relevant information from an external knowledge base (like a collection of documents or a database) to augment the prompt provided to the LLM. This grounds the LLM's response in

factual information, reducing hallucinations and improving the accuracy and relevance of its output.

The RAG process typically involves:

1. Index Creation (Offline):

- o Source documents are broken down into manageable chunks (e.g., paragraphs).
- o An embedding model (document encoder) generates an embedding vector for each chunk.
- o These embeddings are stored and indexed in a specialized database known as a vector database or vector store.

2. Query Processing (Online):

- o A user's query (question) is converted into an embedding vector using an appropriate query encoder (often the same or a related model as the document encoder).
- o A similarity search (typically Approximate Nearest Neighbor search) is performed against the indexed embeddings in the vector database to find the document chunks whose embeddings are closest to the query embedding.
- o The content of these relevant chunks is retrieved and combined with the original query to form an augmented prompt for the LLM.
- o The LLM generates a response based on this enriched context.

Efficient vector databases are crucial for the speed and scalability of RAG systems, enabling quick retrieval of relevant context during query processing. The quality of the embedding model itself is paramount; significant progress has been made, with models like Google's `text-embedding-004` achieving substantial improvements on benchmarks like BEIR.

Types of Embeddings

Embeddings can be categorized based on the type of data they represent:

- **Text Embeddings:** Represent textual units (words, sentences, paragraphs, documents) as numerical vectors.

- o **Early Methods:** Techniques like One-Hot Encoding represent token IDs sparsely, while methods like TF-IDF (Term Frequency-Inverse Document Frequency) and BM25 weigh words based on frequency within documents and across the corpus. Bag-of-Words models average word vectors. Latent Semantic Analysis (LSA) and Latent Dirichlet Allocation (LDA) use matrix factorization and probabilistic models, respectively, to find latent topics.
- o **Word Embeddings:** Capture semantic relationships between individual words.
 - *Word2Vec (CBOW, Skip-gram):* Learned based on the context words appear in ("a word is known by the company it keeps").
 - *GloVe (Global Vectors):* Trained on aggregated global word-word co-occurrence statistics.
 - *FastText:* Extends Word2Vec by considering subword information (character n-grams), handling rare words better.
 - *Swivel:* Uses a co-occurrence matrix factorization approach suitable for distributed processing.
- o **Sentence/Document Embeddings:** Aim to capture the meaning of longer text spans.
 - *Doc2Vec:* Extends Word2Vec by adding a paragraph vector.
 - *Deep Pre-trained Models:* Modern approaches leverage large transformer-based models like BERT, T5, PaLM, Gemini, GPT, and Llama. Specialized models like Sentence-BERT, SimCSE, E5, GTR, and SentenceT5 are fine-tuned specifically for producing high-quality sentence or passage embeddings. Techniques like Matryoshka embeddings offer flexibility by allowing truncation to lower dimensions, and multi-vector approaches (e.g., ColBERT) represent documents using multiple vectors for finer-grained retrieval.
- **Image Embeddings:** Typically obtained from the intermediate layers of Convolutional Neural Networks (CNNs) or Vision Transformers (ViTs) trained on large image datasets (e.g., ImageNet). These vectors capture visual features and concepts.

- **Multimodal Embeddings:** Combine embeddings from different modalities (e.g., image and text) into a shared space, enabling joint understanding and cross-modal tasks. Models like CLIP are prominent examples.
- **Structured Data Embeddings:** Representing tabular or relational data often requires application-specific techniques, as the meaning depends heavily on the schema and context. Techniques can involve learning embeddings for categorical features, applying dimensionality reduction like PCA, or learning joint embeddings for users and items in recommendation systems.
- **Graph Embeddings:** Represent nodes and their relationships within network structures (e.g., social networks, knowledge graphs). Algorithms like DeepWalk, Node2Vec, LINE, and GraphSage learn node representations that preserve graph topology and neighborhood information.

Training Embedding Models

Modern embedding models, especially for text and images, are often trained using architectures like the **Dual Encoder**. This typically involves two separate encoder models (e.g., Transformers): one for processing queries and one for processing documents/items.

During training, the objective is often to learn embeddings such that relevant query-document pairs have similar (close) vectors, while irrelevant pairs have dissimilar (distant) vectors. This is commonly achieved using **Contrastive Loss** functions. The loss function aims to pull the embeddings of positive pairs (query and its relevant document) closer together while simultaneously pushing the embeddings of negative pairs (query and irrelevant documents) farther apart in the vector space.

Training usually involves:

1. **Pre-training:** Initial training on large, general datasets to learn broad representations (often leveraging pre-trained LLMs or vision models).

2. **Fine-tuning:** Further training on smaller, task-specific datasets containing labeled relevance information (e.g., query-document relevance judgments). Fine-tuning data can be curated through manual labeling, generated synthetically, obtained via model distillation, or by mining "hard negatives" (irrelevant items that are difficult for the model to distinguish from relevant ones).

Trained embeddings can then be used directly for retrieval or as input features for other downstream tasks like classification or clustering.

Vector Search

Vector search, also known as similarity search or semantic search, is the process of finding items in a dataset whose embedding vectors are closest to a given query embedding. This allows searching based on semantic meaning rather than just keyword matching.

The core process involves:

1. Computing embeddings for all data items (documents, images, etc.).
 2. Storing and indexing these embeddings in a vector database.
 3. Embedding the incoming query into the same vector space.
 4. Searching the index to find the data items with the nearest embedding vectors to the query embedding. using metrics like Euclidean distance, cosine similarity, or dot product.
- Exact nearest neighbor search (linear scan) is often too slow ($O(N)$) for large datasets. Therefore, *Approximate Nearest Neighbor (ANN)* search algorithms are used, offering significantly faster search times (often $O(\log N)$) with a small trade-off in accuracy.

For large datasets, calculating exact distances between the query and all indexed items becomes computationally infeasible. **Approximate Nearest Neighbor (ANN)** search algorithms are used to significantly speed up this process by finding vectors that are highly likely to be

among the true nearest neighbors, albeit without absolute guarantees. Common ANN techniques include:

- **Locality Sensitive Hashing (LSH):** Uses hash functions designed such that similar items are more likely to map to the same hash bucket.
- **Tree-based Methods:** Partition the data space recursively using structures like KD-trees or Ball trees.
- **Graph-based Methods (e.g., HNSW):** Build a navigable graph structure where nodes are data points and edges connect close neighbors, allowing efficient traversal to find nearest neighbors.
- **Quantization-based Methods:** Compress the vectors (e.g., Product Quantization) to reduce memory usage and speed up distance calculations.
- **Hybrid Approaches:** Combining techniques, like Google's ScaNN (Scalable Nearest Neighbors), which often involves partitioning, quantization, and optimized distance calculations.

Vector Databases

Vector databases (or vector stores) are specialized database systems optimized for storing, indexing, and querying large collections of high-dimensional embedding vectors using ANN search algorithms. They provide the infrastructure needed for efficient vector search at scale.

While traditional databases are increasingly adding vector search capabilities (hybrid search), dedicated vector databases are specifically designed for this purpose. The typical workflow involves embedding data, indexing the vectors using an ANN algorithm, embedding incoming queries, and performing similarity searches.

Examples of vector database solutions include managed cloud services like Vertex AI Vector Search (leveraging ScaNN), extensions to existing databases (e.g., pgvector for PostgreSQL, available in AlloyDB and Cloud SQL), and specialized databases like Pinecone, Weaviate, and ChromaDB.

Key operational considerations when choosing and managing a vector database include scalability, availability, consistency guarantees, efficient handling of data updates, backup mechanisms, and security.

Applications of Embeddings and Vector Stores

The combination of high-quality embeddings and efficient vector search infrastructure enables a wide range of powerful applications:

- **Information Retrieval / Semantic Search:** Finding documents, products, or information based on meaning rather than just keywords.
- **Recommendation Systems:** Suggesting items similar to those a user has interacted with or similar to other users with comparable tastes.
- **Retrieval Augmented Generation (RAG):** Enhancing LLM responses by retrieving relevant context from external knowledge bases. Providing sources for retrieved information is crucial for building user trust.
- **Semantic Text Similarity:** Finding pairs of sentences or documents with similar meanings.
- **Classification and Clustering:** Grouping similar items together or categorizing items based on their embedding representations.
- **Reranking:** Refining the results of an initial retrieval stage (e.g., keyword search) using semantic similarity scores from embeddings.
- **Anomaly Detection:** Identifying unusual data points that are distant from others in the embedding space.
- **Few-Shot Classification:** Classifying new items based on their similarity to a small number of labeled examples.

Embeddings and vector stores are becoming foundational components in building intelligent systems capable of understanding and processing information in a more human-like, meaning-driven way.

Day 3: Agents

Generative AI Agents: A Deep Dive

Introduction

Humans naturally extend their capabilities by using tools – from simple implements to complex information systems like books and search engines. Similarly, Generative AI models, particularly Large Language Models (LLMs), can be empowered to use external "tools" to access real-time information, interact with other systems, and perform actions in the digital or even physical world. For instance, an AI agent could query a database for a customer's purchase history to offer personalized shopping recommendations or utilize an API to send an email or complete a financial transaction. These AI agents represent a significant step beyond standalone models, integrating reasoning, logic, and interaction with external resources to achieve specific goals.

What is a Generative AI Agent?

A Generative AI agent is more than just an LLM; it's a system built around a core generative model, augmented with capabilities for reasoning, planning, and tool use. Key characteristics include:

- **Goal-Oriented:** Designed to achieve specific objectives.
- **Observation:** Perceives its environment or context (e.g., user requests, tool outputs).
- **Action:** Takes actions using available tools (e.g., APIs, databases, code execution).
- **Autonomy:** Can operate independently to figure out the necessary steps to reach a goal, often without explicit step-by-step instructions.
- **Reasoning:** Utilizes the LLM's capabilities for planning, decision-making, and logic.

In essence, the LLM acts as the "brain" or central processing unit of the agent, directing its operations based on its goal and observations.

Cognitive Architecture of Agents

The "cognitive architecture" defines the internal structure and operating principles of an agent, governing how it perceives, reasons, and acts. It typically comprises three main components:

- **a. Model:** The core intelligence, usually one or more LLMs. These models provide the reasoning, language understanding, and generation capabilities. The choice of model(s) depends on the agent's specific needs – it could be a large general-purpose model, a smaller specialized model, a multimodal model, or even a model fine-tuned on data relevant to the agent's tasks (e.g., fine-tuned on examples of using a specific API). While models aren't typically pre-trained specifically *as* agents, fine-tuning can teach them how to effectively utilize tools and follow reasoning frameworks. Instruction-based reasoning frameworks like ReAct or Chain of Thought are often employed via prompting or fine-tuning.
- **b. Tools:** The mechanisms enabling the agent to interact with the external world beyond its internal knowledge. Tools allow the agent to retrieve up-to-date information, interact with other software systems (via APIs), perform calculations, or even control physical devices. Examples range from simple search engine queries and database lookups (crucial for Retrieval Augmented Generation - RAG) to complex actions like booking flights, managing calendar events, or executing code. Tools bridge the gap between the agent's internal reasoning and external reality.
- **c. Orchestration Layer:** The control system that manages the agent's operation cycle. It receives input, passes it to the model for reasoning and planning, interprets the model's output to select appropriate tools and actions, executes those actions, gathers the results (observations), and feeds them back to the model for the next cycle. This layer ensures the smooth coordination between the model and tools, manages the agent's internal state (memory), and guides the process until the goal is achieved or a stopping condition is met. The complexity can range from simple loops to sophisticated systems involving custom logic or machine learning components.

Agents vs. Models

It's crucial to distinguish between a standalone LLM and a fully-fledged agent:

- **a. Knowledge:** Models are limited by the data they were trained on (knowledge cutoff). Agents, through tools, can access and incorporate real-time, external information.
- **b. Information Handling Over Time:** Models typically process input and generate output in a single turn. Agents often maintain a history of interactions (memory), allowing for multi-turn conversations and stateful task execution.
- **c. Tool Support:** Models lack inherent mechanisms for external interaction. Tools are a fundamental, integrated part of an agent's architecture.
- **d. Logic Layer:** Guiding model output often relies heavily on prompt engineering. Agents possess a dedicated orchestration layer and cognitive architecture that explicitly incorporates reasoning frameworks and manages the interaction between thought and action.

How Agents Operate: Cognitive Architectures

Imagine an agent as a chef in a busy kitchen. The chef (agent) receives an order (goal/input). They assess available ingredients (tools, information), plan the dishes (reasoning/planning), take actions like chopping, mixing, cooking (tool use), observe the results (feedback), and adjust accordingly.

Similarly, an AI agent operates in a cycle:

1. **Receive Input/Goal:** Understand the user's request or the objective.
2. **Reason/Plan:** The model, guided by the orchestration layer and reasoning frameworks, determines the next step or action needed. Prompt engineering plays a key role here.
3. **Select & Execute Action:** The orchestration layer invokes the chosen tool with the necessary parameters.

4. **Observe Result:** The outcome of the action (e.g., API response, database query result, error message) is captured.
 5. **Update State/Reason:** The observation is fed back to the model to inform the next reasoning step, potentially refining the plan or choosing the next action.
- This loop continues until the goal is achieved. The orchestration layer tracks progress, manages the task state (memory), and oversees the reasoning process.

Reasoning and Planning Frameworks

Several frameworks guide the agent's thought process, often implemented through careful prompting:

1. **ReAct (Reason and Act):** Explicitly prompts the model to cycle through:
 - o **Thought:** Verbalize the reasoning about the current situation and what needs to be done next.
 - o **Action:** Specify the tool to use and the parameters.
 - o **Observation:** Process the result received after executing the action.

This framework encourages step-by-step interaction with the environment (via tools) and makes the agent's process more transparent and potentially more reliable.
2. **Chain of Thought (CoT):** Prompts the model to break down a problem and articulate its intermediate reasoning steps before providing a final answer. While not inherently interactive like ReAct, it improves performance on complex reasoning tasks. Variations exist, including self-consistency (generating multiple chains and taking a majority vote), active prompt, and multimodal CoT.
3. **Tree of Thoughts (ToT):** Extends CoT for problems requiring exploration. The agent explores multiple reasoning paths or possibilities simultaneously, forming a tree structure, allowing it to evaluate different potential solutions or plans.

ReAct in Practice: Example

Consider a user wanting to book a flight:

1. **User:** "Book me a flight."
2. **Agent (Thought):** The user's request is vague. I need the origin and destination.
3. **Agent (Action):** Ask Clarifying Question: "Where are you flying from and to?"
4. **User:** "From London to New York."
5. **Agent (Observation):** Origin=London, Destination=New York. Now I need dates.
6. **Agent (Thought):** I have the route. I need travel dates to check availability.
7. **Agent (Action):** Ask Clarifying Question: "What dates would you like to travel?"
8. ... (Cycle continues, potentially involving calling a flight API tool based on subsequent thoughts and observations) ...

Tools: Keys to the Outside World

LLMs excel at processing information and generating text, but they cannot inherently interact with external systems or access real-time data. Tools provide this crucial connection. For Google's Vertex AI agents, tools generally fall into three categories:

- **a. Extensions:** Standardized connectors that allow an agent to interact with an API (e.g., Google Search, GMail, Calendar, custom enterprise APIs). Extensions simplify API usage, often handling authentication and data formatting. The agent learns (usually through examples provided during configuration or fine-tuning) how and when to use specific extensions based on the user's query and the extension's description. They are typically executed on the agent's side (server-side). A notable example is the code interpreter extension, which allows the agent to generate and execute Python code in a sandboxed environment.
- **b. Functions (Function Calling):** Self-contained pieces of code (defined by the developer) that perform specific tasks. The language model decides *when* to call a function and determines the appropriate arguments based on the function's description (provided in the prompt or configuration). However, the actual execution of the function happens on the client-side (the developer's infrastructure), not within the agent itself. This provides

greater control over execution, security, and handling of sensitive operations or authentication.

- **c. Data Stores:** Provide access to external knowledge bases, enabling Retrieval Augmented Generation (RAG). Agents can query these stores (often vector databases containing indexed documents, web pages, PDFs, spreadsheets, etc.) to retrieve relevant, up-to-date information needed to answer user questions or complete tasks. This allows agents to access vast amounts of domain-specific or timely data without requiring constant retraining of the core LLM.

Retrieval Augmented Generation (RAG)

RAG is a specific pattern of tool use where the agent retrieves information from an external knowledge base (often using vector search on embeddings stored in a Data Store) to augment its internal knowledge before generating a response. This significantly improves factual accuracy and allows the agent to answer questions about information not present in its original training data.

Tools Recap

Tool	Execution Location	Best Used For
Extensions	Agent-side	Directly controlling API interactions, pre-built integrations, multiple API calls.
Functions	Client-side	Security concerns, authentication needs, fine-grained control over API execution.
Data Stores	Agent-side	Implementing RAG, providing access to broad external knowledge sources.

Enhancing Model Performance

Ensuring the LLM reliably chooses the correct tool and uses it effectively is a key challenge.

Approaches include:

- **a. In-Context Learning:** Providing clear descriptions of tools and few-shot examples of their usage directly within the prompt. This guides the model on how to reason about tool selection and parameterization for the current task.
- **b. Retrieval-Based In-Context Learning:** Similar to RAG, but retrieving relevant examples of tool usage from a larger library based on the current query, rather than including static examples in the prompt.
- **c. Fine-Tuning Based Learning:** Training the core LLM on a dataset specifically containing examples of task decomposition, reasoning, and tool usage relevant to the agent's intended functions. This can significantly improve the model's proficiency in acting as an agent.

Often, combining these approaches yields the best performance.

Agent Quick Start with LangChain

Frameworks like LangChain and LangGraph simplify the process of building agents. They provide abstractions for defining agents, tools, and orchestration logic. An example might involve creating an agent using a Gemini model, equipping it with tools for Google Search and the Google Places API, and tasking it to answer a multi-step question like, "Who did the Texas Longhorns play in their last game and what's the address of the stadium?" The agent would need to use the search tool to find the opponent and game details, then potentially use the Places API tool to find the stadium address.

Vertex AI Agents

Google Cloud's Vertex AI offers a managed platform specifically designed for building, deploying, evaluating, and managing enterprise-grade generative AI agents. It provides user interfaces, pre-built components, evaluation tools, and systems for monitoring and continuous

improvement, often allowing agent behavior and tool usage to be defined using natural language instructions.

Main Takeaways

- Generative AI agents represent a significant evolution from standalone LLMs, enabling interaction with the real world and autonomous goal achievement.
- They combine a core model (the brain) with tools (the hands) managed by an orchestration layer.
- Cognitive architectures and reasoning frameworks (like ReAct, CoT) guide their operation.
- Tools (Extensions, Functions, Data Stores) are essential for external data access and action execution. RAG is a critical pattern enabled by Data Stores.
- Building reliable agents involves careful tool selection, performance enhancement techniques (prompting, fine-tuning), and robust evaluation.
- Platforms like Vertex AI and frameworks like LangChain facilitate agent development.
- The field is rapidly evolving, with future directions including more sophisticated tools, improved reasoning, multi-agent systems (agent chaining), and iterative development processes.

Day 3: Agents Companion

AI Agents White Paper Companion: A Deep Dive

Introduction to Generative AI Agents

Generative AI agents mark a pivotal advancement beyond conventional language models. These systems are engineered to achieve defined objectives through a cycle of perceiving their environment, formulating strategies, and acting upon that environment using a repertoire of available tools. The essence of an agent lies in its capacity to integrate reasoning, logical deduction, and access to external information sources. A key characteristic is their potential for

autonomous operation, enabling them to pursue goals and determine appropriate actions without needing explicit, step-by-step human guidance.

Advanced Agent Concepts for Developers

As the field progresses from experimental prototypes to production-ready applications, developers require a deeper understanding of advanced concepts and emerging best practices for building reliable and effective agents. This companion guide aims to accelerate that understanding, focusing on the operational challenges and sophisticated architectures needed for real-world deployment.

Agent Ops: Tailored DevOps for AI Agents

Operationalizing AI agents necessitates a specialized approach, termed "Agent Ops," which blends principles from DevOps (managing software development and deployment) and MLOps (managing machine learning model lifecycles). Agent Ops specifically addresses the unique challenges of agent systems:

- **Tool Management:** Robust processes for defining, versioning, testing, and monitoring the tools agents rely on.
- **Workflow Orchestration:** Managing the complex sequences of reasoning, action, and observation, especially in multi-step tasks or multi-agent systems.
- **Memory Handling:** Efficiently managing short-term (conversational context) and long-term (persistent knowledge) memory for agents.
- **Task Decomposition:** Strategies for breaking down large, complex goals into smaller, manageable sub-tasks that agents can tackle.

The objective of Agent Ops is to ensure agents function reliably, predictably, and maintainably, akin to a well-engineered machine with built-in monitoring and maintenance systems.

Measuring Success with Business KPIs

While technical metrics are important, the ultimate success of an AI agent should be measured against tangible business Key Performance Indicators (KPIs). Since agents are tools designed to achieve specific outcomes, evaluations must track metrics like:

- **Goal Completion Rate:** How often does the agent successfully achieve its assigned objective?
- **User Engagement/Satisfaction:** Are users finding the agent helpful and easy to interact with?
- **Efficiency Gains:** Does the agent reduce human effort or process time?
- **Revenue Impact:** Does the agent contribute directly or indirectly to business revenue?

Achieving this requires instrumenting the agent system to capture granular operational metrics (e.g., task success rates, tools used, errors encountered, user interaction patterns) that feed into these higher-level business KPIs. Detailed logging of agent actions is also crucial for debugging and understanding failure modes, providing insights into *how* an agent arrived at an outcome, not just the outcome itself.

The Importance of Human Feedback

Quantitative metrics provide only part of the picture. Human feedback remains invaluable for understanding agent performance in nuanced, real-world scenarios. Mechanisms for collecting this feedback include:

- Simple thumbs up/down ratings on agent responses.
 - User surveys targeting specific aspects of the interaction.
 - Open-ended feedback forms allowing users to describe issues or successes in detail.
- This qualitative data provides crucial insights into user experience, perceived helpfulness, and areas where the agent might be failing in ways not captured by automated metrics (e.g., tone, common sense failures, subtle biases).

Automated Evaluation

Given the complexity and potential scale of agent operations, automated evaluation is essential for continuous monitoring and quality assurance. Automated methods can assess various aspects:

- **Agent Capabilities:** Testing the agent's ability to perform specific tasks correctly.
- **Problem-Solving Trajectory:** Analyzing the sequence of steps (reasoning and actions) the agent takes.
- **Final Response Quality:** Judging the accuracy, relevance, and coherence of the agent's output.

Analyzing Agent Trajectory

Evaluating *how* an agent reached a solution is critical for understanding its efficiency and reliability. Key questions include: Did it select the appropriate tools? Was its reasoning process logical and efficient? Did it encounter and recover from errors? Did it waste resources exploring unproductive paths?

Techniques for measuring trajectory quality include:

- **Exact Match:** Comparing the agent's sequence of actions against a predefined "golden" path. (Often too rigid).
- **In-Order Match:** Checking if essential steps are performed in the correct relative order, allowing for some flexibility.
- **Any-Order Match:** Suitable when the sequence of actions is less critical than ensuring all necessary actions are taken.

Judging the Quality of Final Response

One automated approach uses "LLM auditors," where one LLM is tasked with evaluating the output of another (the agent's LLM) based on predefined criteria (e.g., factual accuracy, helpfulness, adherence to instructions). The auditor compares the agent's response against these standards, providing a scalable way to perform quality checks.

The Role of Humans in Evaluation

Despite advances in automated evaluation, human judgment remains indispensable, particularly for assessing qualities that are difficult to quantify:

- Creativity and novelty.
- Common sense reasoning.
- Subtle nuances of language, tone, and context.
- Ethical considerations and potential biases.

Human oversight ensures that automated evaluation metrics align with actual user needs and real-world expectations, providing essential calibration and validation.

Multi-Agent Systems: Divide and Conquer

Instead of relying on a single, monolithic agent to handle complex, multifaceted problems, a multi-agent system approach breaks down the problem into smaller, more specialized tasks. Each task is then assigned to an agent specifically designed or configured for it. This mirrors real-world teamwork, leveraging specialization for improved performance.

Benefits of Multi-Agent Systems

- **Accuracy:** Specialized agents can achieve higher proficiency; agents can also cross-check or validate each other's work.
- **Efficiency:** Tasks can often be processed in parallel by different agents, reducing overall completion time.
- **Scalability:** System capacity can potentially be increased by adding more specialized agents.
- **Fault Tolerance:** If one agent fails, others might be able to compensate or take over its task, increasing system robustness.

- **Mitigation of AI Issues:** Combining outputs from multiple agents with potentially different perspectives or training data can help reduce the impact of individual model biases or hallucinations.

Structuring Multi-Agent Systems: Design Patterns

Organizing the interaction and workflow between multiple agents requires thoughtful design.

Common patterns include:

- **Sequential Pattern:** Agents operate in a pipeline, where the output of one agent becomes the input for the next. Simple, but can be slow and prone to single points of failure.
- **Hierarchical Pattern:** A "manager" or "orchestrator" agent oversees a team of "worker" agents. The manager decomposes the task, delegates sub-tasks to appropriate workers, monitors progress, and synthesizes the final result.
- **Collaborative Pattern:** Agents work as peers, sharing information, contributing expertise, and coordinating amongst themselves (potentially through a shared workspace or message bus) to achieve a common goal.
- **Competitive Pattern:** Multiple agents work on the same task independently, perhaps using different approaches. The best solution is then selected (e.g., based on confidence scores or external validation). Effective for optimization or exploration problems.

Challenges in Building Multi-Agent Systems

- **Task Allocation:** Efficiently assigning tasks to the most suitable agent.
- **Coordination:** Ensuring agents work together effectively, share information appropriately, and avoid conflicts.
- **Context Management:** Managing the potentially large volume of information (context) that needs to be shared or maintained across agents.
- **Complexity & Cost:** Designing, implementing, and managing multi-agent systems can be significantly more complex and potentially costly than single-agent systems.

Evaluating Multi-Agent Systems

Evaluation extends beyond individual agent performance to assess the effectiveness of the collaboration:

- Are tasks being routed correctly?
- Is communication between agents clear and efficient?
- Is the overall system achieving the goal more effectively than a single agent could?

Techniques like trajectory analysis and final response evaluation are still applicable but need to consider the interactions between agents.

Agentic RAG: Retrieval Augmented Generation with Agents

Agentic RAG enhances the standard RAG process by incorporating intelligent agents into the retrieval and synthesis loop. Instead of a simple query-retrieve-generate pipeline, agents can play a more active role:

- **Query Refinement:** An agent might analyze an initial user query, break it down, and formulate more targeted search queries for the retrieval system.
- **Information Evaluation:** An agent could assess the relevance and credibility of retrieved information before passing it to the generation model.
- **Adaptive Retrieval:** Agents could dynamically adjust the retrieval strategy based on the context or intermediate results.

This leads to potentially more accurate, contextually relevant, and adaptable RAG systems.

Optimizing Basic Search Engine (for RAG/Agentic RAG)

The effectiveness of any RAG system heavily depends on the underlying search/retrieval component. Optimization steps include:

- **Parsing and Chunking:** Effectively segmenting source documents into meaningful, appropriately sized chunks for embedding.
- **Metadata Enrichment:** Adding relevant metadata (keywords, authors, dates, categories, synonyms) to chunks to aid filtering and retrieval.

- **Embedding Model Tuning:** Fine-tuning the embedding model on domain-specific data or using techniques like search adapters to improve relevance.
- **Vector Database Performance:** Utilizing a fast and scalable vector database for efficient ANN search.
- **Re-ranking:** Implementing a second-stage ranker (potentially a cross-encoder model) to re-order the initial set of retrieved candidates based on finer-grained relevance assessment.

Google Cloud Tools for Search

Google Cloud provides tools to facilitate building sophisticated search and RAG systems:

- **Vertex AI Search:** A managed service offering Google-quality search capabilities over private enterprise data.
- **Vertex AI Search Builder APIs:** Allow for more customization in creating tailored search engines.
- **Vertex AI RAG Engine:** Provides orchestration capabilities for building and managing RAG pipelines.

Real-World Example: Google's Co-scientist

Google's Co-scientist project demonstrates a multi-agent system applied to scientific discovery. It employs specialized agents to generate hypotheses, design experiments (in silico), search literature, analyze results, and refine understanding. In one case study, it successfully identified existing drugs potentially useful for liver fibrosis and proposed novel drug candidates by navigating complex biological pathways.

Automotive AI: Multi-Agent Systems in Cars

Modern vehicles require sophisticated AI to handle diverse user needs, including navigation, media control, answering vehicle-specific questions (e.g., from the car manual), and handling general knowledge queries. A multi-agent approach is emerging as a natural fit:

- Conversational navigation agent.
- Media search and control agent.
- Car manual Q&A agent.
- General knowledge/assistant agent.

Patterns in Automotive AI

Different interaction patterns can be used:

- **Hierarchical:** A central orchestrator routes queries to the appropriate specialist agent.
- **Diamond:** Responses from specialist agents are filtered through a central moderator agent responsible for ensuring consistency in tone, style, and safety before being presented to the user.
- **Peer-to-Peer:** Agents communicate directly as needed (less common due to complexity).
- **Collaborative:** Multiple agents might need to work together to answer complex queries spanning different domains (e.g., "Find a charging station near a highly-rated Italian restaurant along my route").

Benefits of Multi-Agent Approach in Automotive AI

- **Quality:** Specialized agents provide more accurate and relevant responses within their domain.
- **Efficiency:** Queries are handled by the most appropriate resource quickly.
- **Resilience:** Failure in one specialized agent might not cripple the entire system.

Agent Builder: Google Cloud's Toolkit for Agent Developers

Agent Builder is Google Cloud's suite of tools aimed at simplifying the development, deployment, and management of generative AI agents. Key components include:

- **Vertex AI Agent Engine:** Facilitates building and deploying agents, likely incorporating features for defining behavior, managing tools, and orchestration.

- **Vertex AI Evaluation Service:** Provides tools specifically for evaluating LLMs, RAG systems, and agent performance, incorporating both automated and human feedback loops.

Agents as Contractors: A Conceptual Shift

As AI agents gain autonomy, simply giving them instructions might be insufficient for ensuring reliable and accountable behavior. An emerging concept suggests applying principles from real-world human contracts to AI agents. This involves defining clear expectations, scope of work, performance criteria, risk management protocols, and potentially even mechanisms for dispute resolution. This conceptual shift aims to foster greater accountability, transparency, and trustworthiness as AI systems become more capable and integrated into critical processes.

Agentic RAG Deep Dive

Revisiting Agentic RAG, the key distinction from standard RAG is the active role of agents within the loop. An agent might receive a user query, realize it's ambiguous, interact with the user to clarify, formulate precise search queries, evaluate the retrieved chunks for relevance and potential contradictions, synthesize the information, and potentially even update the knowledge base based on the interaction. This iterative, intelligent process allows for more robust and context-aware information retrieval and generation. Check grounding, where the agent verifies claims in its generated response against the retrieved source information, is a crucial technique [here](#).

Optimizing Search for Agentic RAG

The search optimization techniques mentioned earlier (parsing, metadata, embedding tuning, vector DB speed, re-ranking) are even more critical for Agentic RAG, as the agents rely heavily on the quality and efficiency of the retrieval step to perform their reasoning and evaluation effectively.

Conceptual Shifts: Trust and Reliability

As AI agents become more autonomous and take on higher-stakes tasks, establishing trust and ensuring reliability are paramount. Key pillars contributing to trustworthy AI agents include:

- **Transparency:** Understanding how an agent makes decisions. This involves explainability (making the reasoning process clear, e.g., via ReAct traces) and auditability (allowing users or regulators to review actions and data).
- **Accountability:** Defining responsibility for the outcomes of agent actions. This requires robust monitoring, logging, and potentially legal or organizational frameworks to address errors or unintended consequences.
- **Robustness:** Ensuring the agent can handle unexpected inputs, environmental changes, or tool failures gracefully without catastrophic errors. This involves rigorous testing, incorporating fail-safe mechanisms, and designing for adaptability.
- **Fairness:** Designing and evaluating agents to ensure they do not perpetuate harmful biases or discriminate against specific groups. This requires careful consideration of training data, algorithmic design, and evaluation metrics focused on equity.

Building agents that embody these principles is essential for their responsible adoption and long-term success.

Day 4: Solving Domain-Specific Problems Using LLMs: Cybersecurity and Medicine

Introduction to LLMs and Domain Specialization

While general-purpose Large Language Models (LLMs) demonstrate broad capabilities, their true potential often unfolds when they are specialized for specific domains. Fine-tuning and adapting LLMs for fields like cybersecurity and medicine can yield significant improvements in tackling complex, domain-specific challenges. However, applying LLMs in these areas presents unique hurdles, including the scarcity of public specialized data, the need to understand intricate technical language, and the critical importance of handling sensitive information and use cases responsibly. This chapter explores how LLMs are being tailored for these demanding fields, focusing on Google's SecLM for cybersecurity and Med-PaLM for healthcare.

Cybersecurity Challenges and SecLM

The cybersecurity landscape is characterized by a constant influx of diverse and evolving threats, a large volume of operational tasks for security teams, and a persistent shortage of skilled professionals. Analyzing security data often involves navigating limited public datasets, understanding highly technical concepts (malware behavior, network protocols, exploit techniques), and dealing with sensitive information related to vulnerabilities and incidents. Use cases like malware analysis demand particularly careful model development to avoid unintended risks.

To address these challenges, specialized models like **SecLM (Security Language Model)** are being developed. SecLM represents not just a security-focused LLM but an ecosystem of supporting techniques designed to assist security professionals in tasks like threat identification, risk analysis, incident response, and vulnerability management.

Pressures in Cybersecurity

- **Sophisticated Attacks:** Adversaries constantly devise new attack vectors and malware.
- **Operational Toil:** Security teams face alert fatigue and repetitive investigation tasks.
- **Skills Gap:** A global shortage of experienced cybersecurity analysts persists.

LLMs as AI Assistants in Cybersecurity

LLMs offer the potential to act as powerful AI assistants, augmenting human expertise and automating laborious tasks:

- Translating natural language requests into complex query languages used by security platforms (e.g., SIEM, SOAR).
- Automating the initial investigation and categorization of security alerts.
- Generating personalized remediation plans based on identified threats and system configurations.
- Assisting in reverse engineering malware by explaining code behavior.

- Summarizing threat intelligence reports and identifying key indicators of compromise (IOCs).
- Providing insights into potential attack pathways within an organization's infrastructure.
- Identifying critical areas for security testing (e.g., penetration testing) and generating secure code examples.

Layered Approach in Cybersecurity

Effectively integrating LLMs into security operations often involves a layered approach:

1. **Foundation:** Existing security tools (SIEM, EDR, firewalls) provide raw data and context.
2. **Intelligence Core:** A specialized model API, like SecLM, processes this data, answers queries, and performs analysis.
3. **Oversight:** Authoritative security intelligence feeds (threat databases) and crucial human expertise guide and validate the system.

SecLM as a Central Resource

The vision for SecLM is to serve as a central, conversational interface for security operations. Analysts could ask complex questions in natural language (e.g., "Summarize recent activity related to threat group AP41 affecting our European servers") and receive synthesized answers grounded in the organization's internal security data and external threat intelligence.

Standards for SecLM

Developing a reliable security LLM requires meeting high standards:

- **Timeliness:** The model must incorporate knowledge of the latest threats, vulnerabilities, and attacker techniques (TTPs).
- **Data Sensitivity:** Must operate securely, analyzing potentially sensitive user or organizational data without risking exposure or leakage.
- **Deep Security Knowledge:** Requires a nuanced understanding of cybersecurity concepts, terminology, tools, and workflows.

- **Multi-Step Reasoning:** Needs the ability to break down complex queries, combine information from multiple sources (logs, threat feeds, vulnerability databases), and potentially orchestrate multiple tools or models to arrive at an answer.

Reasons General-Purpose LLMs Fall Short

General-purpose LLMs often struggle in the cybersecurity domain due to:

- **Data Scarcity:** High-quality, labeled cybersecurity data is less abundant publicly compared to general web text.
- **Knowledge Depth:** The required breadth and depth of specialized security knowledge often exceed that of general models.
- **Sensitive Use Cases:** Tasks like analyzing live malware samples require specialized handling and safety protocols not inherent in general models.

Creating Specialized SecLMs: Targeted Training Approach

Building effective SecLMs involves a multi-stage, targeted training process:

1. **Foundation:** Start with a capable general-purpose foundation model (e.g., Gemini or PaLM), ideally one with strong multilingual capabilities to handle diverse data sources.
2. **Domain Pre-training:** Continue pre-training the model on a large corpus of cybersecurity-specific text, including security blogs, threat intelligence reports, malware analyses, security framework documentation (e.g., MITRE ATT&CK), detection rules (e.g., YARA, Sigma), and potentially textbooks.
3. **Supervised Fine-Tuning (SFT):** Fine-tune the model on curated datasets containing examples of tasks that mimic real-world security expert activities. This includes:
 - o Analyzing potentially malicious code snippets or scripts.
 - o Explaining the purpose and potential impact of specific command-line executions.
 - o Interpreting various security event log formats.
 - o Summarizing lengthy and complex threat reports.
 - o Generating queries for specific security platforms (e.g., Chronicle, Splunk).

4. **Privacy Focus:** Ensure that fine-tuning data related to specific users or organizations is kept separate and handled securely, potentially using techniques like Parameter Efficient Tuning for on-premise customization.

Evaluating Performance of Specialized Models

Evaluating SecLMs requires a combination of methods:

- **Closed-Ended Tasks:** For tasks with definitive answers (e.g., classifying malware families, extracting IOCs), standard classification or extraction metrics (accuracy, precision, recall, F1-score) can be used against benchmark datasets.
- **Open-Ended Tasks:** For generative tasks (e.g., summarizing reports, explaining concepts, generating remediation advice), compare model outputs against expert-written answers using metrics like ROUGE (for content overlap) and BERTScore (for semantic similarity).
- **Automated Side-by-Side Comparison:** Use capable general-purpose LLMs (like GPT-4 or Gemini Ultra) as automated evaluators to compare the quality of responses from different SecLM versions based on predefined criteria (accuracy, completeness, clarity).
- **Human Evaluation:** Crucially, involve human cybersecurity experts to provide nuanced judgments on the quality, accuracy, actionability, and safety of the model's outputs in realistic scenarios.

Techniques to Help Models

Beyond core training, several techniques enhance SecLM's capabilities and adaptability:

- **In-context Learning (Few-Shot Prompting):** Allow the model to adapt quickly to new tools, platforms, or data formats by providing examples directly within the prompt.
- **Parameter Efficient Tuning (PET):** Enable users or organizations to customize a base SecLM model with their own private data (e.g., internal security policies, specific tool query syntax) without needing to retrain the entire large model, preserving privacy.
- **Retrieval Augmented Generation (RAG):** Connect SecLM to external, continuously updated knowledge bases (e.g., threat intelligence feeds, vulnerability databases, internal

incident reports) in real-time. This keeps the model informed about the latest threats without constant retraining.

Flexible Planning and Reasoning Framework

SecLM often acts as an orchestrator within a larger system. For example, responding to a query like "What evidence do we have of AP41 activity in our network?" might involve SecLM coordinating several steps:

1. **Retrieve:** Use RAG to fetch information about AP41's known TTPs and IOCs from a threat intelligence database.
2. **Extract:** Identify key patterns (file hashes, IP addresses, domains, specific commands) from the retrieved information.
3. **Translate:** Convert these patterns into a syntactically correct query for the organization's SIEM system (e.g., Chronicle Query Language).
4. **Execute & Analyze:** Run the query against the SIEM logs and summarize the findings for the analyst.

This multi-step reasoning and tool-use capability, whether predefined or dynamically generated, can automate tasks that previously took analysts significant time.

SecLM Applications

The SecLM ecosystem envisions agents interacting with various tools and data sources:

- Using RAG to query external security platforms.
- Employing smaller, specialized models fine-tuned for specific analytical tasks (e.g., malware classification, log parsing).
- Utilizing long-term memory to retain user preferences, context from previous interactions, and details about the organization's environment.

Ultimate Goal for SecLM

The aspiration is for SecLM to become a transformative platform in cybersecurity, significantly reducing the daily operational burden on security professionals, amplifying their expertise, and improving the speed and effectiveness of threat detection and response.

Healthcare and Med-PaLM

Similar to cybersecurity, healthcare presents immense opportunities and significant challenges for LLMs. The potential to improve diagnostics, streamline clinical workflows, enhance patient communication, and accelerate medical research is vast. However, the paramount importance of patient safety, data privacy (HIPAA), and the need for rigorous validation demand a highly responsible approach.

Med-PaLM is Google's initiative to develop LLMs specifically adapted for the medical domain, building upon the PaLM family of models, with a focus on improving health outcomes through safe and helpful AI applications.

Potential Uses of GenAI in Healthcare

- **Patient Engagement:** Answering patient questions about their conditions or medical history, providing personalized guidance (within safe boundaries).
- **Clinical Workflow:** Triageing patient messages, assisting with clinical documentation (e.g., summarizing patient encounters), drafting referral letters.
- **Patient Intake:** Revolutionizing data collection before appointments.
- **Consultation Support:** Providing real-time information or differential diagnoses to clinicians during consultations.
- **Medical Knowledge Access:** Acting as an AI consultant with access to a vast corpus of medical literature, guidelines, and research.

Responsible Innovation in Medicine

Given the high stakes, development in this area must prioritize safety and ethical considerations. Rigorous validation through retrospective analysis on historical data and,

critically, prospective clinical studies in real-world settings is essential before any widespread deployment impacting patient care. The focus is on creating human-centered AI that assists, rather than replaces, clinicians, emphasizing empathy, understanding, and collaboration. Med-PaLM is positioned as a step towards this vision, initially focusing on applications like medical question answering.

Med-PaLM Progress

- Med-PaLM was the first AI system reported to surpass the passing score on USMLE-style (US Medical Licensing Exam) questions, a benchmark for medical knowledge.
- Med-PaLM 2 demonstrated further improvements, achieving performance comparable to expert clinicians on these challenging exams.
- Evaluations also showed improvements in the quality, factual correctness, and reduced potential for harm in its long-form answers compared to general-purpose LLMs.

Measuring AI's Medical Knowledge: Evaluation Strategy

Evaluating medical LLMs requires a multi-faceted strategy:

- **Quantitative Benchmarks:** Using standardized exams like USMLE provides a measure of foundational medical knowledge. Datasets like MedQA are commonly used.
- **Qualitative Assessments:** Human experts (clinicians) evaluate model responses based on multiple dimensions:
 - o Factual correctness and alignment with medical consensus.
 - o Appropriate application of medical knowledge.
 - o Helpfulness and completeness of the answer.
 - o Readability and clarity.
 - o Evidence of potential bias.
 - o Potential for patient harm (a critical safety check).

Human Evaluations

A common methodology involves having both Med-PaLM and human physicians answer the same set of medical questions independently. These responses (anonymized and randomized) are then presented side-by-side to expert clinician raters who judge which response is superior based on the qualitative criteria mentioned above. This comparative evaluation provides insights into the model's strengths and weaknesses relative to human experts.

Areas for Improvement

Despite progress, current models still require significant improvement before widespread clinical use. Scoring well on standardized tests or offline datasets does not guarantee safe and effective performance in the complexities of real-world clinical practice. A careful progression of studies, from retrospective analysis to controlled prospective trials, is necessary to validate each specific application.

Task-Specific vs. Broad Domain Models

Med-PaLM's success highlights the benefits of domain specialization. While a broad medical foundation is crucial, specific clinical applications (e.g., radiology report summarization, dermatology image analysis) may require further task-specific fine-tuning and validation. The multimodal nature of medicine – integrating text (clinical notes, research), images (radiology, pathology), structured data (EHRs, lab results), sensor data, and genomics – presents a major frontier for future development.

Applications Beyond Patient Care

LLMs in medicine also hold promise for accelerating scientific discovery, such as identifying potential gene-disease associations or analyzing large cohorts for epidemiological insights.

Med-PaLM as a Suite of Commercially Available Models

Building on the research, Google aims to provide Med-PaLM 2 capabilities as commercially available models (likely via APIs on Vertex AI), enabling healthcare organizations and partners to build their own specialized GenAI solutions on a secure, compliant platform.

Med-PaLM 2 Training

Med-PaLM 2 builds upon the general PaLM 2 foundation model. Its specialization comes from:

- Extensive fine-tuning on medical domain data, particularly question-answering datasets (like MedQA).
- Instruction fine-tuning using curated examples relevant to medical tasks.
- Employing advanced prompting techniques during inference for multiple-choice questions, such as few-shot prompting and Chain of Thought (CoT) prompting to encourage step-by-step reasoning.
- Using techniques like self-consistency (generating multiple reasoning paths and taking a majority vote) and ensemble refinement (where the model considers its own generated explanations to improve its final answer) to boost accuracy and robustness.

Conclusion

Specialized LLMs like SecLM and Med-PaLM demonstrate the immense potential of tailoring generative AI for complex, high-stakes domains. In cybersecurity, SecLM aims to alleviate operational burdens, augment analyst capabilities, and enhance threat response. In healthcare, Med-PaLM focuses on responsibly improving information access, supporting clinical workflows, and ultimately contributing to better patient outcomes. Success in both fields hinges on deep domain expertise, targeted training, rigorous evaluation (including essential human oversight), responsible development practices, and close collaboration with domain practitioners. The development of these vertical-specific foundation models signals a future where AI becomes increasingly integrated into specialized professional workflows.

Day 5: Operationalizing Generative AI on Vertex AI using MLOps

Introduction

Generative AI, particularly foundation models and the agent systems built upon them, holds transformative potential across industries. However, moving from exciting prototypes to reliable, scalable, and maintainable real-world applications requires a disciplined operational framework. This chapter explores how established Machine Learning Operations (MLOps) principles must be adapted and extended to address the unique characteristics of generative AI systems, with a practical focus on leveraging Google Cloud's Vertex AI platform for this purpose.

Foundation Models vs. Traditional ML Models

Operationalizing generative AI differs significantly from managing traditional ML models (e.g., classifiers, regressors) due to the inherent nature of foundation models:

- **Multi-Purpose Nature:** Foundation models are often pre-trained for general capabilities and adapted for various tasks, unlike traditional models typically built for a single, specific purpose.
- **Emergent Properties:** They can exhibit capabilities they were not explicitly trained for, making their behavior sometimes unpredictable.
- **Prompt Sensitivity:** Their output is highly sensitive to the nuances of the input prompt, turning prompt engineering into a critical development and operational activity.
- **Adaptation over Training:** Development often focuses on adapting existing large models (via prompting, fine-tuning, RAG) rather than training models from scratch.

The Generative AI Lifecycle

Adapting the standard MLOps lifecycle for generative AI involves several key phases, each with unique considerations:

1. **Discover:** Identifying and selecting the most appropriate foundation model(s) for the task.

2. **Develop and Experiment:** Crafting prompts, potentially tuning the model, building chains or agentic systems, and preparing data.
3. **Evaluate:** Rigorously assessing the performance, safety, and reliability of the prompted model or system.
4. **Deploy:** Moving the system into a production environment.
5. **Monitor & Govern:** Continuously tracking performance, managing costs, ensuring compliance, and iterating based on feedback.

Discovery Phase

The sheer number of available foundation models (proprietary like Gemini, GPT-4; open-source like Llama, Mixtral) presents a significant selection challenge. Key factors include:

- **Quality:** Assessed through benchmark scores (e.g., HELM, MMLU), but more importantly, through performance on tasks representative of the target use case.
- **Latency:** Meeting the response time requirements of the application.
- **Cost:** Considering infrastructure needs (GPUs/TPUs), software licenses, and usage-based pricing.
- **Legal & Compliance:** Understanding model licenses, data usage policies, and regulatory constraints.

Solution: Platforms like **Vertex AI Model Garden** provide a curated catalog of models, offering "model cards" that detail performance metrics, intended use cases, limitations, and other relevant metadata to aid selection.

Development and Experimentation Phase

This phase involves creating the core components of the generative AI application.

Prompted Model Components:

At its simplest, a generative AI application consists of a foundation model combined with a specific prompt.

- **Prompt Engineering:** Crafting effective prompts is crucial. Prompt templates offer structured ways to combine static instructions, dynamic inputs (user queries), and potentially few-shot examples.
- **Dual Nature of Prompts:** Prompts act as both:
 - o *Data:* Few-shot examples, retrieved context (RAG), user queries provide factual input.
 - o *Code:* Instructions, role definitions, output format specifications, guardrails define the task logic.
- **MLOps Implication:** Prompts require rigorous version control, testing, and tracking, similar to application code. It's vital to know which prompt version works best with which model version.

Chaining and Augmentation:

To overcome the limitations of single model calls (e.g., knowledge cutoffs, lack of real-time data), developers often chain multiple prompted model components together or augment them with external tools and custom logic.

- **Retrieval Augmented Generation (RAG):** A common pattern where a retrieval system (often using vector search) fetches relevant information from an external knowledge base (e.g., documents, databases) to provide context to the LLM, grounding its responses and reducing hallucinations.
- **Agents:** More sophisticated systems where an LLM acts as a reasoning engine, capable of using various tools (APIs, code interpreters, databases) sequentially or in parallel to accomplish complex tasks.
- **MLOps Implications:**
 - o Evaluation must assess the entire chain or agent system end-to-end, not just individual components.
 - o Versioning needs to encompass all parts: models, prompts, retrieval components, tools, and orchestration logic.

- o Defining expected input distributions becomes harder due to the complexity and variability of natural language interactions.

Tuning and Training:

While often starting with pre-trained models, some adaptation might be necessary.

- **Supervised Fine-Tuning (SFT):** Training the model on labeled datasets specific to the target task or domain.
- **Reinforcement Learning from Human Feedback (RLHF):** Using human preferences to train a reward model, which then guides the LLM's fine-tuning process to better align its outputs with desired characteristics (e.g., helpfulness, harmlessness).
- **MLOps Considerations:**
 - o Requires meticulous tracking of datasets, hyperparameters, tuning procedures, and resulting model performance metrics.
 - o Due to the high cost of training/tuning large models, *continuous tuning* (periodically updating the model) might be more practical than *continuous training* (constantly retraining).
 - o Techniques like model quantization (reducing numerical precision) can help manage computational costs during tuning and deployment, potentially with minor trade-offs in performance.

Data Practices for Generative AI

Data management in the generative AI lifecycle has unique aspects:

- **Diverse Data Types:** Involves managing prompts, few-shot examples, RAG grounding data (documents, vector embeddings), user feedback data (for RLHF or evaluation), evaluation datasets, and potentially synthetic data generated by models themselves.
- **Rapid Prototyping:** Initial prototypes can often be built with less curated data than traditional ML, relying heavily on the base model's capabilities and prompt engineering.

- **Synthetic Data Generation:** LLMs can be used to generate varied prompts, sample responses, or even evaluation data, although the quality and representativeness of synthetic data require careful validation.
- **Challenges:** Managing these diverse data sources requires robust data governance and versioning. The unknown nature of the initial training data for proprietary foundation models complicates drift detection. Creating high-quality, custom evaluation datasets that accurately reflect specific use cases and failure modes is critical.

Evaluation

Evaluating generative AI systems is complex and evolves as projects mature:

- **Challenges:** Outputs are often high-dimensional (text, code, images), making simple metrics insufficient. Defining "good" performance can be subjective and context-dependent. Ground truth data for comparison is often unavailable for generative tasks.
- **Progression:** Early stages may rely heavily on manual, qualitative evaluation. As the system matures, automated processes become essential for scalability and consistency.
- **Approaches:**
 - o **Model-Based Evaluation (Auto-Eval):** Using capable LLMs (e.g., Gemini, GPT-4) as evaluators, providing them with specific criteria (rubrics) to score the target model's output (e.g., assessing factual accuracy, coherence, safety, creativity). Requires careful calibration and validation against human judgment.
 - o **Adversarial Testing:** Probing the system with challenging or malicious inputs (prompt injection, requests for harmful content) to test robustness and safety guardrails.
 - o **Custom Metrics:** Defining metrics tailored to the specific use case (e.g., code execution success rate for code generation, factual consistency score for RAG systems, adherence to specific style guides).

Deployment

Deploying generative AI typically involves deploying an entire *system* rather than just a single model artifact.

- **Complexity:** Requires managing dependencies between models, prompts, external data sources (for RAG), vector databases, and orchestration logic.
- **Best Practices:**
 - Robust version control for all components (models, prompts, code, data schemas).
 - CI/CD (Continuous Integration/Continuous Deployment) pipelines adapted for the unique testing and deployment needs of GenAI systems (e.g., including prompt testing, RAG component testing, end-to-end evaluation).
 - Scalable solutions for managing external data sources (e.g., BigQuery for structured data, Vertex AI Feature Store for embeddings or features, vector databases for RAG).
- **Foundation Model Deployment:** Handling the massive size of foundation models requires specialized infrastructure (GPUs/TPUs), potentially model compression techniques (quantization, distillation), and scalable serving platforms like **Vertex AI Prediction**.

Monitoring and Logging

Continuous monitoring is vital for maintaining performance and reliability in production.

- **Requirements:** Need end-to-end tracking across potentially chained components or agent tool calls.
- **Key Concepts:**
 - **Skew Detection:** Comparing the distribution of evaluation data (used during development) with the distribution of live production data to detect mismatches that could impact performance.
 - **Drift Detection:** Monitoring changes in the characteristics of input data (prompts, user queries) over time.

- o **Continuous Evaluation:** Capturing a sample of production inputs and outputs for ongoing assessment using automated or human evaluation processes.
- o **Tracing:** Recording the flow of events and data through the system (e.g., which prompts were used, which tools were called by an agent, what data was retrieved by RAG) to understand component interactions and debug failures. **Vertex AI Tracing** can assist here.

Governance

Governance extends beyond the model itself to encompass the entire system.

- **Scope:** Includes managing versions and access controls for prompts, RAG data sources, tool APIs, evaluation datasets, and model artifacts.
- **Implementation:** Leverages standard MLOps and DevOps practices, supported by tools for metadata management (**Vertex ML Metadata**), experiment tracking (**Vertex Experiments**), and data governance (**Dataplex**) to ensure reproducibility, compliance, and responsible AI practices.

Agent Ops: The Next Frontier

Operationalizing AI agents introduces further complexities beyond standard generative AI MLOps.

Unique Challenges:

- **Autonomy:** Agents make decisions and take actions (potentially with real-world consequences) often without direct human oversight in each step.
- **External Interactions:** Agents interact with numerous external systems (APIs, databases, code interpreters), increasing the surface area for potential failures or unexpected behavior.
- **Trust Requirements:** The autonomy and potential impact necessitate exceptionally robust governance, monitoring, safety guardrails, and control mechanisms.

Key Concepts:

- **Tool Orchestration:** Managing the selection, execution, and monitoring of the various tools an agent can use.
- **Tool Registry:** A centralized catalog for discovering, managing, versioning, and controlling access to available tools.
- **Tool Selection Strategies:** Determining how agents choose tools:
 - *Generalist:* Access to all available tools.
 - *Specialist:* Limited to a predefined set of tools relevant to its specific function.
 - *Dynamic:* Runtime selection based on relevance scoring or retrieval.
- **Evaluation and Optimization:** Requires a multi-stage process, from unit testing individual tools to evaluating the agent's ability to orchestrate tools effectively for complex tasks, ultimately measuring operational metrics and business KPIs. Observability (understanding *what* the agent did) and explainability (understanding *why*) are critical.
- **Memory Management:** Designing and managing the agent's short-term (conversation history) and long-term (persistent knowledge from past interactions) memory effectively and efficiently.

Deployment Considerations:

- Requires robust CI/CD pipelines that include automated testing of tools and agent logic.
- Automated tool registration and management within the registry.
- Continuous monitoring specifically focused on tool usage patterns, error rates, and agent decision-making quality.
- An iterative improvement loop incorporating feedback from monitoring and evaluation to refine agent behavior and tool usage.

The Changing MLOps Landscape

Generative AI is reshaping the roles and tools within the MLOps ecosystem:

- **New Roles:** Emergence of roles like Prompt Engineers, AI Engineers (focusing on system integration), alongside traditional DevOps and ML Engineers.
- **Unified Platforms:** Platforms like **Vertex AI** aim to provide comprehensive, integrated tooling across the entire lifecycle, from data preparation and model discovery to deployment, monitoring, and governance, streamlining the operationalization process.
- **Future Challenges:** The rapid pace of innovation in foundation models, agent architectures, and evaluation techniques means MLOps practices must continuously evolve to keep pace.

Conclusion

Successfully operationalizing generative AI requires a thoughtful adaptation of existing MLOps principles combined with new strategies tailored to the unique characteristics of foundation models, prompts, RAG systems, and autonomous agents. While the challenges are significant, platforms like Vertex AI provide the necessary infrastructure, tools, and integrated workflows to build, deploy, monitor, and govern these powerful systems effectively and responsibly. Embracing a robust MLOps culture is key for organizations seeking to unlock the full, sustainable potential of generative AI in real-world applications.

References:

All "full versions" of Google Whitepapers covered during the training can be found on the links below:

Day 1:

<https://www.kaggle.com/whitepaper-foundational-llm-and-text-generation>

<https://www.kaggle.com/whitepaper-prompt-engineering>

Day 2:

<https://www.kaggle.com/whitepaper-embeddings-and-vector-stores>

Day 3:

<https://www.kaggle.com/whitepaper-agents>

<https://www.kaggle.com/whitepaper-agent-companion>

Day 4:

<https://www.kaggle.com/whitepaper-solving-domains-specific-problems-using-llms>

Day 5:

<https://www.kaggle.com/whitepaper-operationalizing-generative-ai-on-vertex-ai-using-mlops>