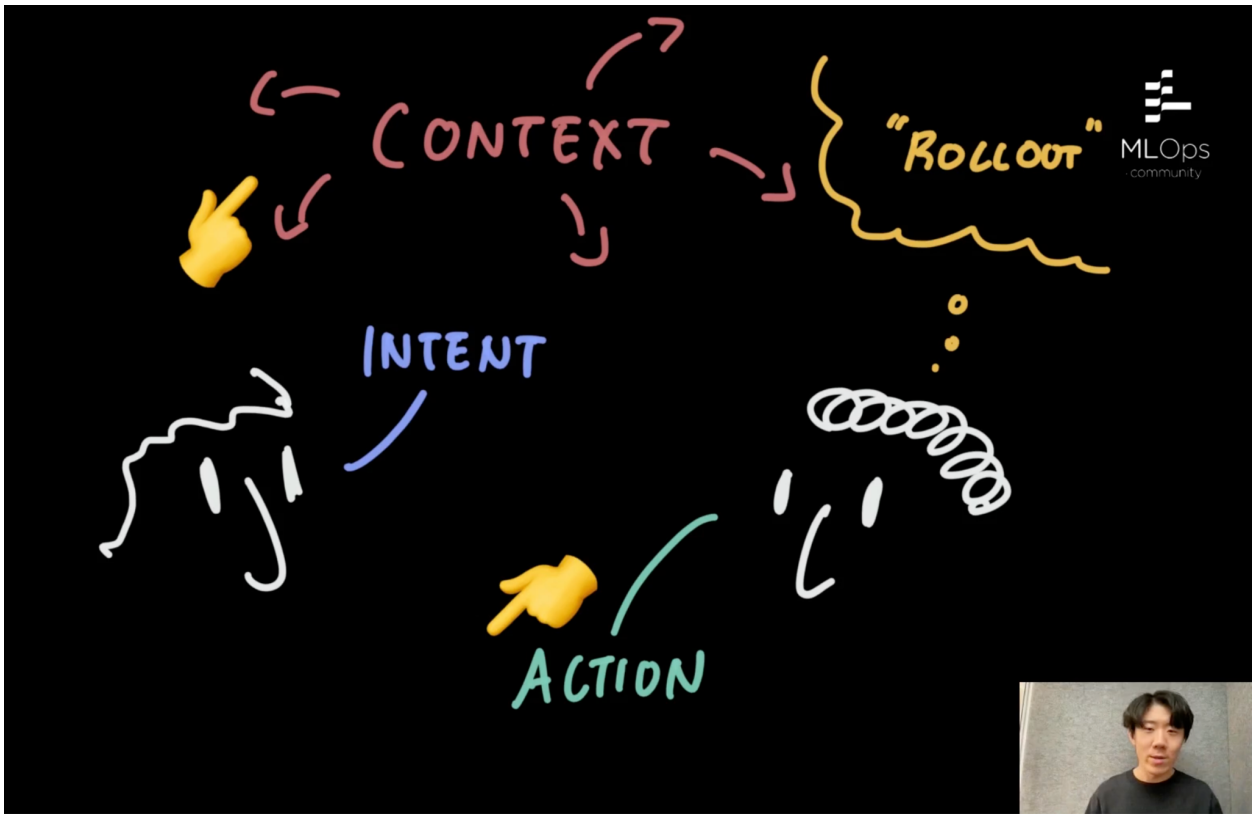
Anatomy of a "turn"

- Context in which to interpret the user's intent & decide
- Intent, like instructions or questions
- Rollout = interpretation (chain of thought, scratchpad)
- Action or response



Point-and-select

- Nouns, then verbs
- In the real world, "built in" to every material
- What makes the web great!
- Help clarify context or direct actions

Constraints & guardrails

Predictive actions

Intuitive, but flexible

Model capabilities

Tool use

Fall back to chat

MLOps community

Open-ended natural language UI trades off intuitiveness for flexibility.

|

"Learning curve"

Ease of discovery

Progressive onboarding

Closing feedback loops

— Respond with a range of outputs (if easy to evaluate)

— "Shopping" over "Creation"

— Interactive components?

> *The possibility space should be narrow enough to exclude broken artifacts (such as models that fall over or break when 3D printed) but broad enough to contain surprising artifacts as well. The surprising quality of the artifacts motivates the user to explore the possibility space in search of new discoveries, a motivation which disappears if the space is too uniform.*
>
> *— Kate Compton, Casual Creators*

Good dialogue interfaces should...

MLOps community

1. Have agents that coinhabit your workspace
2. Take full advantage of rich shared context
3. Lead with constrained "happy path" actions
4. Fall back to chat as an escape hatch
5. Speed up iteration for fast feedback loops

Outline

- *Why "foundation models" (not LLMs)?*
- Some predictions for the near future of FMs: **GPT-You, not GPT-X**
- Data-centric development of FMs
- *(If time) Other challenges & opportunities*

Goal: Provide both high level perspective on FMs + data-centric development, as well as a look at how we support @ Snorkel AI

Closed APIs aren't defensible- and OSS is booming!

Stanford Alpaca

Recent results show the transferability of FM data moats, the power of OSS base models (and the power of data-centric development)

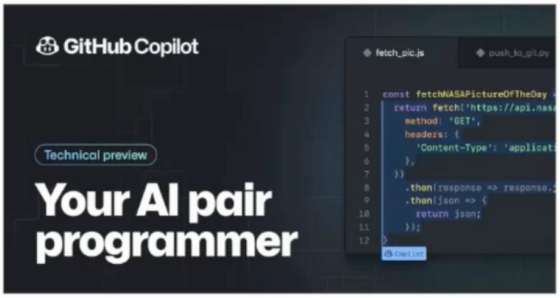
Case Study 1: Github Co-pilot

MLO
·commur

One of the most successful LLM Apps

- > 1mn Users and growing fast
- Trusted by cantankerous programmers

How is it built? How do they make it defensible?



The Anatomy of an LLM App

LLM Block:

1. Base-model (often finetuned)
2. Prompt-template
3. Data selection strategy

Often repeated or used in sequence or as an agent

Case Study: Github Co-pilot

Much more than a GPT-3 wrapper:

1. Custom fine-tuned 12Bn param model GPT
2. Complex Strategy for including user content
3. Careful capture of evaluation data
4. Regular fine-tuning gives them a network effect.

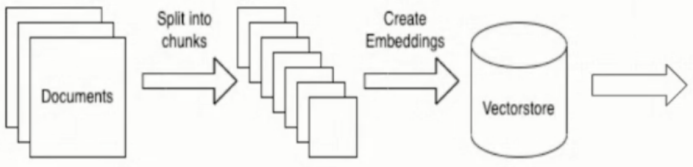
Source:
https://thakkarparth007.github.io/copilot-explorer/posts/copilot-internals.html

Challenges to building a strong LLM app

1. **Prompt Engineering is still an art** — small changes in prompt have a big outcome on performance
2. **LLMs confidently bullshit** — Need to add guardrails to make them more factual
3. **Evaluation is hard —** Often very subjective so testing is difficult.
4. **Latency and Cost —** Cloud provider and large model latency can be a problem for realtime apps
5. **Going beyond thin moat —** If you're app is just GPT + a prompt competition will be fierce

Getting LLMs to be Factual — Tools

Augment the LLM with retrieval via embeddings or APIs



{X} QA with vector look up

You're the world's best legal assistant. Answer the following question.

Use the context to help:
Question: {{query}}
context: {{ legal_search(query, top_k) }}

Answer:

When to use Prompt Engineering vs Finetuning

Prompt Engineering:

- Very fast to change
- Less extra work to upgrade when underlying models improve
- Can get good performance if you experiment
- Easier to add fast changing factual content

Finetuning:

- Have access to a special dataset (e.g user data) and want better performance
- Improved latency (10x or more)
- Improved cost
- Trying to achieve a particular tone
- Local/private model

Strategies for making LLM Apps Defensible

1. Integrate deeply with private knowledge sources (switching costs)
2. Build a data flywheel through feedback capture and finetuning (network effect)
3. Focus on private applications where public LLMs are harder to use (counter-positioning)
4. Build amazing UX — less theoretically sound but important in practice

What do we mean by agentic *relationship management*?

In this world, agents are AI systems personalized to:

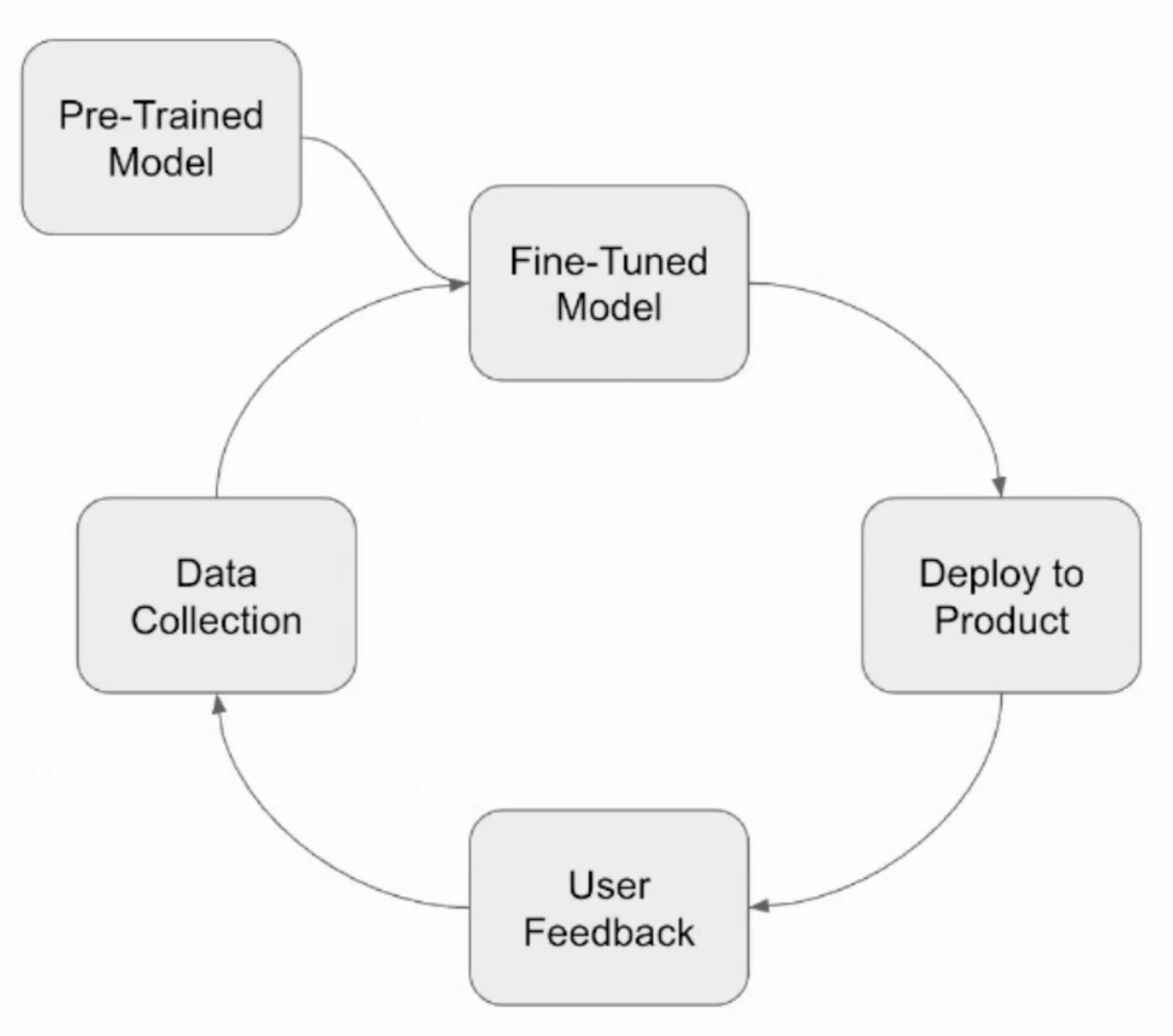
- a persona
- a team
- an individual

through communication and social network integrations.

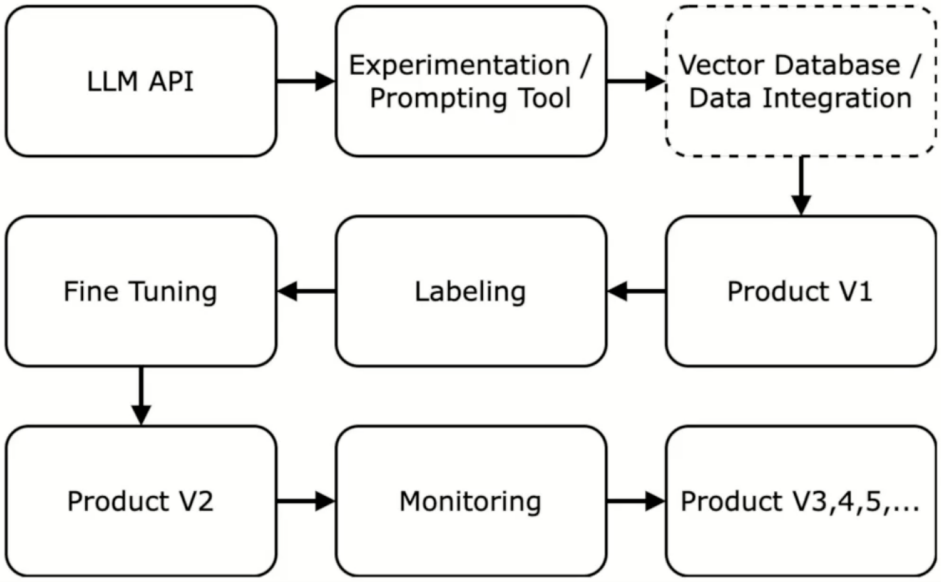
Their objective is to maximize different network workflows.

The world of workflows could look like:

- Which of my relationships are growing cold?
- I'm in a new city, who should I meet with?
- Who are the people in this sales meeting?
- What is my network news today?
- Auto-add conference attendees to my socials


Pre-Trained Model
Fine-Tuned Model
Deploy to Product
User Feedback
Data Collection

The Development Process with LLMs



Orchestration, Experimentation and Prompting Tools

- LLMs “APIs” are natural language in form of prompts
- Mastering the API requires tinkering and experimenting with single and chained prompts
- Various tools have emerged that provide ability:
  - to connect to data sources,
  - provide indices,
  - coordinate chains of calls
  - other core abstractions

LlamaIndex

Cognosis DUST

LangChain

FIXIE

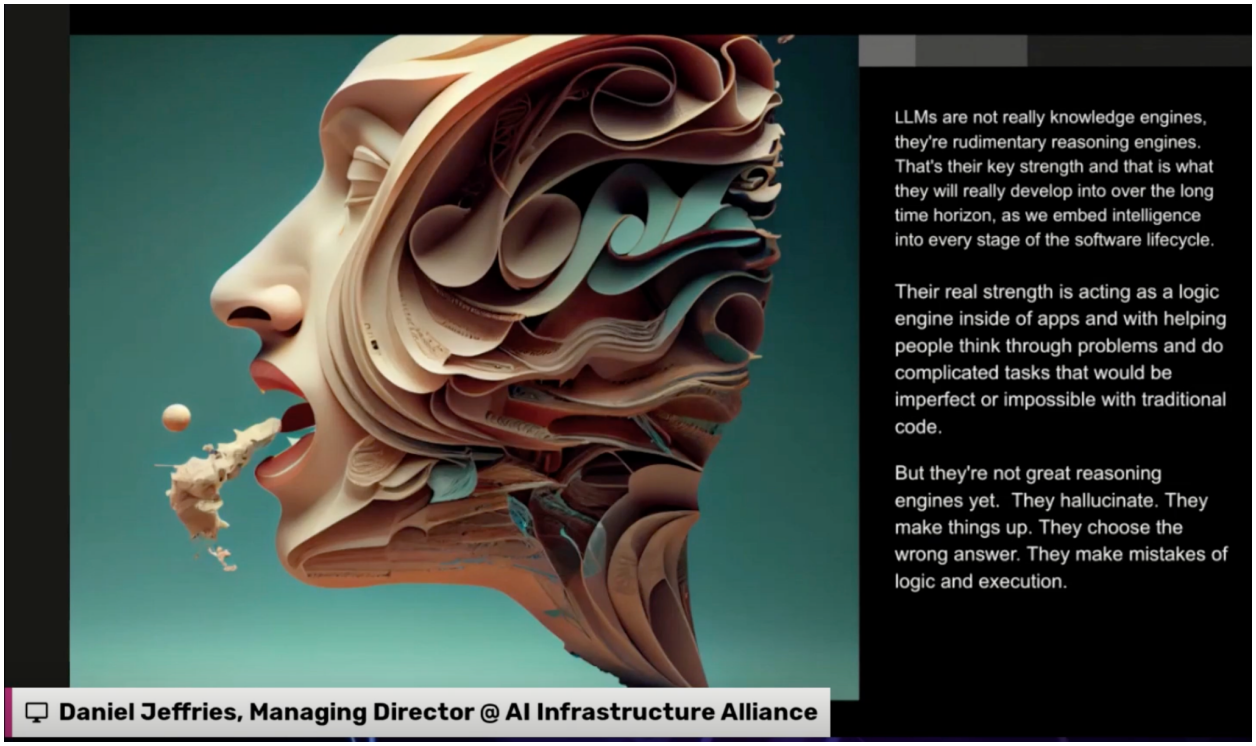
Monitoring, Observability and Testing

HoneyHive

HELM

- LLM Performance: Unique challenge; assess quality via user interactions.
- A/B Testing: Evaluate LLM features via product analytics (full workflow)
- Eye Test vs. HELM: as OSS gains traction comparison frameworks will become more critical.
- Performance Impact: Affects UX; guides model selection and fine-tuning.

Seeing the new character gets her inspired. She quickly knocks out a story outline in **meta-story language** and feeds it to her story iterator **LTM**. It generates 50 completed versions of the story in a few seconds.


LLMs are not really knowledge engines, they're rudimentary reasoning engines. That's their key strength and that is what they will really develop into over the long time horizon, as we embed intelligence into every stage of the software lifecycle.
Their real strength is acting as a logic engine inside of apps and with helping people think through problems and do complicated tasks that would be imperfect or impossible with traditional code.
But they're not great reasoning engines yet. They hallucinate. They make things up. They choose the wrong answer. They make mistakes of logic and execution.
Daniel Jeffries, Managing Director @ AI Infrastructure Alliance

But nobody is really fixing these bugs fast enough or at all.

To fix them we're going to need a suite of new tools and new strategies. To get there we've got to break down the problem so we know where to start. Projects and companies looking to make a difference can look at it like this:

Where can the LLM break down? A few major areas:

1. In the middle:
    1. At the prompt point, aka in between the model and the user
    2. In between the model and other models/tools/software
    3. At the output, aka after the LLM responce
    4. During the workflow, aka DAG breakdowns
2. During training and/or fine-tuning:

Let's look at each in turn.

Reality: Training LLMs is accessible!

LLM Training Costs on MosaicML Cloud			
Model	Billions of Tokens (Compute-optimal)	Days to Train on MosaicML Cloud	Approx. Cost on MosaicML Cloud
GPT-1.3B	26B	0.14	$2,000
GPT-2.7B	54B	0.48	$6,000
GPT-6.7B	134B	2.32	$30,000
GPT-13B	260B	7.43	$100,000
GPT-30B *	**610B**	**35.98**	**$450,000**
GPT-70B **	1400B	176.55	$2,500,000

← Many business use cases

← GPT-3 Quality

← Chinchilla Qua

Focus on

- Projects with proprietary data
- Projects with "subjective" machine learning, e.g. similarity ML
- Avoid plain vanilla projects
- Machine learning projects with specific requirements (e.g. user privacy, security, low latency)
- Be the moderator and advisor between/to stakeholders

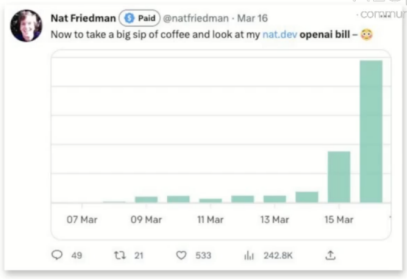
Tipping the Scales in your Favour

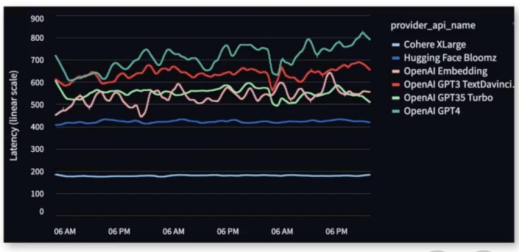
- **Focus on deploying to non-critical workflows:** Add value, don't become a dependency.
- **Find relatively higher latency use-cases**: User expectations unlikely to change overnight - integrating LLMs in low latency scenarios raises required value creation.
- **Plan to build while you buy**: Reduce upfront investment and project uncertainties by derisking LLM component by leveraging foundational model APIs to build out MVPs, but have a clear plan to store data and train in-house model as adoption scales.
- **Don't underestimate HCI component**: Responding *seemingly* faster, failing gracefully, enabling large-scale user feedback are all vital aspects.

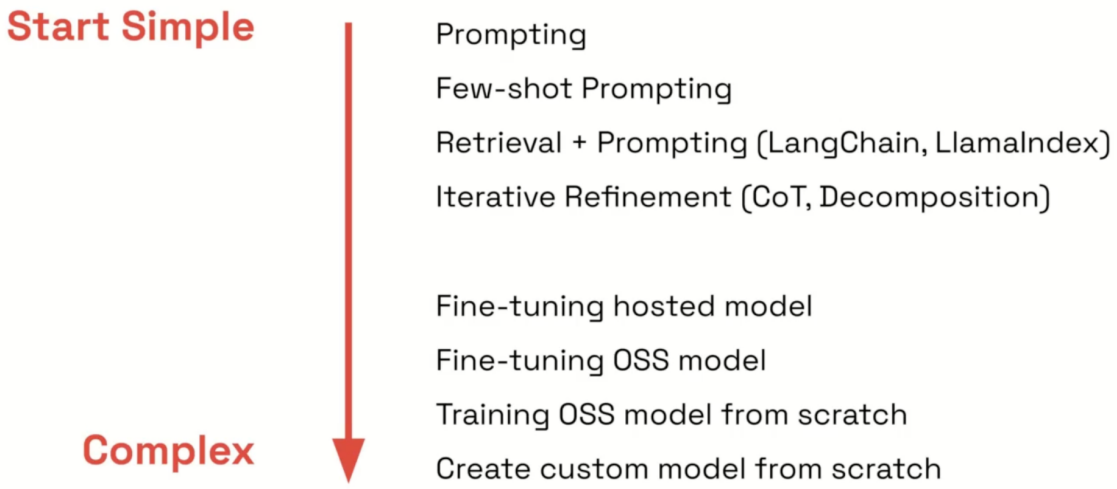
Why LLMs present new challenges in production

- (38.09%) Reliability (Hallucination)
- (28.12%) Cost
- (26.65%) Latency
- (7.14%) Safety

(percentages = AI builders that consider this a problem)







Reliability: Add structure

Give me the ingredients for a **grilled cheese sandwich**

You must comply with the following Typescript schema

```
type IngredientSchema = {
    ingredient: string;
    quantity: number;
    unit: string;
};
```

```
[
 { "ingredient": "bread",   "quantity": 2,   "unit": "slices" },
 { "ingredient": "cheese",  "quantity": 1,   "unit": "slices" }
]
```

How?

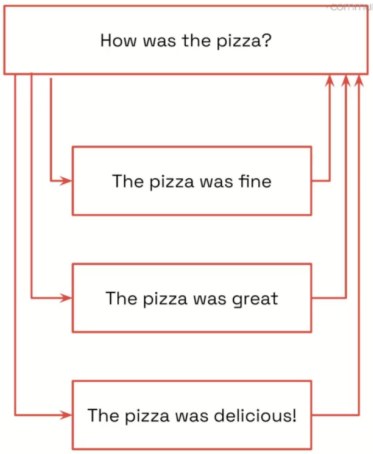
- **Logic:** Responses must match Typescript schema
- **Demonstration:** For example...
- **Persona:** You are a chatbot that speaks perfect Typescript
- **Remind:** Remember always return JSON!

Failure?

- Re-ask
- Ramp up temperature or model

Accuracy: Self-refinement

- Model evaluates its own output.
- No supervised data or reinforcement needed.
- Outperforms baseline in many cases

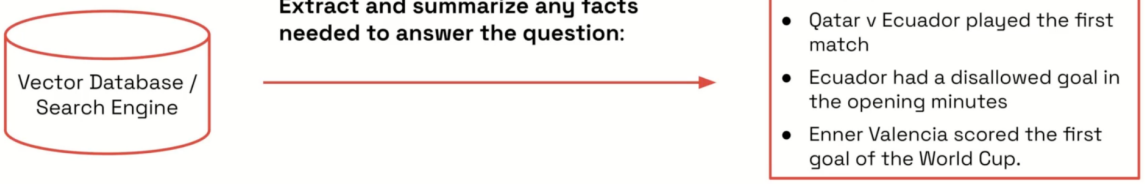


Metric	Dataset	Base LLM	SELF-REFINE
Solve Rate	Math Reasoning	71.3	**76.2**
Coverage	Constrained Generation	4.0	**22.5**
% Programs Optimized	Code Optimization	9.7	**15.6**
% Readable Variables	Code Readability	37.4	**51.3**
	Dialogue Response	27.2	**37.6**
Human Eval.	Sentiment Reversal	15.3	**84.7**
	Acronym Generation	11.8	**23.5**

SELF-REFINE: ITERATIVE REFINEMENT WITH SELF-FEEDBACK

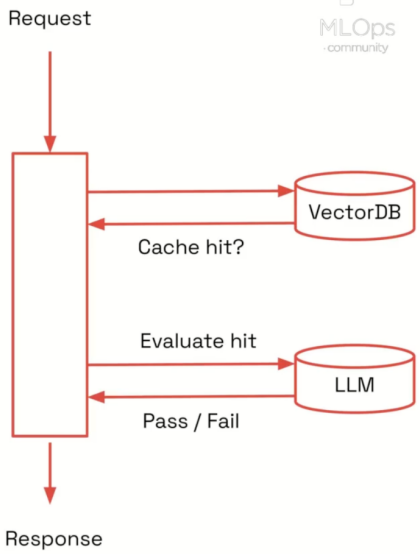
Accuracy: Contextual Compression

- Context window limits the amount of external data
- Compression allows significantly more examples
- Take task into account (question)



Latency: Semantic Caching

- Semantic caching uses vector database for caching
- Cache hits are evaluated by semantic similarity or by LLM
- When does this work best?
  - Completions are expensive (many trips to LLM)
  - Selection problems (tools, categories)



https://github.com/zilliztech/GPTCache

Different types of tools

- Search engines
- Calculators
- Retrieval systems
- Python REPLs
- Arbitrary functions
- **APIs**

Challenge 1: Getting them to use tools in the right scenario

Solutions

1. Instructions/System message
2. Tool description
3. Repeat instructions at the end
4. Tool retrieval

Challenge 3:
Parsing LLM Output to tool invocation

Solutions

1. More structured responses (JSON)
2. Output Parsers
3. Fix output parsing errors

Prompting:

- TypeScript/JSON to communicate the parameters

Composability:

- Wrap each endpoint in its own “chain”, have the “agent” call that chain