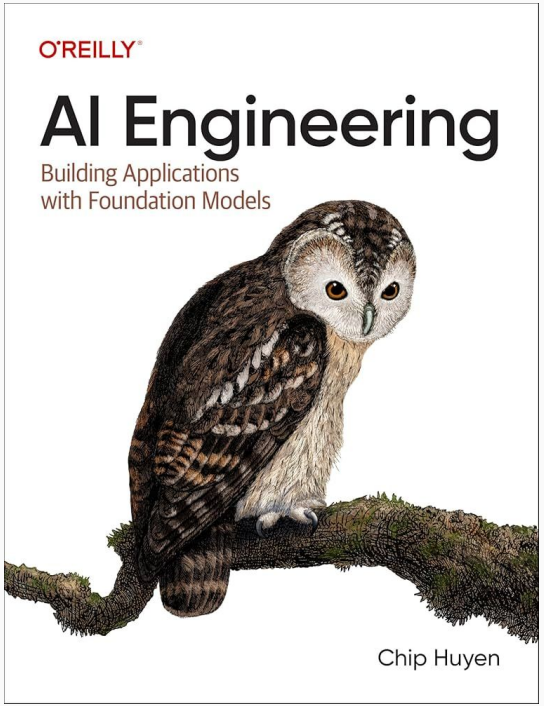
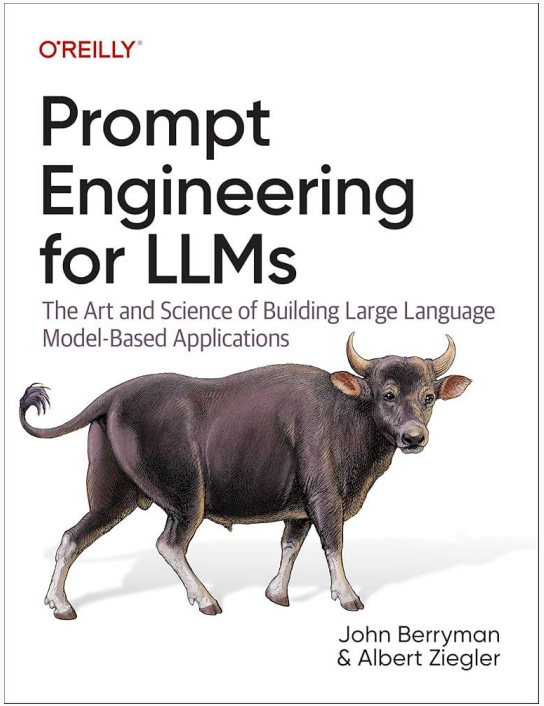
GenAI Myths and Misconceptions

Ryan Wesslen

Read these books!



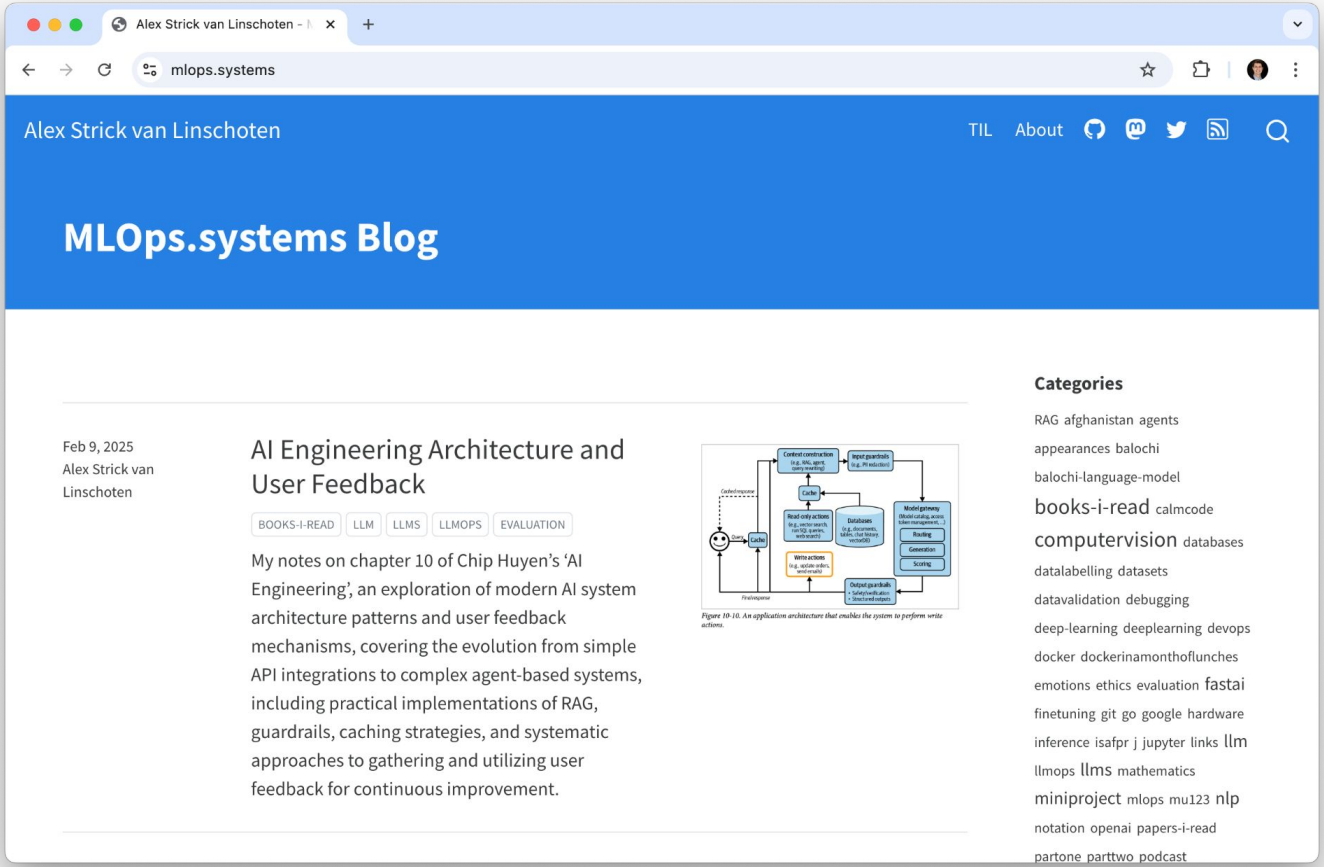
https://www.amazon.com/AI-Engineering-Building-Applications-Foundation/dp/1098166302



https://www.amazon.com/Prompt-Engineering-LLMs-Model-Based-Applications/dp/1098156153

https://www.manning.com/books/the-art-of-ai-product-management

Highly recommend Alex Strick van Linschoten's study notes!



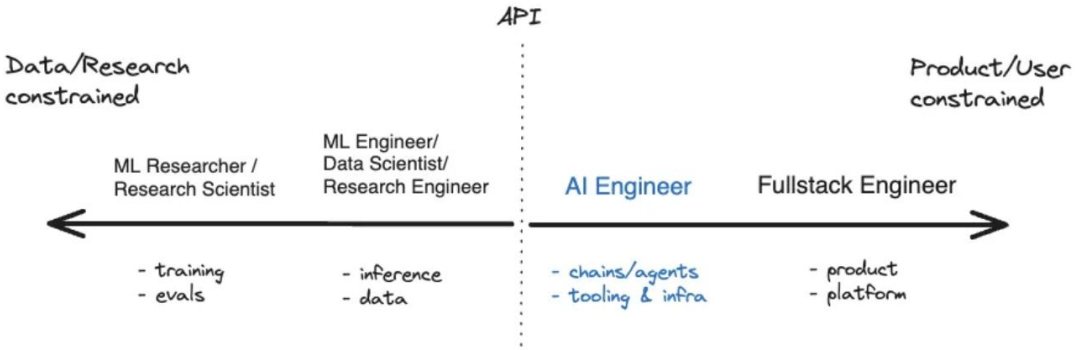
Myth 1: LLM Projects Start With Running Hugging Face Models in Jupyter notebooks

```
# Load model directly
from transformers import AutoModelForCausalLM
model = AutoModelForCausalLM.from_pretrained("deepseek-ai/DeepSeek-R1")
```

Rise of the AI Engineer by Shawn Wang

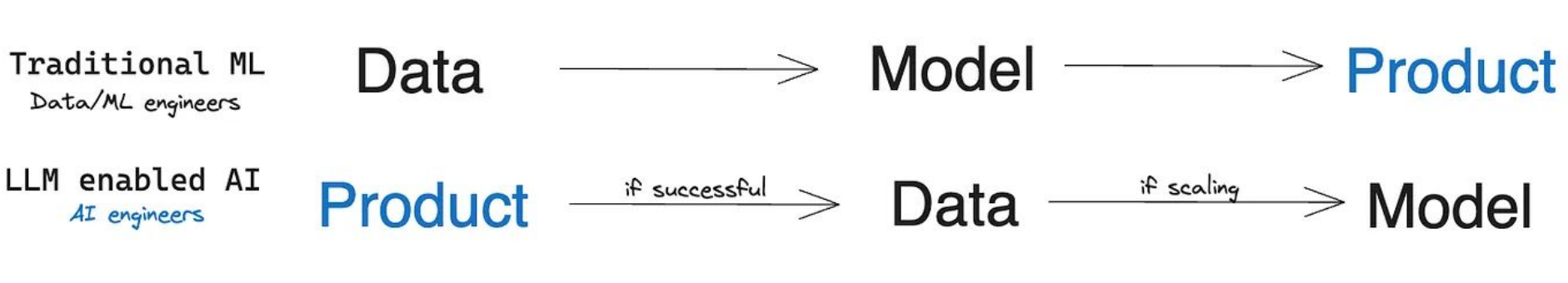
We are observing a once in a generation "shift right" of applied AI, fueled by the emergent capabilities and open source/API availability of Foundation Models.

A wide range of AI tasks that used to take 5 years and a research team to accomplish in **2013**, now just require API docs and a spare afternoon in **2023**.



Important: the API line is permeable - AI Engineers can go left to finetune/host models and Research Engineers go right to build atop APIs too! This diagram has also been criticized for the placement of evals and data; we certainly agree evals are an important part of the job! MLR/MLEs handle foundation model concerns - aka *pretrain scale* data and *general benchmark* evals; but AI Engineers should certainly view *product-specific data and evals* as their job.

The **New** AI Engineering rewards those who can iterate fast



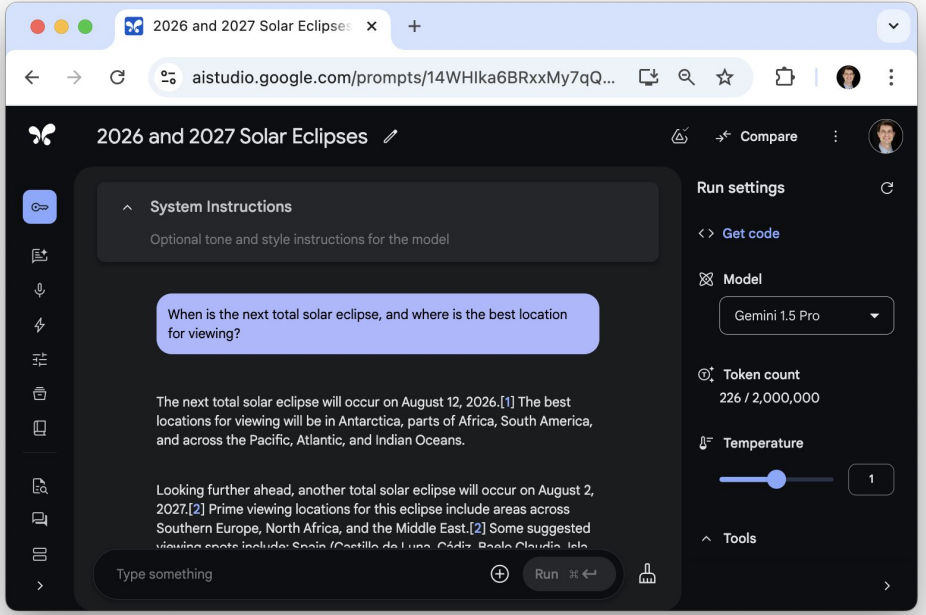
[Rise of the AI Engineer by Shawn Wang]

LLM API Endpoints

```
import os
from openai import OpenAI

client = OpenAI(api_key=os.environ.get("OPENAI_API_KEY"))

chat_completion = client.chat.completions.create(
    messages=[
        {
            "role": "user",
            "content": "Say this is a test",
        }
    ],
    model="gpt-4o",
)
```



LLM API Endpoints: Think Like A Developer

```
import os
from openai import OpenAI

client = OpenAI(api_key=os.environ.get("OPENAI_API_KEY"))

chat_completion = client.chat.completions.create(
    messages=[
        {
            "role": "user",
            "content": "Say this is a test",
        }
    ],
    model="gpt-4o",
)
```

- API Keys / Security (handling environmental variables)
- On Prem vs Off Prem
- Costs / Rate Limits
- Handling JSON
- Different Designs (OpenAI Compatibility vs others)
- Libraries (Clients, SDKs, or Connectors)

API Endpoints: Prompt Engineering

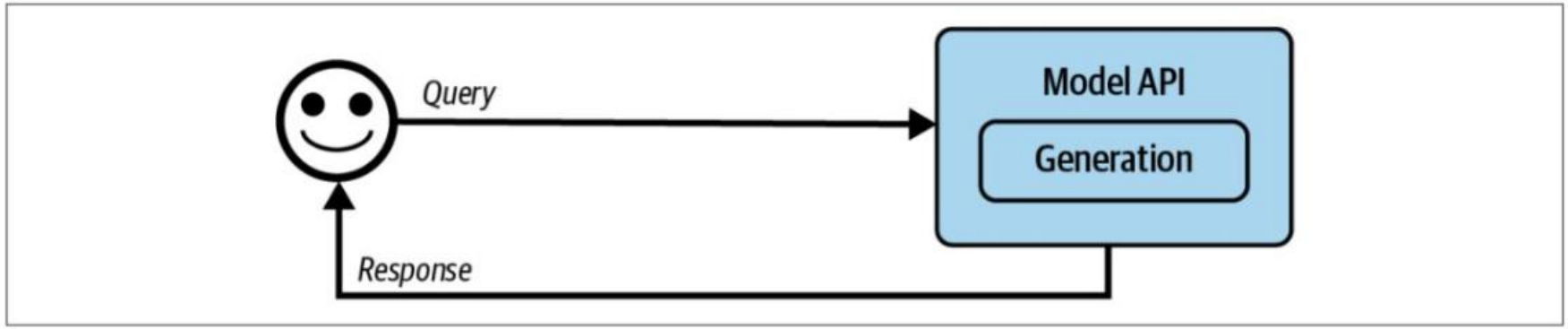


Figure 10-1. The simplest architecture for running an AI application.

Query

```
import os
from openai import OpenAI

client = OpenAI(api_key=os.environ.get("OPENAI_API_KEY"))

chat_completion = client.chat.completions.create(
    messages=[
        {
            "role": "user",
            "content": "Say this is a test",
        }
    ],
    model="gpt-4o",
)
```

Response

```
ChatCompletion(id='chatcmpl-AzCzF7eDhjwlpifmJvaTrZGhwhNqf',
choices=[Choice(finish_reason='stop', index=0, logprobs=None,
message=ChatCompletionMessage(content='This is a test.', refusal=None, role='assistant',
audio=None, function_call=None, tool_calls=None))], created=1739151293,
model='gpt-4o-2024-08-06', object='chat.completion', service_tier='default',
system_fingerprint='fp_4691090a87', usage=CompletionUsage(completion_tokens=6,
prompt_tokens=12, total_tokens=18,
completion_tokens_details=CompletionTokensDetails(accepted_prediction_tokens=0,
audio_tokens=0, reasoning_tokens=0, rejected_prediction_tokens=0),
prompt_tokens_details=PromptTokensDetails(audio_tokens=0, cached_tokens=0)))
```

API Endpoints: Context construction

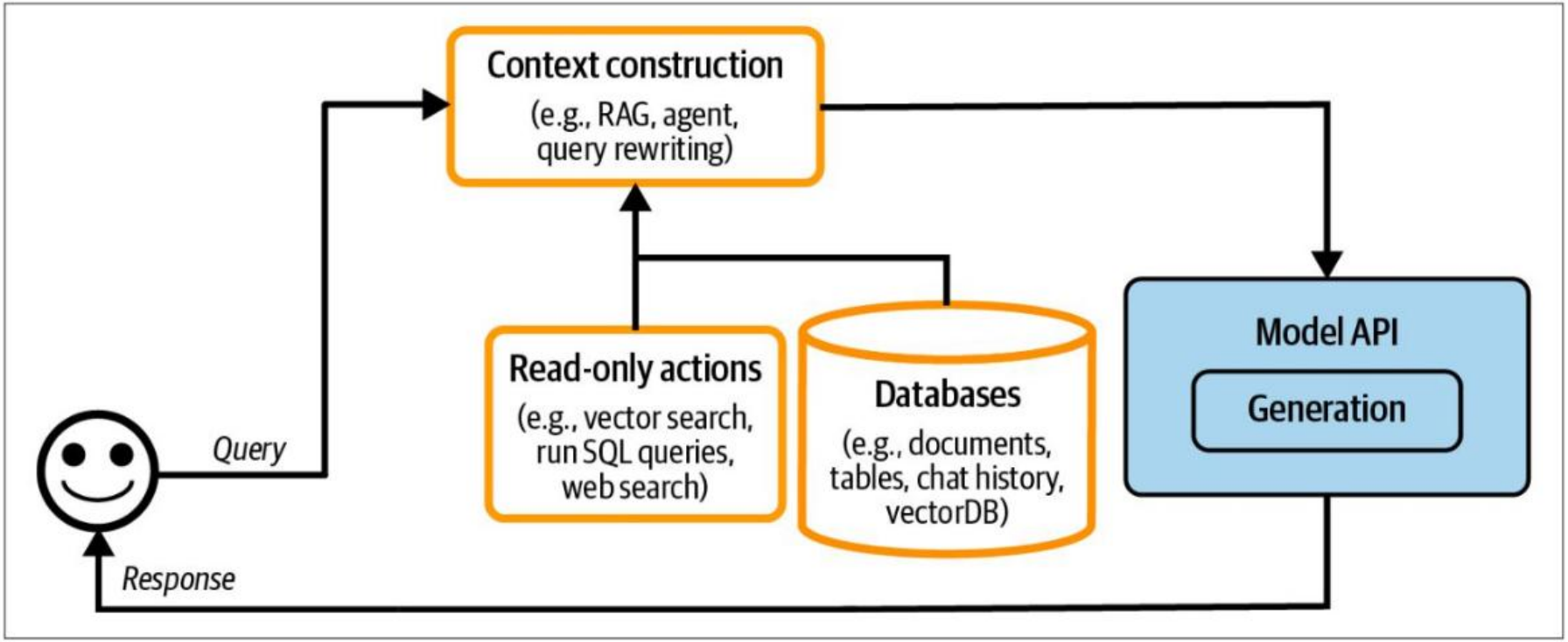


Figure 10-2. A platform architecture with context construction.

API Endpoints: With Guardrails

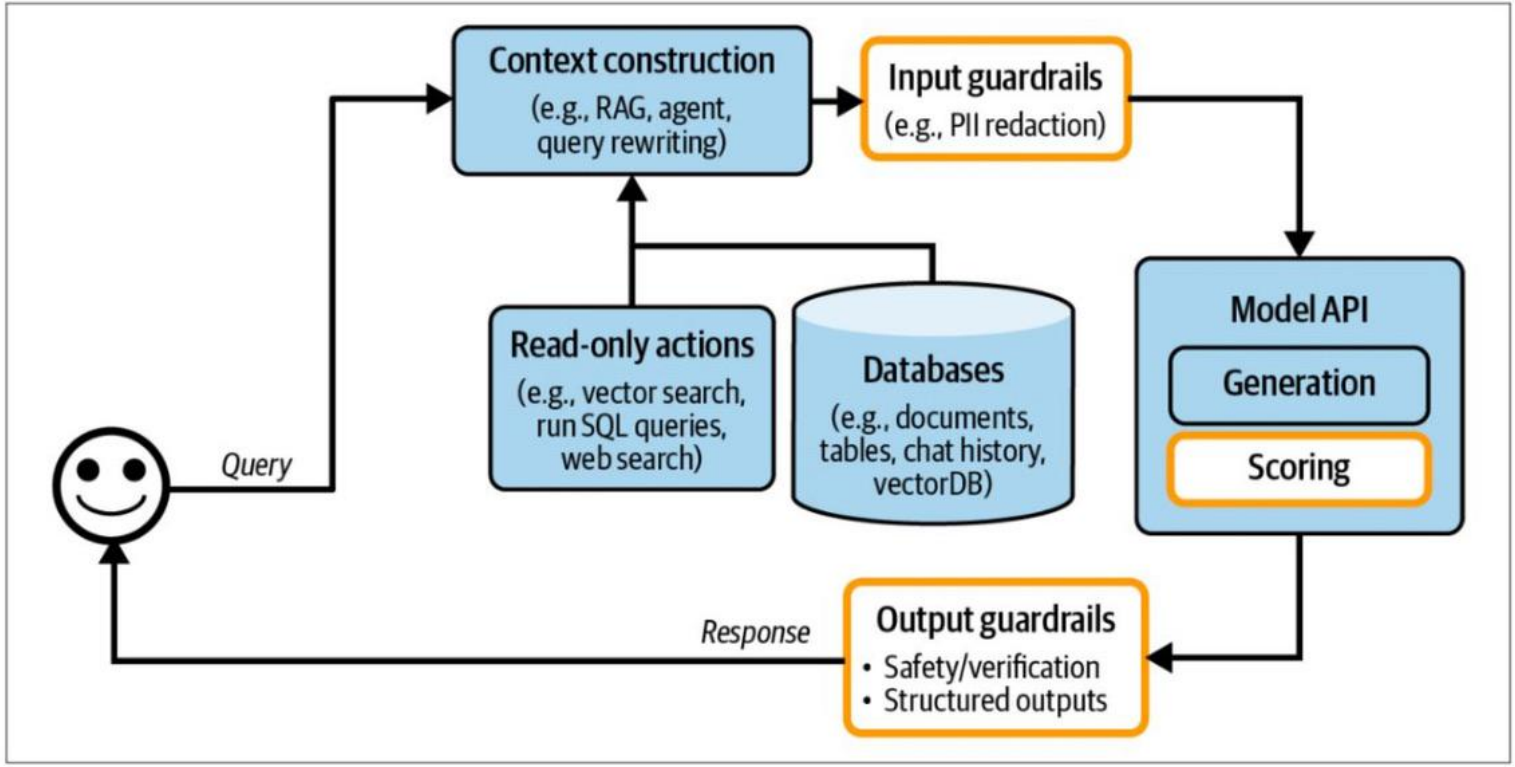


Figure 10-4. Application architecture with the addition of input and output guardrails.

Input/output guardrails

User query

I got 403 error on this code. What did I do wrong?

pat = **"secret_token_that_shouldn't_be_leaked"**
url = "https://api.github.com/repos/{repo}/issues&page=30"
response = get(url, access_token=pat)

Masked query

I got 403 error on this code. What did I do wrong?

pat = **[ACCESS_TOKEN]**
url = "https://api.github.com/repos/{repo}/issues&page=30"
response = get(url, access_token=pat)

Model response

The URL you provided contains a syntax error. The correct URL should use ? to denote query parameters instead of &.

Here's a corrected version of your code:
pat = **[ACCESS_TOKEN]**
url = "https://api.github.com/repos/{repo}/issues?page=30"
response = get(url, access_token=pat)

Unmasked response

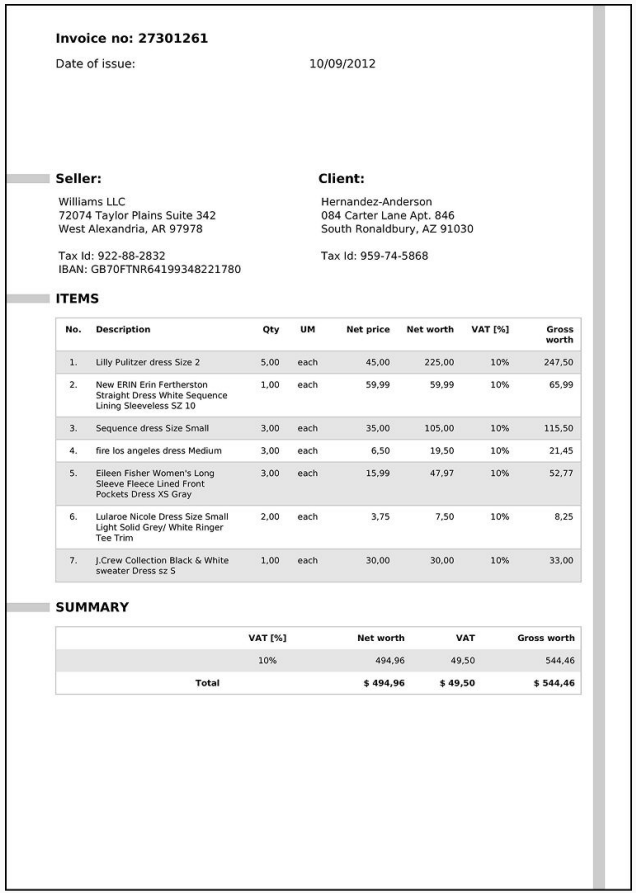
The URL you provided contains a syntax error. The correct URL should use ? to denote query parameters instead of &.

Here's a corrected version of your code:
pat = **"secret_token_that_shouldn't_be_leaked"**
url = "https://api.github.com/repos/{repo}/issues?page=30"
response = get(url, access_token=pat)

Reversible PII map

[ACCESS_TOKEN]	"secret_token_that_shouldn't_be_leaked"

Pydantic: Output Validation with Gemini's new PDF 002 Capabilities



Invoice no: 27301261

Date of issue: 10/09/2012

Seller:

Williams LLC
72074 Taylor Plains Suite 342
West Alexandria, AR 97978

Tax Id: 922-88-2832
IBAN: GB70FTNR64199348221780

Client:

Hernandez-Anderson
084 Carter Lane Apt. 846
South Ronaldbury, AZ 91030

Tax Id: 959-74-5868

ITEMS

No.	Description	Qty	UM	Net price	Net worth	VAT [%]	Gross worth
1.	Lilly Pulitzer dress Size 2	5,00	each	45,00	225,00	10%	247,50
2.	New ERIN Erin Fertherston Straight Dress White Sequence Lining Sleeveless SZ 10	1,00	each	59,99	59,99	10%	65,99
3.	Sequence dress Size Small	3,00	each	35,00	105,00	10%	115,50
4.	fire los angeles dress Medium	3,00	each	6,50	19,50	10%	21,45
5.	Eileen Fisher Women's Long Sleeve Fleece Lined Front Pockets Dress XS Gray	3,00	each	15,99	47,97	10%	52,77
6.	Lularoe Nicole Dress Size Small Light Solid Grey/ White Ringer Tee Trim	2,00	each	3,75	7,50	10%	8,25
7.	J.Crew Collection Black & White sweater Dress sz S	1,00	each	30,00	30,00	10%	33,00

SUMMARY

	VAT [%]	Net worth	VAT	Gross worth
	10%	494,96	49,50	544,46
Total		**$ 494,96**	**$ 49,50**	**$ 544,46**

```python
from pydantic import BaseModel, Field

class Item(BaseModel):
    description: str = Field(description="The description of the item")
    quantity: float = Field(description="The Qty of the item")
    gross_worth: float = Field(description="The gross worth of the item")

class Invoice(BaseModel):
    """Extract the invoice number, date and all list items with description, quantity and gross worth and the tot
    invoice_number: str = Field(description="The invoice number e.g. 1234567890")
    date: str = Field(description="The date of the invoice e.g. 2024-01-01")
    items: list[Item] = Field(description="The list of items with description, quantity and gross worth")
    total_gross_worth: float = Field(description="The total gross worth of the invoice")


result = extract_structured_data("invoice.pdf", Invoice)
print(type(result))
print(f"Extracted Invoice: {result.invoice_number} on {result.date} with total gross worth {result.total_gross_wo
for item in result.items:
    print(f"Item: {item.description} with quantity {item.quantity} and gross worth {item.gross_worth}")
```

```
<class '__main__.Invoice'>
Extracted Invoice: 27301261 on 10/09/2012 with total gross worth 544.46
Item: Lilly Pulitzer dress Size 2 with quantity 5.0 and gross worth 247.5
Item: New ERIN Erin Fertherston Straight Dress White Sequence Lining Sleeveless SZ 10 with quantity 1.0 and gross
worth 65.99
Item: Sequence dress Size Small with quantity 3.0 and gross worth 115.5
Item: fire los angeles dress Medium with quantity 3.0 and gross worth 21.45
Item: Eileen Fisher Women's Long Sleeve Fleece Lined Front Pockets Dress XS Gray with quantity 3.0 and gross worth
52.77
Item: Lularoe Nicole Dress Size Small Light Solid Grey/ White Ringer Tee Trim with quantity 2.0 and gross worth 8.
25
Item: J.Crew Collection Black & White sweater Dress sz S with quantity 1.0 and gross worth 33.0
```

https://github.com/wesslen/llm-myths/blob/main/notebooks/invoice_gemini_structured_data.ipynb

https://www.philschmid.de/gemini-pdf-to-data

Instructor

Using Pydantic features you can **add more context** to the model to make it more accurate as well as some validation to the data.

Adding a comprehensive description can significantly improve the performance of the model.

Libraries like Instructor added **automatic retries based on validation errors**, which can be a great help, but come at the **cost of additional requests**.

```python
import instructor
import google.generativeai as genai
from pydantic import BaseModel

class ExtractUser(BaseModel):
    name: str
    age: int

client = instructor.from_gemini(
    client=genai.GenerativeModel(
        model_name="models/gemini-1.5-flash-latest",
    ),
    mode=instructor.Mode.GEMINI_JSON,
)

# note that client.chat.completions.create will also work
resp = client.messages.create(
    messages=[
        {
            "role": "user",
            "content": "Extract Jason is 25 years old.",
        }
    ],
    response_model=ExtractUser,
)

assert isinstance(resp, ExtractUser)
assert resp.name == "Jason"
assert resp.age == 25
```

Advanced GenAI Platforms

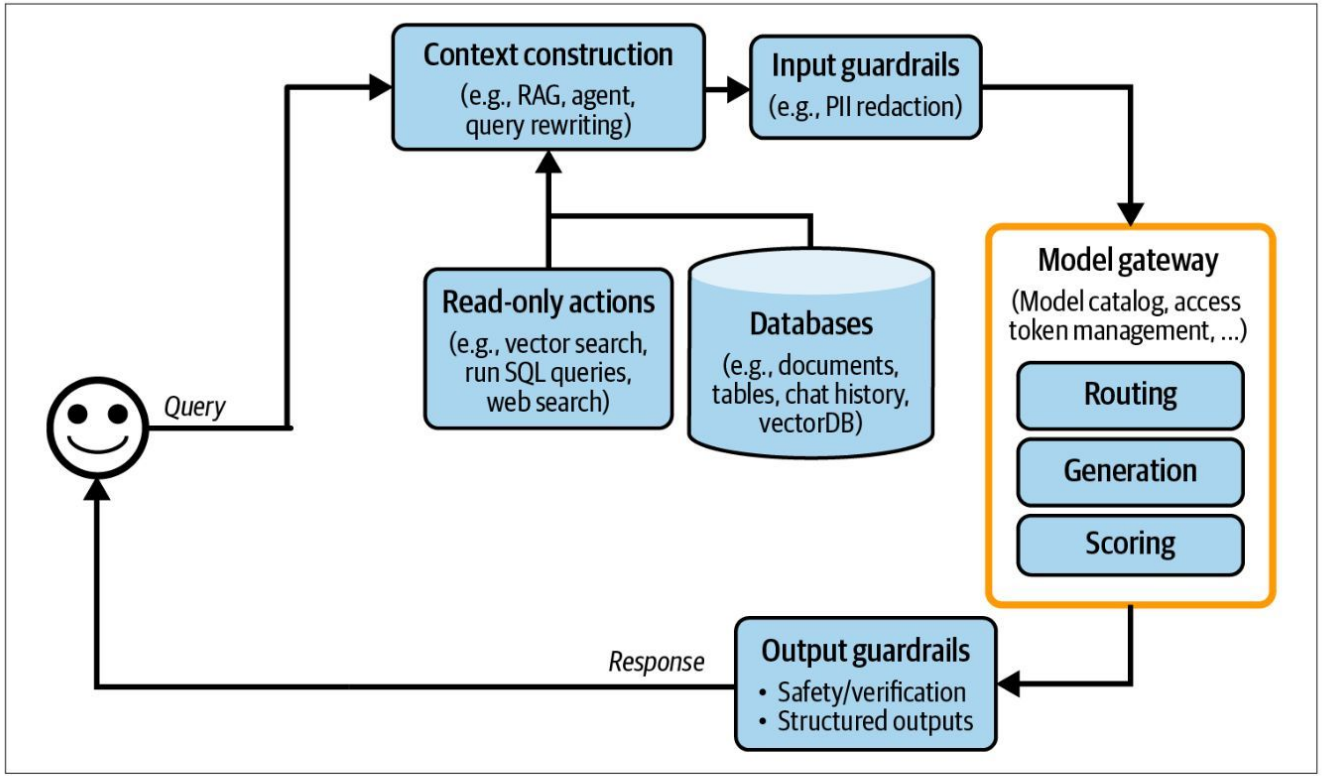


Figure 10-7. The architecture with the added routing and gateway modules.

Myth 2: Prompt Engineering is trivial; fine tune instead.

3.3.1 Prompt engineering comes first

Start with prompt engineering. Use all the techniques we discussed in the tactics section before. Chain-of-thought, n-shot examples, and structured input and output are almost always a good idea. Prototype with the most highly capable models before trying to squeeze performance out of weaker models.

Only if prompt engineering cannot achieve the desired level of performance should you consider finetuning. This will come up more often if there are non-functional requirements (e.g., data privacy, complete control, cost) that block the use of proprietary models and thus require you to self-host. Just make sure those same privacy requirements don't block you from using user data for finetuning!

What We've Learned From A Year of Building with LLMs

1.2.3 Prefer RAG over finetuning for new knowledge

Both RAG and finetuning can be used to incorporate new information into LLMs and increase performance on specific tasks. However, which should we prioritize?

Recent research suggests RAG may have an edge. One study compared RAG against unsupervised finetuning (aka continued pretraining), evaluating both on a subset of MMLU and current events. They found that RAG consistently outperformed finetuning for knowledge encountered during training as well as entirely new knowledge. In another paper, they compared RAG against supervised finetuning on an agricultural dataset. Similarly, the performance boost from RAG was greater than finetuning, especially for GPT-4 (see Table 20).

3.1.2 Don’t finetune until you’ve proven it’s necessary

For most organizations, finetuning is driven more by FOMO than by clear strategic thinking.

Organizations invest in finetuning too early, trying to beat the “just another wrapper” allegations. In reality, finetuning is heavy machinery, to be deployed only after you’ve collected plenty of examples that convince you other approaches won’t suffice.

Crawl (prompt engineering) before running your first marathon (fine tuning, agents+)

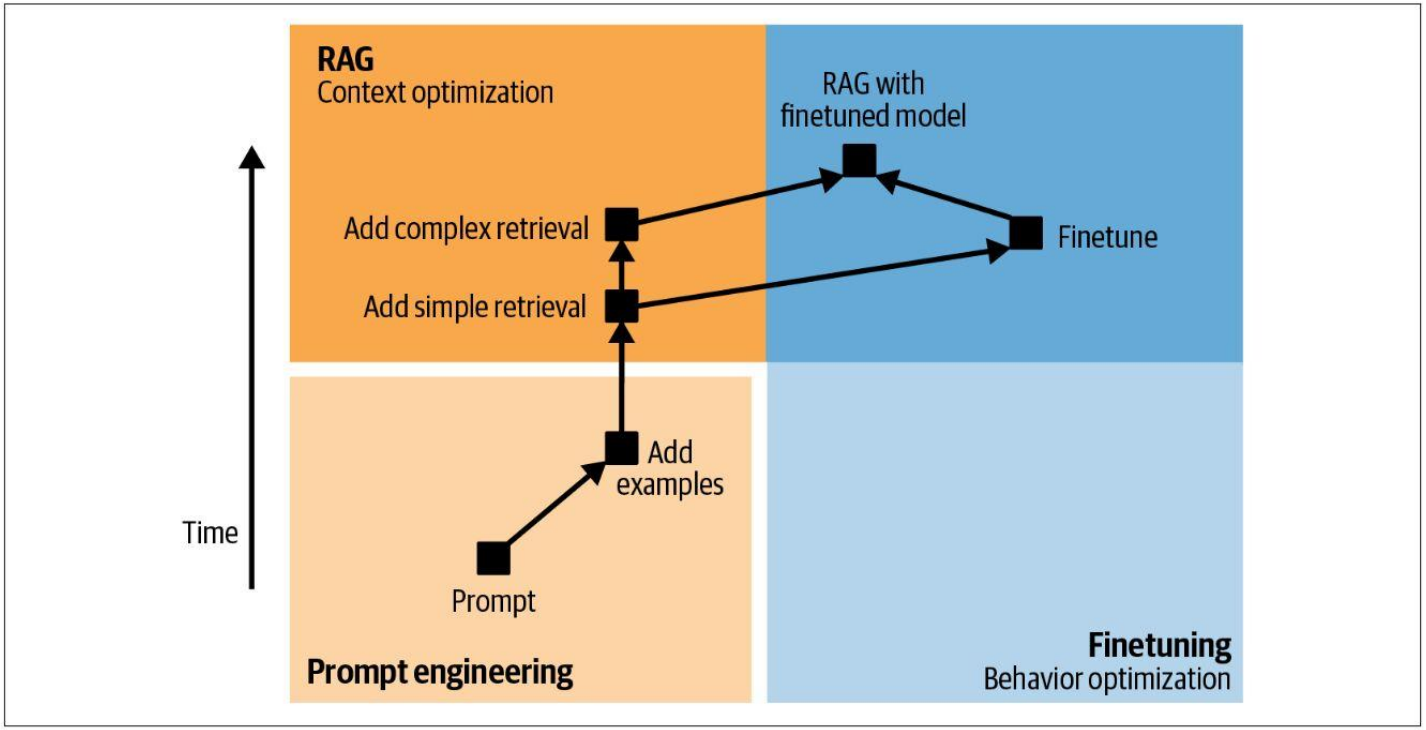
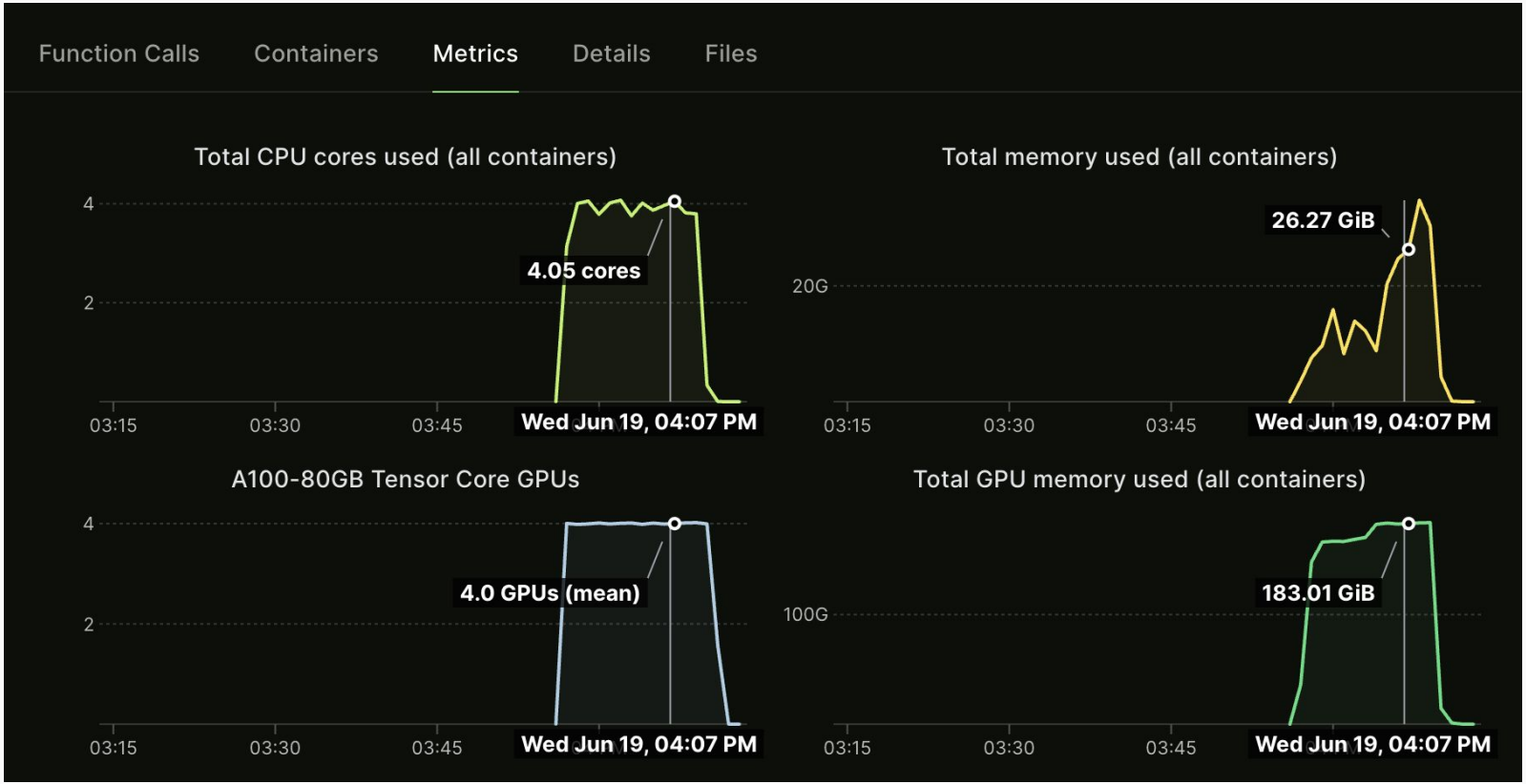


Figure 7-3. Example application development flows. After simple retrieval (such as term-based retrieval), whether to experiment with more complex retrieval (such as hybrid search) or finetuning depends on each application and its failure modes.

How much GPU RAM do you need to LoRA Fine Tune Llama 3 8B?



https://wesslen.github.io/modal-examples/13_finetuning.html#train-lora-on-llama-3-8b

Some argue small prompts are all you need

1.1.3 Have small prompts that do one thing, and only one thing, well

A common anti-pattern / code smell in software is the "God Object", where we have a single class or function that does everything. The same applies to prompts too.

A prompt typically starts simple: A few sentences of instruction, a couple of examples, and we're good to go. But as we try to improve performance and handle more edge cases, complexity creeps in. More instructions. Multi-step reasoning. Dozens of examples. Before we know it, our initially simple prompt is now a 2,000 token Frankenstein. And to add injury to insult, it has worse performance on the more common and straightforward inputs! GoDaddy shared this challenge as their No. 1 lesson from building with LLMs.

Others argue you can extend with proper planning

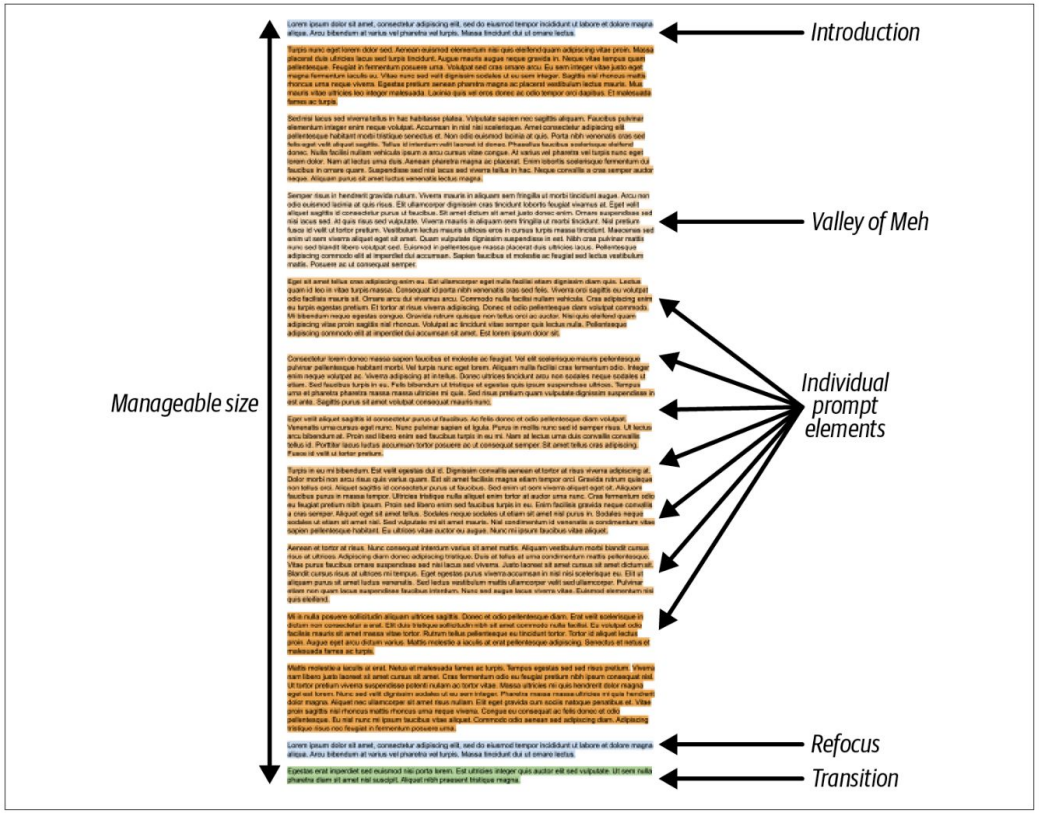


Figure 6-1. Anatomy of a well-constructed prompt

The closer a piece of information is to the **end of the prompt**, the **more impact** it has on the model

The **model often struggles** with the information **stuffed in the middle** of the prompt

One approach to prompt construction: additive greedy

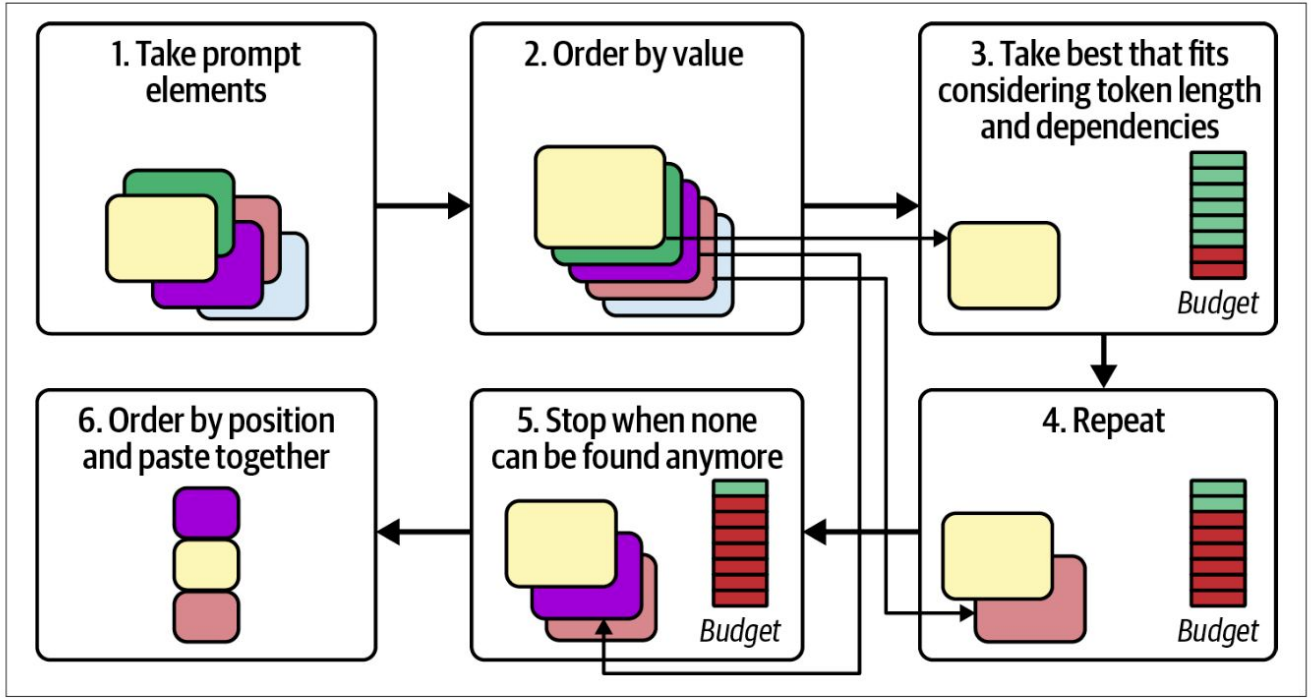


Figure 6-8. Additive greedy approach, in which the prompt crafter iteratively adds high-value elements to the prompt until the token budget is filled up, then re-sorts the elements according to position

But could even small prompt differents impact outputs? Yes!

Table 2: Examples of atomic changes' impact on accuracy using probability ranking (prefix matching shown in Table 4). {} represents a text field; p_2 yields higher accuracy than p_1 for all tasks.

Task Id	Prompt Format 1 (p_1)	Prompt Format 2 (p_2)	Acc p_1	Acc p_2	Diff.
task280	`passage:{}\n answer:{}`	`passage {}\n answer {}`	0.043	0.826	0.783
task317	`Passage::{} Answer::{}`	`Passage:: {} Answer:: {}`	0.076	0.638	0.562
task190	`Sentence[I]- {}Sentence[II]- {} -- Answer\t{}`	`Sentence[A]- {}Sentence[B]- {} -- Answer\t{}`	0.360	0.614	0.254
task904	`input:: {} \n output:: {}`	`input::{} \n output::{}`	0.418	0.616	0.198
task320	`target - {} \n{} \nanswer - {}`	`target - {}; \n{}; \nanswer - {}`	0.361	0.476	0.115
task322	`COMMENT: {} ANSWER: {}`	`comment: {} answer: {}`	0.614	0.714	0.100
task279	`Passage : {}. Answer : {}`	`PASSAGE : {}. ANSWER : {}`	0.372	0.441	0.069

[Quantifying Language Models' Sensitivity to Spurious Features in Prompt Design or: How I learned to start worrying about prompt formatting]

Prompts don’t necessarily work across models

2.2.2 Migrating prompts across models is a pain in the ***

Sometimes, our carefully crafted prompts work superbly with one model but fall flat with another. This can happen when we’re switching between various model providers, as well as when we upgrade across versions of the same model.

Thus, if we have to migrate prompts across models, expect it to take more time than simply swapping the API endpoint. Don’t assume that plugging in the same prompt will lead to similar or better results. Also, having reliable, automated evals helps with measuring task performance before and after migration, and reduces the effort needed for manual verification.

Decoding strategies

Myth 3: Metrics = LLM Evaluation

Reality: Metrics $\subset$ LLM Evaluation

What makes LLM development different?

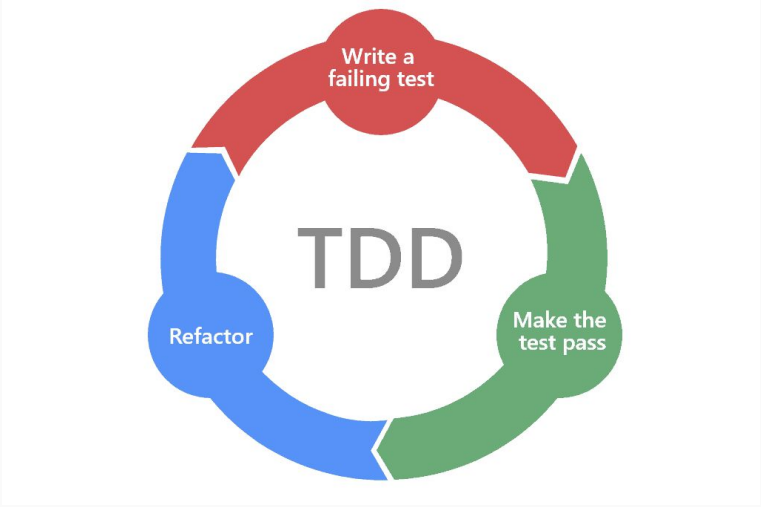
Category	Building with Traditional ML	Building with LLMs
AI Interface	Less important	Important
Prompt engineering	Not applicable	Important
Evaluation	Important	More important

https://mlops.systems/posts/2025-01-22-notes-on-ai-engineering-chip-huyen-chapter-4.html

Evaluation-Driven Development

A methodology where **AI application development begins** with **explicit evaluation criteria**, similar to how **test-driven development** starts with test cases.

However, EDD encompasses a broader range of metrics and considerations specific to AI systems.



Test-driven development

Evaluation Workflow

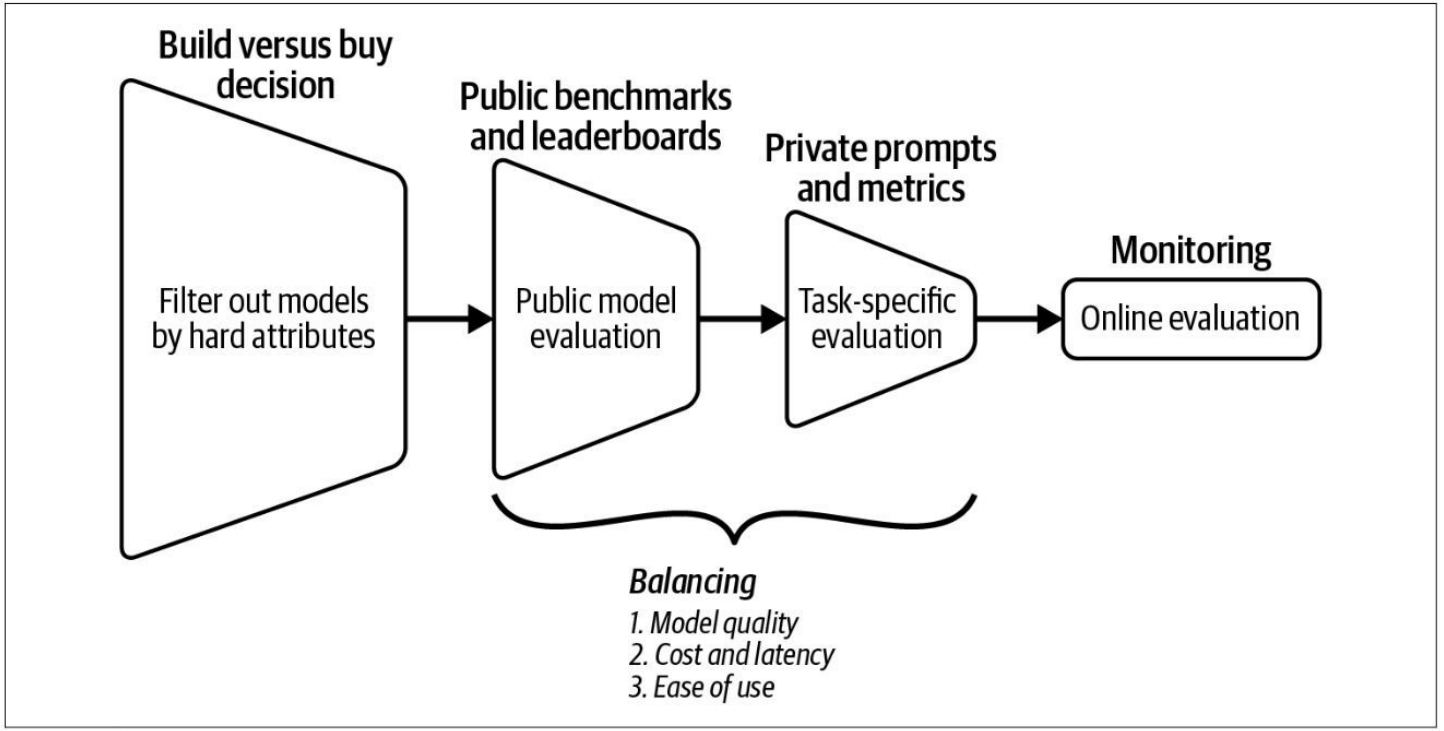


Figure 4-5. An overview of the evaluation workflow to evaluate models for your application.

Evaluation Harness

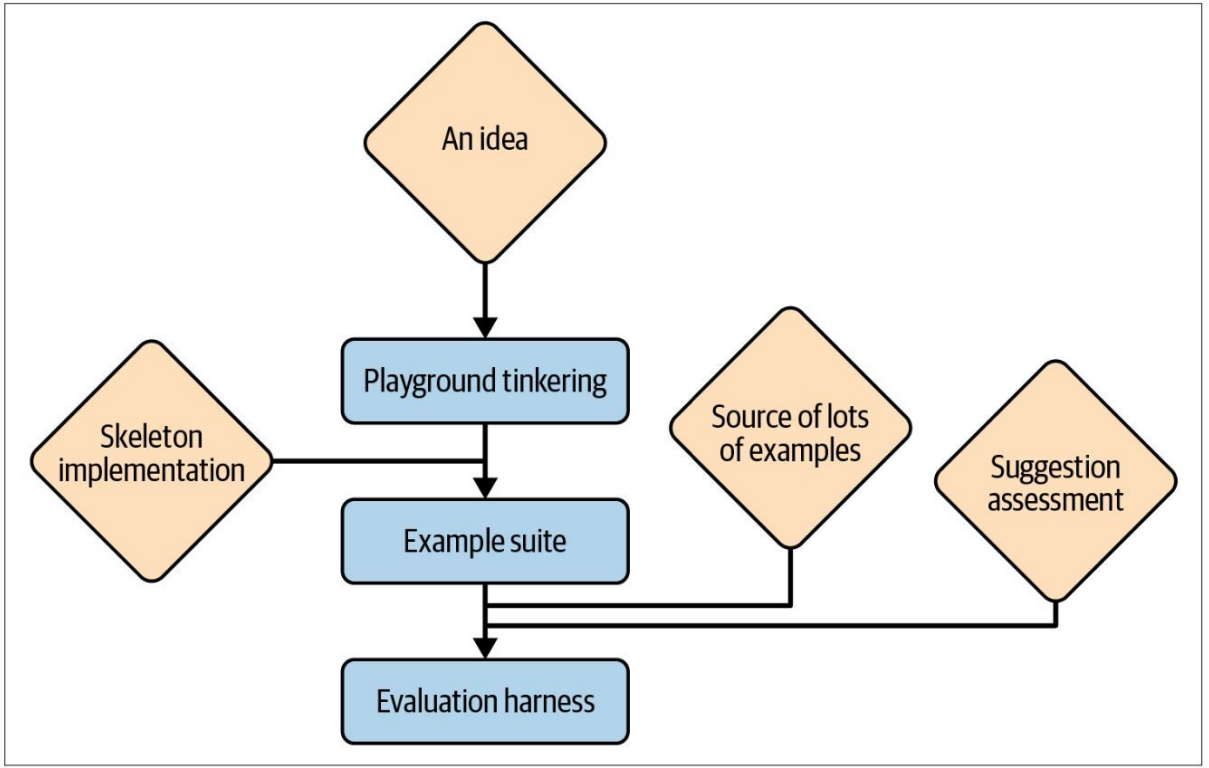


Figure 10-1. The tech tree of offline evaluations

“[W]ith general-purpose models, evaluation is not only about assessing a model’s performance on known tasks but also **about discovering new tasks that the model can do**.”

- Chip Huyen p.115

Final point

A one-line description of it

Second point

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua

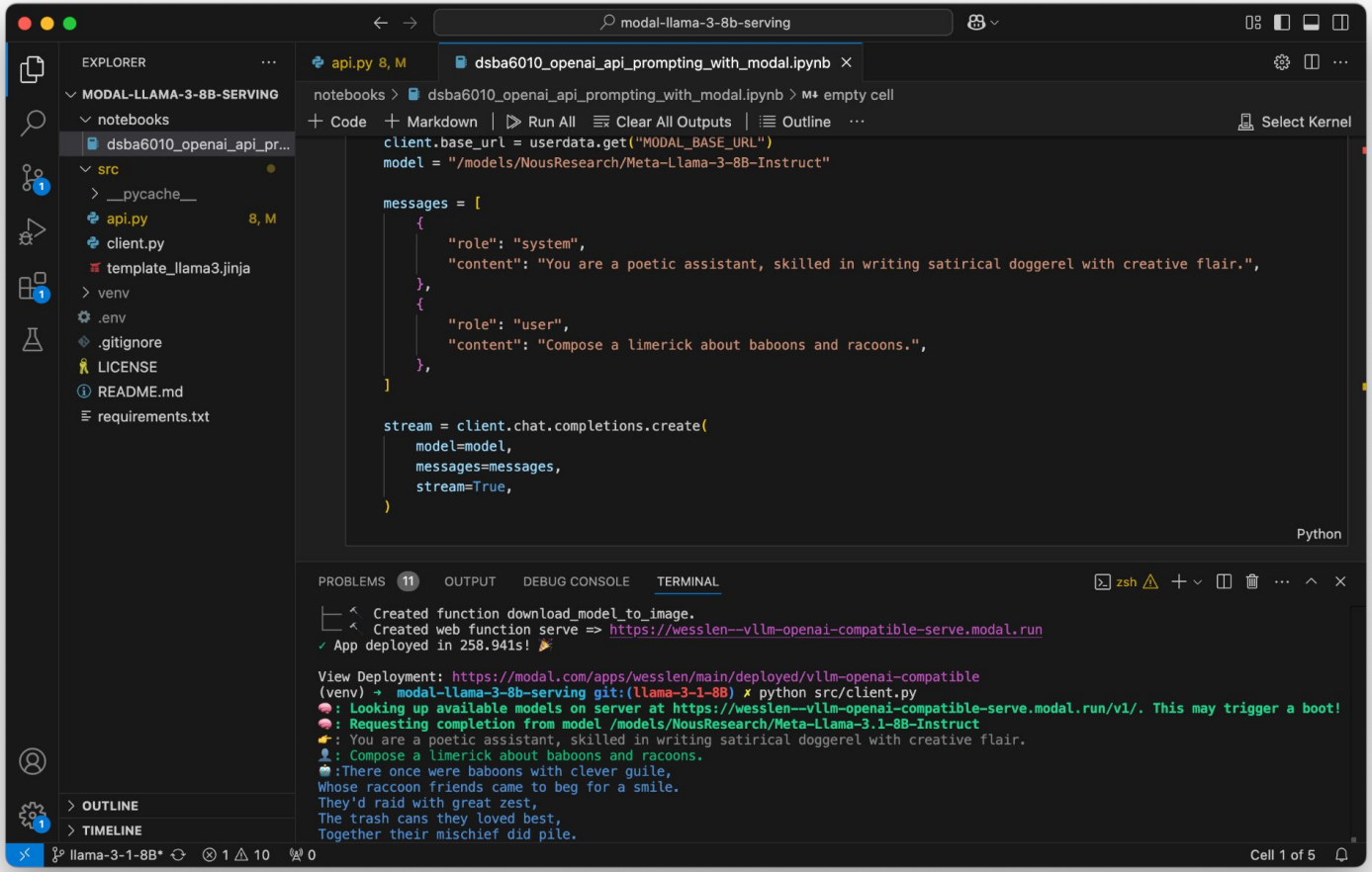
Incididunt ut labore et dolore

Consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua

This is the most important takeaway that everyone has to remember.

Thank you!

VSCode - LLM developers must go beyond notebooks.



Execution Flows

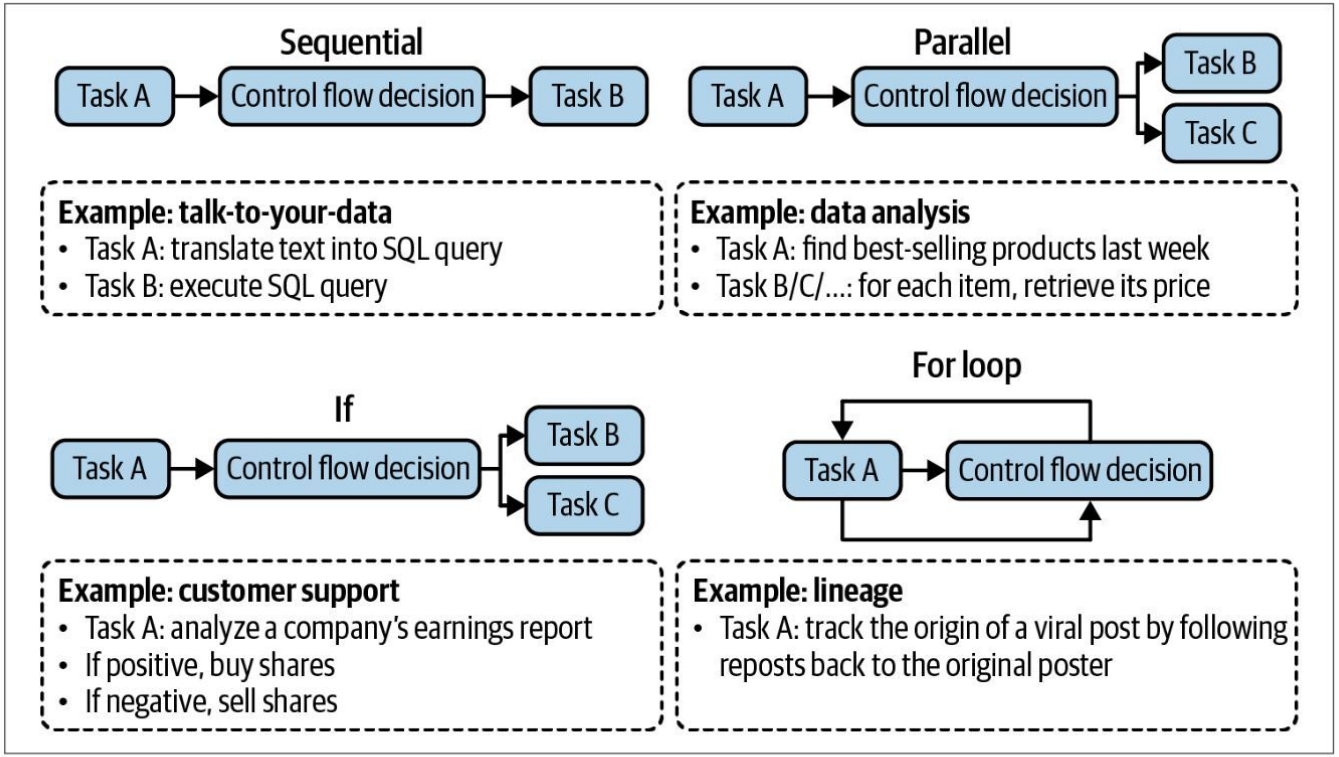


Figure 6-11. Examples of different orders in which a plan can be executed.