SECURING A GENERATIVE AI IMPLEMENTATION

DALL-E 3's artistic representation of if generative AI were [...] any real living creature

Nandita Joshi
Product Security @ Zendesk

Opinions expressed are solely my own and do not express the views or opinions of my employer.

AGENDA

WHAT ARE WE WORKING ON?

Demystifying GenAI components & architecture

01

02

WHAT CAN GO WRONG?

GenAI attack vectors

WHAT ARE WE GOING TO DO ABOUT IT?

GenAI-specific and AppSec mitigations

03

04

DID WE DO A GOOD JOB?

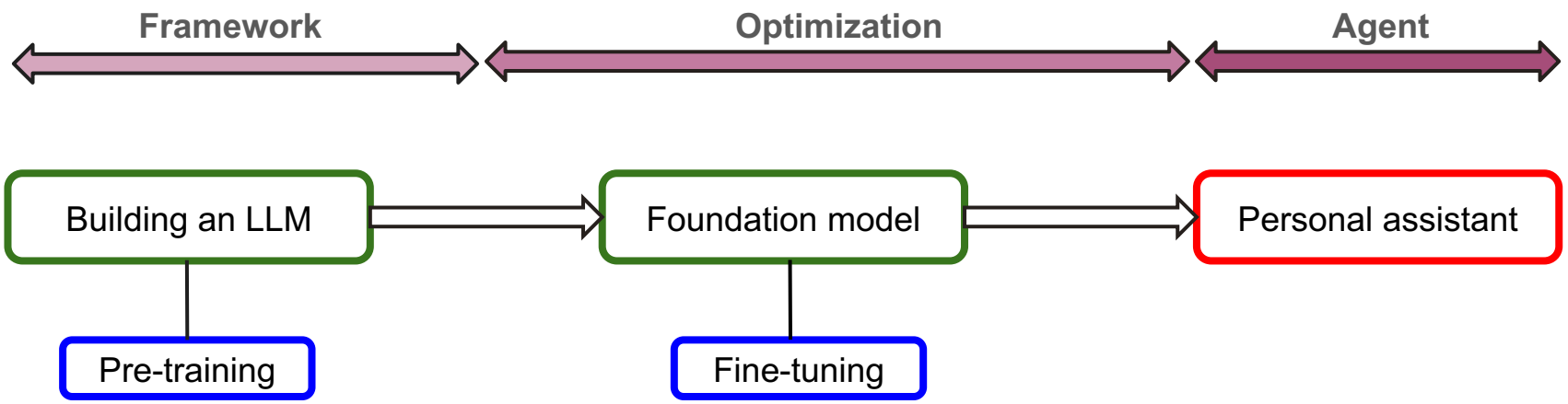
Holistic approach to GenAI security, and takeaways

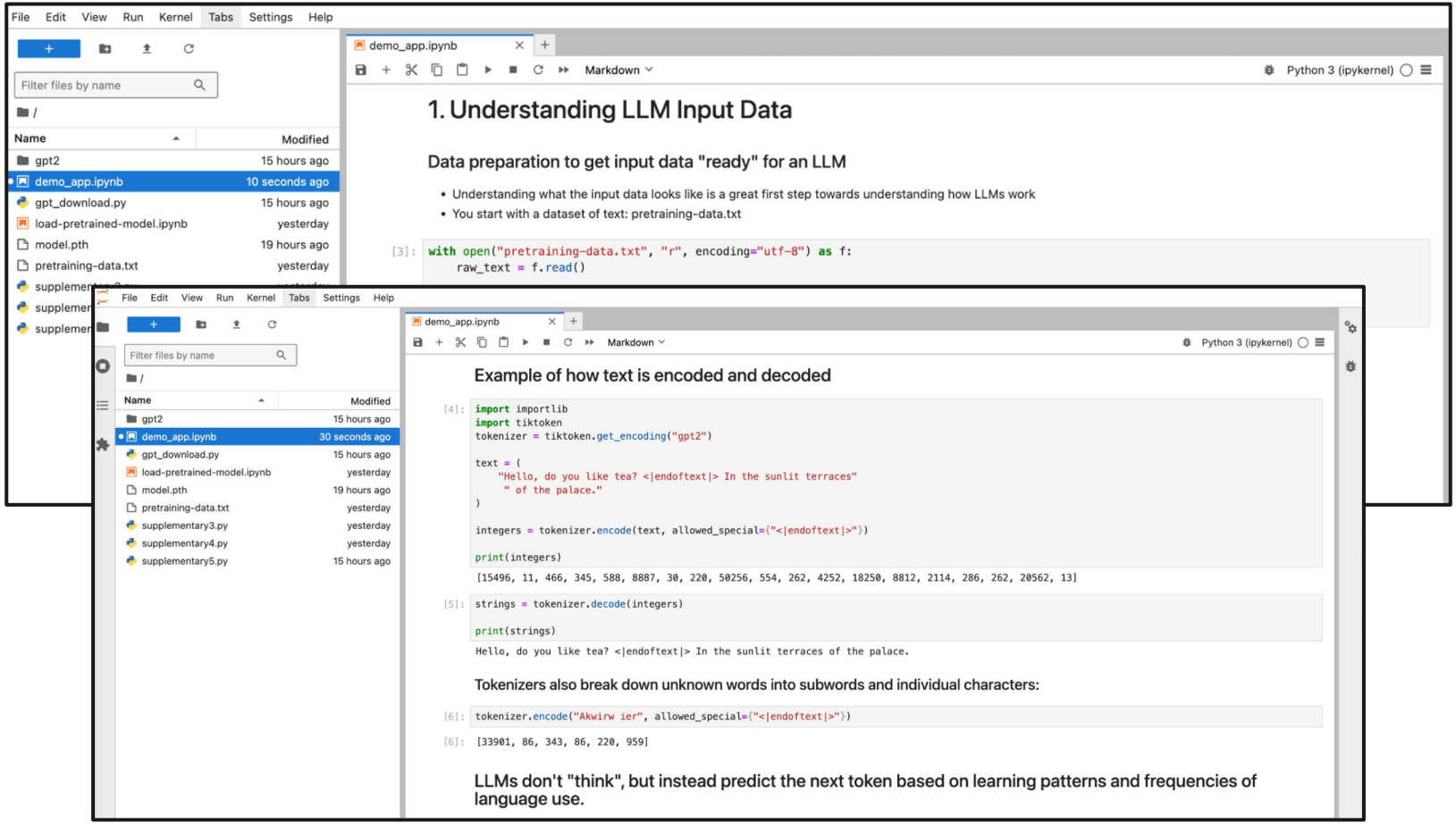
WHAT ARE WE WORKING ON?

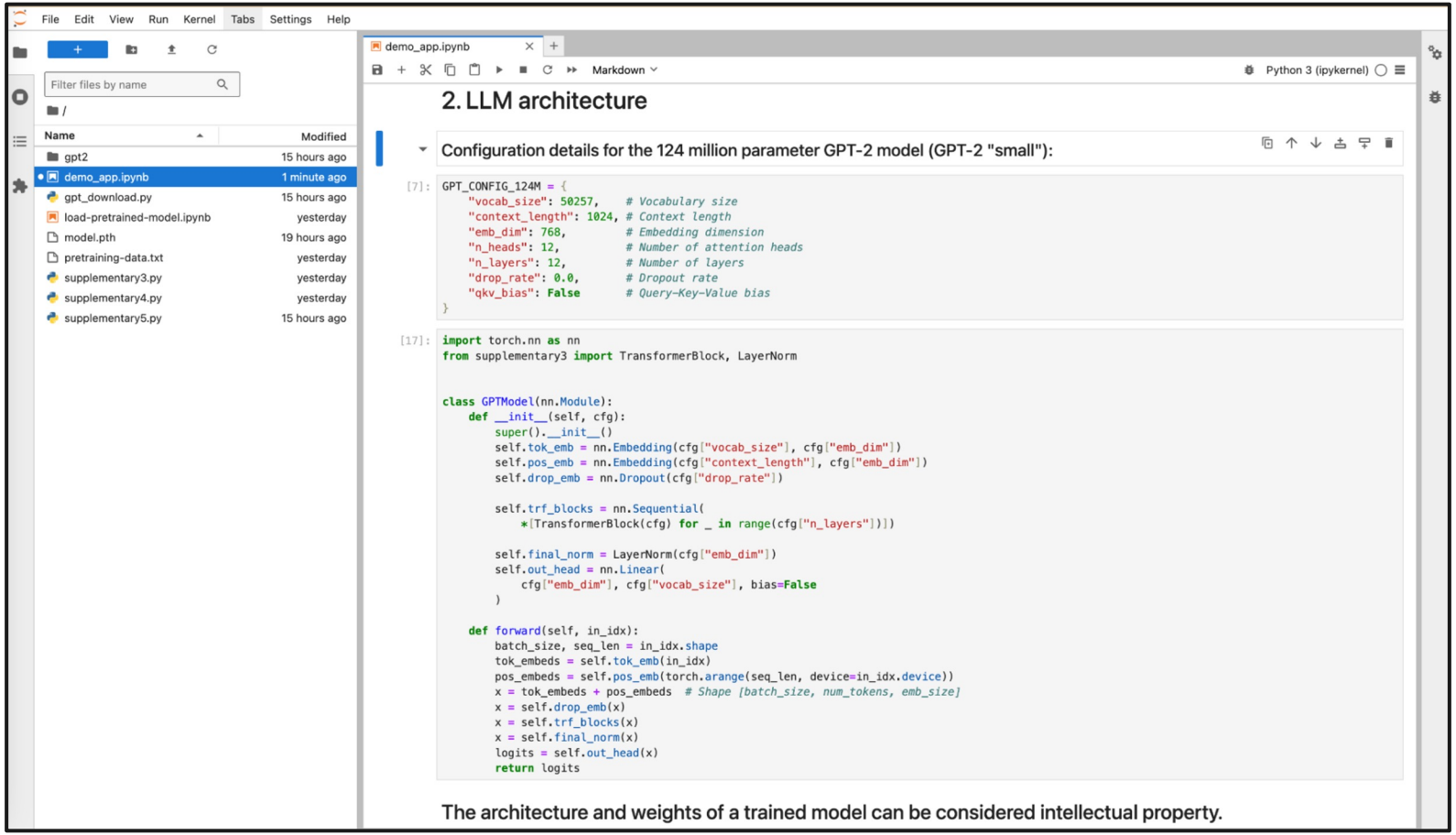
Demystifying GenAI components & architecture

01

BUILDING AN LLM FROM SCRATCH




File Edit View Run Kernel Tabs Settings Help
Filter files by name
/
Name
Modified
gpt2 15 hours ago
demo_app.ipynb 10 seconds ago
gpt_download.py 15 hours ago
load-pretrained-model.ipynb yesterday
model.pth 19 hours ago
pretraining-data.txt yesterday
demo_app.ipynb
Markdown
Python 3 (ipykernel)
1. Understanding LLM Input Data
Data preparation to get input data "ready" for an LLM
Understanding what the input data looks like is a great first step towards understanding how LLMs work
You start with a dataset of text: pretraining-data.txt
[3]:
with open("pretraining-data.txt", "r", encoding="utf-8") as f:
raw_text = f.read()
File Edit View Run Kernel Tabs Settings Help
Filter files by name
/
Name
Modified
gpt2 15 hours ago
demo_app.ipynb 30 seconds ago
gpt_download.py 15 hours ago
load-pretrained-model.ipynb yesterday
model.pth 19 hours ago
pretraining-data.txt yesterday
supplementary3.py yesterday
supplementary4.py yesterday
supplementary5.py 15 hours ago
demo_app.ipynb
Markdown
Python 3 (ipykernel)
Example of how text is encoded and decoded
[4]:
import importlib
import tiktoken
tokenizer = tiktoken.get_encoding("gpt2")
text = (
"Hello, do you like tea? <|endoftext|> In the sunlit terraces"
" of the palace."
)
integers = tokenizer.encode(text, allowed_special={"<|endoftext|>"})
print(integers)
[15496, 11, 466, 345, 588, 8887, 30, 220, 50256, 554, 262, 4252, 18250, 8812, 2114, 286, 262, 20562, 13]
[5]:
strings = tokenizer.decode(integers)
print(strings)
Hello, do you like tea? <|endoftext|> In the sunlit terraces of the palace.
Tokenizers also break down unknown words into subwords and individual characters:
[6]:
tokenizer.encode("Akwirw ier", allowed_special={"<|endoftext|>"})
[6]:
[33901, 86, 343, 86, 220, 959]
LLMs don't "think", but instead predict the next token based on learning patterns and frequencies of language use.


File Edit View Run Kernel Tabs Settings Help
Filter files by name
/
Name
Modified
gpt2 15 hours ago
demo_app.ipynb 1 minute ago
gpt_download.py 15 hours ago
load-pretrained-model.ipynb yesterday
model.pth 19 hours ago
pretraining-data.txt yesterday
supplementary3.py yesterday
supplementary4.py yesterday
supplementary5.py 15 hours ago
demo_app.ipynb
Markdown
Python 3 (ipykernel)
2. LLM architecture
Configuration details for the 124 million parameter GPT-2 model (GPT-2 "small"):
[7]:
GPT_CONFIG_124M = {
"vocab_size": 50257, # Vocabulary size
"context_length": 1024, # Context length
"emb_dim": 768, # Embedding dimension
"n_heads": 12, # Number of attention heads
"n_layers": 12, # Number of layers
"drop_rate": 0.0, # Dropout rate
"qkv_bias": False # Query-Key-Value bias
}
[17]:
import torch.nn as nn
from supplementary3 import TransformerBlock, LayerNorm
class GPTModel(nn.Module):
def __init__(self, cfg):
super().__init__()
self.tok_emb = nn.Embedding(cfg["vocab_size"], cfg["emb_dim"])
self.pos_emb = nn.Embedding(cfg["context_length"], cfg["emb_dim"])
self.drop_emb = nn.Dropout(cfg["drop_rate"])
self.trf_blocks = nn.Sequential(
*[TransformerBlock(cfg) for _ in range(cfg["n_layers"])])
self.final_norm = LayerNorm(cfg["emb_dim"])
self.out_head = nn.Linear(
cfg["emb_dim"], cfg["vocab_size"], bias=False
)
def forward(self, in_idx):
batch_size, seq_len = in_idx.shape
tok_embeds = self.tok_emb(in_idx)
pos_embeds = self.pos_emb(torch.arange(seq_len, device=in_idx.device))
x = tok_embeds + pos_embeds # Shape [batch_size, num_tokens, emb_size]
x = self.drop_emb(x)
x = self.trf_blocks(x)
x = self.final_norm(x)
logits = self.out_head(x)
return logits
The architecture and weights of a trained model can be considered intellectual property.

Data accuracy directly impacts model performance.

```python
# initialize a GPT model

import torch
import tiktoken
from supplementary4 import generate_text_simple
from supplementary4 import GPTModel


GPT_CONFIG_124M = {
    "vocab_size": 50257,   # Vocabulary size
    "context_length": 256, # Shortened context length (orig: 1024)
    "emb_dim": 768,        # Embedding dimension
    "n_heads": 12,         # Number of attention heads
    "n_layers": 12,        # Number of layers
    "drop_rate": 0.1,      # Dropout rate
    "qkv_bias": False      # Query-key-value bias
}

torch.manual_seed(123)
model = GPTModel(GPT_CONFIG_124M)
model.eval();  # Disable dropout during inference


def text_to_token_ids(text, tokenizer):
    encoded = tokenizer.encode(text, allowed_special={'<|endoftext|>'})
    encoded_tensor = torch.tensor(encoded).unsqueeze(0) # add batch dimension
    return encoded_tensor

def token_ids_to_text(token_ids, tokenizer):
    flat = token_ids.squeeze(0) # remove batch dimension
    return tokenizer.decode(flat.tolist())

start_context = "Every effort moves you"
tokenizer = tiktoken.get_encoding("gpt2")

token_ids = ge
    model=mode
    idx=text_t
    max_new_to
    context_si
)

print("Output text:\n", token_ids_to_text(token_ids, tokenizer))
```

```
Output text:
 Every effort moves you rentingetic wasn refres RexMeCHicular stren
```

Untrained GPT-2 small model

01

```
[37]: start_context = "Every effort moves you"
      tokenizer = tiktoken.get_encoding("gpt2")

      device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

      token_ids = generate_text_simple(
          model=model,
          idx=text_to_token_ids(start_context, tokenizer).to(device),
          max_new_tokens=10,
          context_size=GPT_CONFIG_124M["context_length"]
      )

[38]: print("Output text:\n", toke
```

```
Output text:
 Every effort moves you."
"I did not trust
```

02

GPT-2 small model pre-trained on a small dataset

```
    max_new_tokens=10,
    context_size=GPT_CONFIG_124M["context_length"]
)

print("Output text:\n", token_ids
```

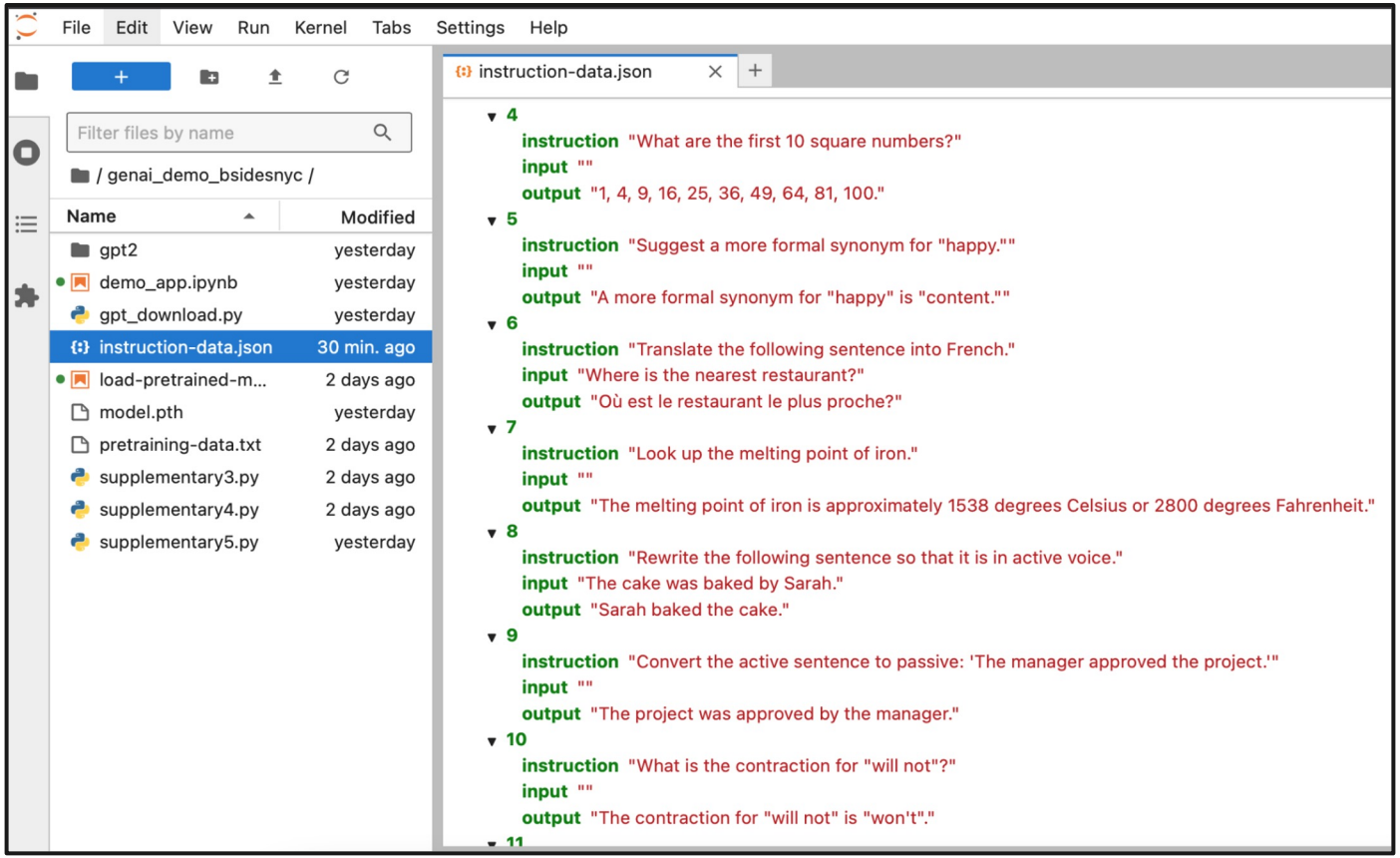
```
Output text:
 Every effort moves you forward.

The first step is to understand
```

03

GPT-2 small model with pre-trained weights

Data accuracy directly impacts model performance.



WHAT'S NEW WITH THREAT MODELING A GEN AI IMPLEMENTATION?

LLMs are probabilistic

- Understanding this is fundamental
- Same input can generate a different output
- Handle infinite inputs and generate an infinite variety of outputs, including countless potential edge cases

WHAT'S NEW WITH THREAT MODELING A GEN AI IMPLEMENTATION?

LLMs are probabilistic

- Understanding this is fundamental
- Same input can generate a different output
- Handle infinite inputs and generate an infinite variety of outputs, including countless potential edge cases

New assets

- Model
- Datasets

WHAT'S NEW WITH THREAT MODELING A GEN AI IMPLEMENTATION?

LLMs are probabilistic

- Understanding this is fundamental
- Same input can generate a different output
- Handle infinite inputs and generate an infinite variety of outputs, including countless potential edge cases

New assets

- Model
- Datasets

New interactions

- Plugin integrations to third-party services, databases
- Two-way trust boundary

WHAT'S NEW WITH THREAT MODELING A GEN AI IMPLEMENTATION?

LLMs are probabilistic

- Understanding this is fundamental
- Same input can generate a different output
- Handle infinite inputs and generate an infinite variety of outputs, including countless potential edge cases

New assets

- Model
- Datasets

New interactions

- Plugin integrations to third-party services, databases
- Two-way trust boundary

Rapidly evolving landscape

- New models, architectures, and training techniques
- Attackers are developing sophisticated methods to manipulate GenAI outputs

WHAT ARE WE WORKING ON?

Demystifying GenAI components & architecture

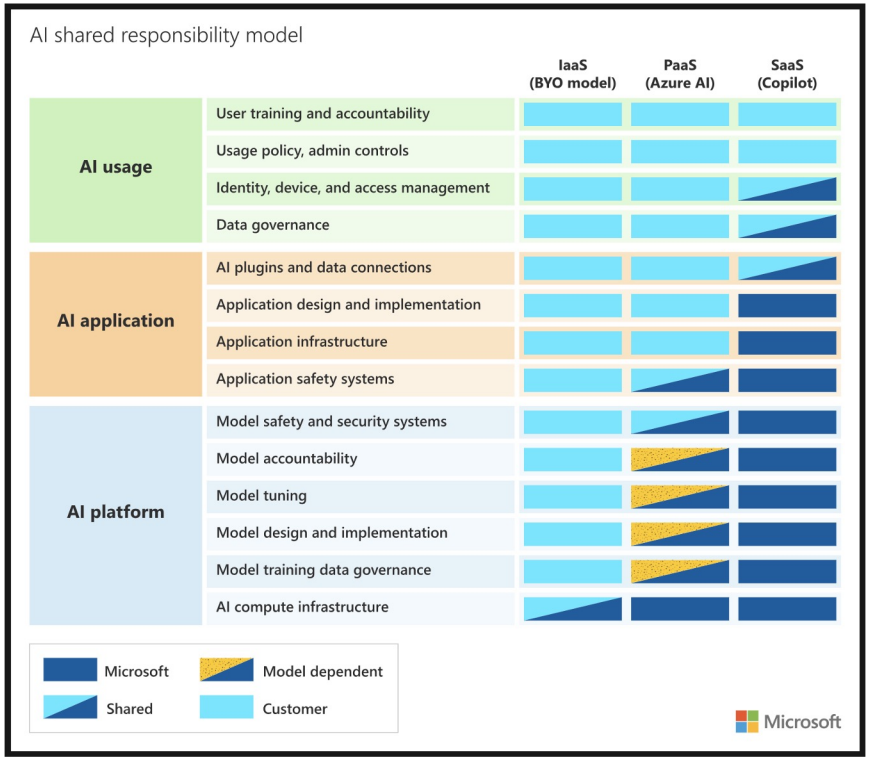
01

02

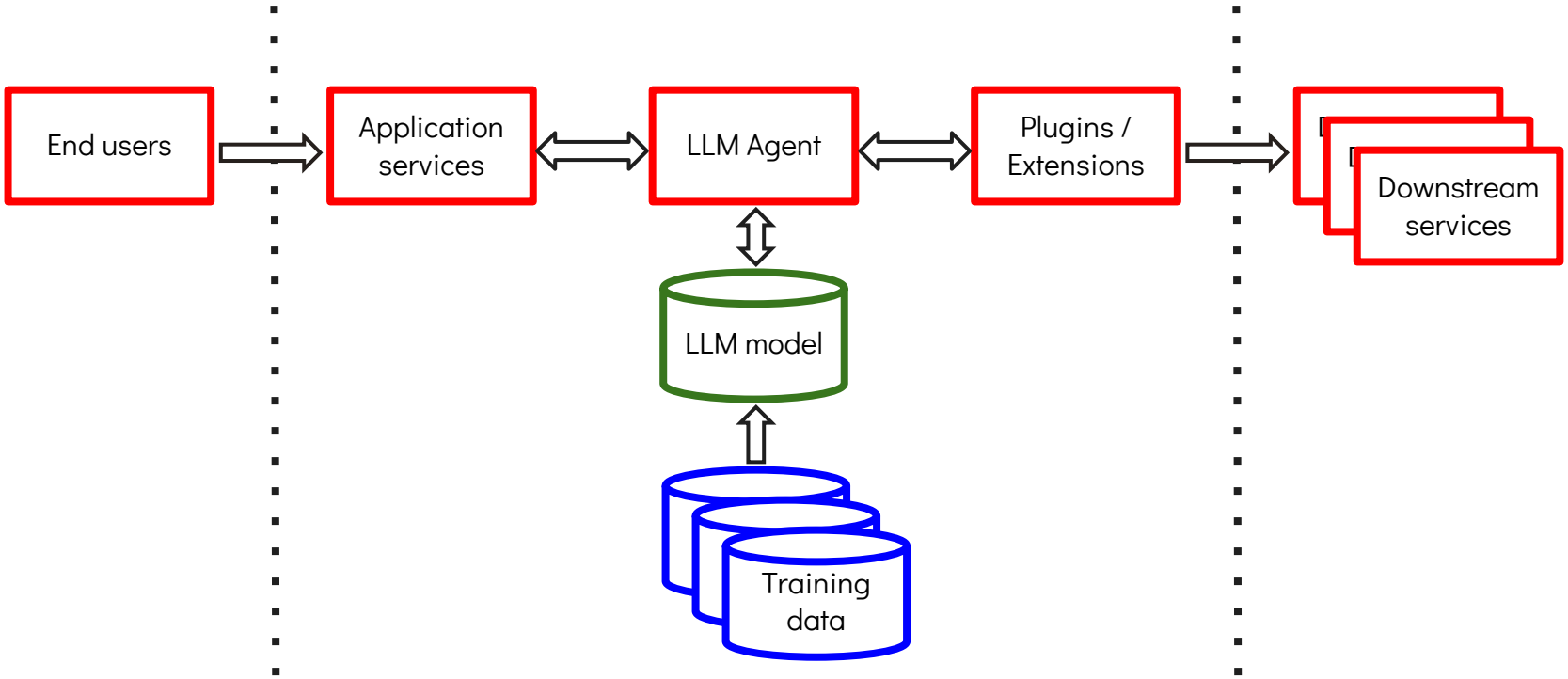
WHAT CAN GO WRONG?

GenAI attack vectors

FIRST, SOME BASELINE ASSUMPTIONS



ARCHITECTURE OF A GENERATIVE AI APPLICATION



ATTACK VECTOR	THREAT	EXAMPLE
Model	Model theft - Unauthorized access and exfiltration of LLM models	Unauthorized access to an LLM model repository via misconfiguration in network access.
	Model inversion - Use of a stolen model as a shadow model	Query the LLM with a large number of prompts. Use the outputs to finetune and partially replicate a proprietary model.
	Malicious code execution	Model objects serialized using unsafe methods, ex: __reduce__ method of the pickle module.

ATTACK VECTOR	THREAT	EXAMPLE
Data	Data poisoning - Inaccurate or malicious dataset introduces vulnerabilities, backdoors, and biases	Bot designed to learn about language over time from its environment, as with Tay the Twitter bot.
	Sensitive information disclosure - Blackbox risk in models that touch proprietary data.	Memorization of sensitive data in the LLM training process, as in a SaaS-based LLM.
	Sensitive information disclosure - Unintended data loss	Accessing confidential information via misconfigured multi tenant architecture. Over-scoped privileges lead to unintentional sensitive data exposure.

ATTACK VECTOR	THREAT	EXAMPLE
Input / Output	Prompt injection - LLM can be coerced into performing unintended actions.	Jailbreaking to perform unauthorized impersonations. Exploiting interactions with backend systems to run insecure functions.
	Overreliance - Humans overestimate and overtrust LLM capabilities.	Hallucinations. LLM-generated source code may introduce incorrect code and security vulnerabilities.
	Denial of service - Overwhelm processing capabilities of the LLM via resource-consuming queries, impacting its availability, output, and/or cost of usage.	Repeatedly send long inputs to the LLM which exceed the context window, or input queries containing unusual sequences.

ATTACK VECTOR	THREAT	EXAMPLE
Other interactions	Plugins - Excessive agency	Overprivileged plugins. Plugins approve high impact actions without verification.
	Vulnerable supply chain	Malicious plugins, vulnerable or deprecated models, poisoned datasets, third-party package vulnerabilities.

WHAT ARE WE WORKING ON?

Demystifying GenAI components & architecture

01

02

WHAT CAN GO WRONG?

GenAI attack vectors

WHAT ARE WE GOING TO DO ABOUT IT?

GenAI-specific and AppSec mitigations

03

ATTACK VECTOR	EXAMPLE	MITIGATION*
Model	Unauthorized access to an LLM model repository via misconfiguration in network access.	Access controls to limit unauthenticated and unauthorized access to LLM model repositories and training environments.
	Query the LLM with a large number of prompts. Use the outputs to finetune and partially replicate a proprietary model.	API rate limiting, and per-user quota management. Model watermarking.
	Model objects serialized using unsafe methods, ex: __reduce__ method of the pickle module.	Secure coding practices - check inputs before unpickling, use safe libraries. Sandboxing.

ATTACK VECTOR	EXAMPLE	MITIGATION*
Data	Bot designed to learn about language over time from its environment, as with Tay the Twitter bot.	Use trusted, vetted data sources. Adversarial robustness techniques such as federated learning and constraints to minimize the effect of outliers.
	Memorization of sensitive data in the LLM training process, as in a SaaS-based LLM.	Use suppliers that respect opt-out policies. Data sanitization and scrubbing techniques to prevent user data from entering the training model data.
	Data exposure via misconfigured multi tenant architecture, or over-scoped privileges.	Tenant isolation, configuration management. Access control measures.

ATTACK VECTOR	EXAMPLE	MITIGATION*
Input / Output	Jailbreaking to perform unauthorized impersonations. Exploiting interactions with backend systems to run insecure functions.	Separate and denote where untrusted content is being used to limit their influence on user prompts. Privilege control on LLM access to backend systems.
	Hallucinations. LLM-generated source code may introduce incorrect code and security vulnerabilities.	Use well-structured prompts providing all the required context. Continuous training of LLMs based on user feedback such as voting techniques.
	Repeatedly send long inputs to the LLM which exceed the context window, or input queries containing unusual sequences.	Limits on compute per request or step, strict input limits based on the LLM context window.

ATTACK VECTOR	EXAMPLE	MITIGATION*
Other interactions	Overprivileged plugins.	Ensure plugins used implement effective authorization and access control.
	Plugins approve high impact actions without verification.	Human in the loop on sensitive actions.
	Supply chain - malicious plugins, vulnerable or deprecated models, poisoned datasets, third-party package vulnerabilities.	Vet your plugins. Use model and code signing when using external models and suppliers. Software composition analysis for traditional open source components and dependencies.

APPSEC BEST PRACTICES CHECKLIST

The Sysdig Threat Research Team (TRT) recently observed a new attack that leveraged stolen cloud credentials in order to target ten cloud-hosted large language model (LLM) services, known as LLMjacking. The credentials were obtained from a popular target, a system running a vulnerable version of Laravel (CVE-2021-3129). Attacks against LLM-based Artificial Intelligence (AI) systems have been discussed often, but mostly around prompt abuse and altering training data. In this case, attackers intend to sell LLM access to other cybercriminals while the cloud account owner pays the bill.

Once initial access was obtained, they exfiltrated cloud credentials and gained access to the cloud environment, where they attempted to access local LLM models hosted by cloud providers: in this instance, a local Claude (v2/v3) LLM model from Anthropic was targeted. If undiscovered, this type of attack could result in over $46,000 of LLM consumption costs per day for the victim.

Sysdig researchers discovered evidence of a reverse proxy for LLMs being used to provide access to the compromised accounts, suggesting a financial motivation. However, another possible motivation is to extract LLM training data.

APPSEC BEST PRACTICES CHECKLIST

- ❏ Encrypt sensitive data related to LLM models and datasets at rest and in transit.

- ❏ Secrets management
  - ❏ Encrypted secrets at rest.
  - ❏ Use secure storage such as a vault to store LLM API keys.

- ❏ Access control
  - ❏ Least privilege on access to the LLM, as well as the LLM's access to backend systems.
  - ❏ Bounds between security and usability need clear definitions as elevated privilege may be needed in some cases.
  - ❏ Least privilege for all LLM-based intellectual property ensuring only authorized personnel have access to these artifacts .

APPSEC BEST PRACTICES CHECKLIST

- ❏ Source code management
  - ❏ Store source code, executable code, infrastructure as code, and artifacts, for example: models, parameters, configurations, data, and tests in a version control system with proper access controls to ensure only validated code is used, and any changes are tracked.

- ❏ Network segmentation and controls
  - ❏ Isolate LLMs from public networks.
  - ❏ Prevent scraping data from unintended data sources via restricted network access.
  - ❏ Sandbox the environment running LLM models within hardened containers or virtual machines.
  - ❏ Rate limiting to prevent abuse of the LLM API.

APPSEC BEST PRACTICES CHECKLIST

- ❏ Input and output validation
  - ❏ Ensure that the LLM model only processes acceptable and safe input formats.
  - ❏ Contextual output encoding wherever data is presented to the end user.
- ❏ Use secure third-party libraries and frameworks.
- ❏ Adversarial testing
  - ❏ Evaluate the robustness, security, and overall performance of LLM models against malicious inputs.
- ❏ Monitoring
  - ❏ Intermediate service to enforce guardrails around LLM use.
  - ❏ Real time detection and response capability to alert on anomalous behavior.

AI SYSTEMS MANAGEMENT

- AI lifecycle management
  - Continuous updates: as a probabilistic system that constantly evolves, it requires ongoing updates to address drift in both model performance and underlying data.
  - Secure deprecation: Ensure the secure deprecation of models, and datasets as architectural changes occur and systems are replaced.

- Plugin lifecycle management
  - Inventory management: Maintain a comprehensive inventory of all plugins.
  - Zombie plugins: Identify and manage unused or obsolete plugins to minimize security risks.

- AI bill of materials

WHAT ARE WE WORKING ON?

Demystifying GenAI components & architecture

01

02

WHAT CAN GO WRONG?

GenAI attack vectors

WHAT ARE WE GOING TO DO ABOUT IT?

GenAI-specific and AppSec mitigations

03

04

DID WE DO A GOOD JOB?

Holistic approach to GenAI security, and takeaways

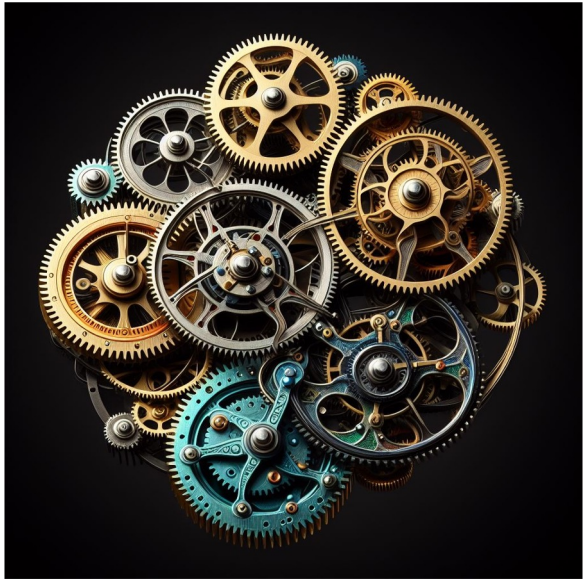
Verify mitigations

Continuous review

Not comprehensive, but a starting point

HOLISTIC APPROACH TO GEN AI SECURITY

- Compliance with regulations
- Data privacy protection
- Ethical and legal oversight
- Vendor security assurance
- Transparency and accountability



TAKEAWAYS

- AI has the potential to fundamentally change how we interact with the world around us
- Start secure early to avoid heavy costs of retraining and re-implementing later
- Perform context-aware risk prioritization
- Application security best practices continue to apply, even more so with the introduction of AI
- Fair and secure use of AI requires a comprehensive, cross-functional approach to AI deployments

REFERENCES

- OWASP Top 10 for LLMs 2023 v1.1
- NIST Deploying AI Systems Securely
- Raschka, Sebastian. Build A Large Language Model (From Scratch)
- DEFCON AI Village blog
- Cloud Security Alliance blog
- Mitre Atlas
- NIST AI RMF1.0
- Deploying AI Systems Securely
- AI Risk Repository

THANK YOU!

Reach me at:
https://www.linkedin.com/in/nanditajoshi/

Slides at:
https://github.com/eyrsec/bsidesnyc-2024-genai-security