AI Readiness:
Knowledge Graphs

FacultyHack26

Deconstructing Large Language Models (The How's and Why's of LLMs)

The Micro-GraphRAG Challenge

A Hands-On Lab Recipe for Structuring Science Text using LLM Co-Pilots

Lab Goals: Building Smartly

AI Co-Piloting

Learn to use LLMs to generate boilerplate code, debug errors, and explain logic.

Graph Structures

Convert unstructured science text into nodes, edges, and triples.

Python Mastery

Utilize networkx and matplotlib for high-impact data visualization.

PROMPT CHALLENGE:

If you could automate any part of your coding homework, which part would you trust an AI with the most? Why?

The Co-Pilot Mindset

Don't Just Copy-Paste

The goal is **Collaborative Engineering**. Use the LLM to:

- Explain obscure Python libraries.
- Transform text into JSON structures.
- Help you "rubber duck" debug your logic.

The 4-Step Lab Recipe

Step 1: Choose Text

Select a 2–3 sentence scientific fact summary containing clear entity connections.

Step 2: Prompt LLM

Feed the text to ChatGPT or Gemini to extract clean, formatted relational JSON triples.

Step 3: Setup Colab

Create a new notebook, import Python libraries, and copy-paste your graph JSON.

Step 4: Visualize

Co-pilot with the LLM to write NetworkX code that plots your interactive visual graph map.

> 💡 **CODING MINDSET**
> Our goal today is "Collaborative Engineering": using the LLM to write, debug, and explain code, not just complete it blind.

Phase 1: Selecting the Source

The Raw Material

Choose a complex scientific summary. For example:

> *"Photosynthesis occurs in chloroplasts. It converts sunlight into glucose using carbon dioxide and water as inputs."*

The Objective

Identify **Entities** (Nodes) and **Relationships** (Edges).

- Entity A: Photosynthesis
- Relationship: Occurs in
- Entity B: Chloroplasts

REFLECTION:

How does human reading differ from how an AI might "read" a paragraph to find these connections?

Phase 1: Selecting Your Source

1. Select Complex Facts

Pick a dense paragraph with clear scientific relationships. Avoid simple statements; choose sequences where cause-and-effect or location links exist.

Example Fact Text:
"Photosynthesis occurs inside chloroplasts. Chloroplasts contain chlorophyll, which captures sunlight to split water molecules."

2. Identify Triples Mentally

Before prompting, think about the relationships yourself. A triple consists of a **Source** entity, a **Relation** action, and a **Target** destination.

Expected Triples:
- (Photosynthesis ➔ occurs inside ➔ Chloroplasts)
- (Chloroplasts ➔ contain ➔ Chlorophyll)

MENTAL CHECK

How does breaking a paragraph into discrete "Triples" help computers read facts more accurately than reading human paragraphs?

The Anatomy of Knowledge

3

The Semantic Triple

Every relationship in our graph is stored as a "Triple". This is the mathematical foundation of Knowledge Graphs.

```
( Subject , Predicate , Object )
```

Example: (Earth, orbits, Sun)

Phase 2: Prompting for JSON

- **The System Message:** "You are a data extraction pipeline. Extract entities and relations as a JSON array."
- **The Constraint:** "Format only as JSON. Use keys: 'source', 'relation', 'target'."
- **The Input:** Provide the AI with your scientific paragraph.

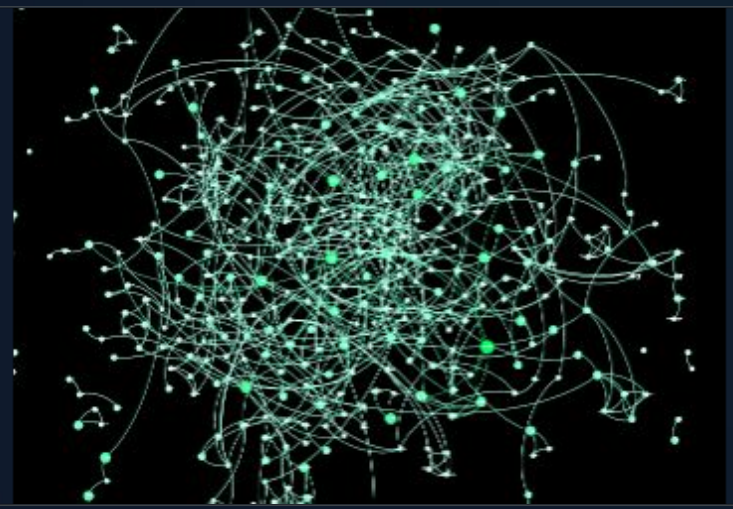
```
[{"source": "Sunlight", "relation": "powers", "target": "Photosynthesis"}]
```

Phase 2: Extraction Prompt

The Structured LLM Prompt

To get perfectly structured data from an LLM, write strict boundary guidelines. Use the exact prompt block below with your source text:

```
Extract all entities and relationships from this text: "
[INSERT YOUR TEXT HERE]". Format the output strictly as a valid
JSON array of objects. Use these keys: "source", "relation",
"target". Output ONLY raw JSON. Do not include introductory text,
markdown formatting, or conversational replies.
```



The Expected JSON Format

- **Strict JSON Arrays:** The LLM must output an outer array [...] containing your relational facts.
- **Lowercase Keys:** Every element inside must look exactly like: {"source": "X", "relation": "Y", "target": "Z"}.
- **A Clean Code Copy:** Copy this exact formatted output block. You will paste this directly into your Google Colab cell.

```
[ {"source": "Photosynthesis", "relation": "occurs inside", "target": "Chloroplasts"}, {"source": "Chloroplasts", "relation": "contain", "target": "Chlorophyll"}, {"source": "Chlorophyll", "relation": "captures", "target": "Sunlight"} ]
```

TROUBLESHOOTING CHECK

If your LLM writes conversational text like "Here is your JSON...", make sure to delete that part so you only copy the array structure!

Phase 3: Launching Jupyter on Frontera

Setting Up Your Notebook

Google Colab runs on cloud servers, meaning you do not need to install Python on your computer.

The Workspace Recipe:

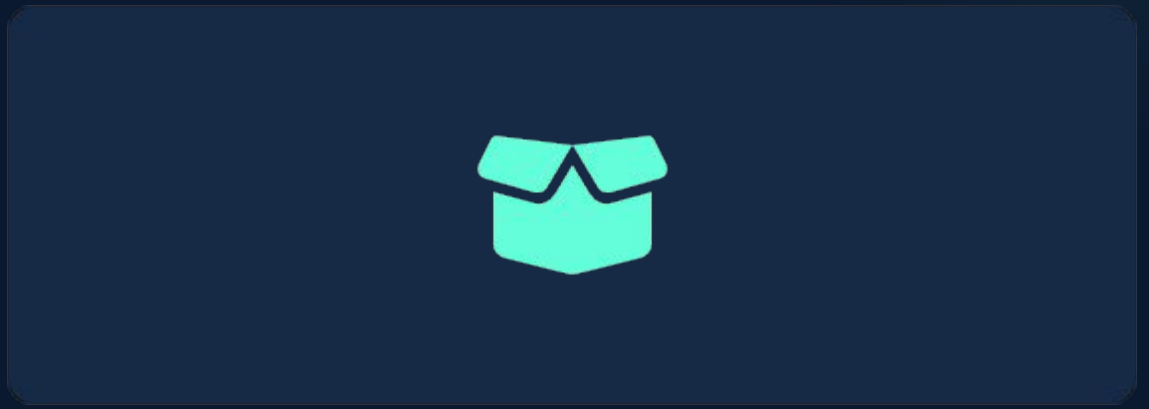
1. Navigate to colab.research.google.com and click "New Notebook".
2. Inside your first code cell, run this installation command:

```
!pip install networkx matplotlib
```

3. Click the Play button to execute the installation setup.

Phase 3: Setting Up Colab

Step A: Run !pip install networkx to ensure the graph library is ready.



Step B: import networkx as nx to bring the tools into your code cells.

Phase 4: Co-Generating Code

The Prompt to Assistant:

"Write a Python script using NetworkX to create a graph from this list of dictionaries: [YOUR_JSON]. Explain the add_edge function."

Key AI Insight

The AI will likely use a loop:

```
for item in data:
  G.add_edge(item['source'],
  item['target'],
  label=item['relation'])
```

This loop maps the abstract JSON to the physical memory of the Graph object.

Phase 4: Requesting Code

The Python Generation Prompt

Now, prompt the LLM to write the Python script. Provide the JSON data you extracted in Step 2 to ensure the script maps your specific entities.

```
I have this JSON graph data: [PASTE YOUR JSON HERE]. Write a beginner-friendly Python script using networkx and matplotlib. Load this JSON into a networkx directed graph and draw the visualization. Add comments to explain the graph drawing functions.
```

What to Ask Your Co-Pilot

To deepen your understanding, query the LLM while it generates code:

- *"What does G = nx.DiGraph() mean? What is a directed graph?"*
- *"How does the add_edge function connect my JSON to the graph?"*
- *"Why do we need plt.show() at the end of the script?"*

Expected Code: Loading Data

- **Libraries:** Standard library declarations import networkx as nx and import matplotlib.pyplot as plt.
- **Instantiation:** Setting up a blank Directed Graph object G = nx.DiGraph().
- **The Mapping Loop:** Running a for loop that extracts keys and populates nodes.

```
import networkx as nx import matplotlib.pyplot as plt # Your extracted triples data = [ {"source": "Photosynthesis", "relation": "occurs inside", "target": "Chloroplasts"}, {"source": "Chloroplasts", "relation": "contain", "target": "Chlorophyll"} ] G = nx.DiGraph() # Instantiate graph # Loop through list and inject nodes/edges for triple in data: G.add_edge(triple["source"], triple["target"], relation=triple["relation"])
```

Expected Code: Drawing

Plotting Code & Positions

To display your graph visually, Python needs to calculate pixel coordinates for each node on a flat surface.

We use a spring layout mechanism which uses virtual physics to pull connected nodes close and push disconnected nodes away.

Drawing Coordinates Cell

```
# 1. Calculate physics positions pos = nx.spring_layout(G) # 2. Draw
nodes and edges nx.draw(G, pos, with_labels=True,
node_color='skyblue', node_size=2000, font_size=10,
font_weight='bold') # 3. Render in Google Colab plt.show()
```

VISUAL CHALLENGE

Paste the drawing cell into Google Colab. Does the graph help you spot which scientific entity is the most critical bottleneck?

Handling Errors with LLMs

The Problem	The Error Message	The AI Prompt for Help
Missing Library	ModuleNotFoundError	"How do I install this in Colab?"
Bad Data Format	KeyError: 'source'	"The JSON keys don't match my loop."
Layout Issue	Nodes are overlapping	"Explain spring_layout k parameter."

Phase 5: Fixing Common Errors

Colab Error Encountered	Underlying Cause	Exact Prompt to Ask Your LLM
ModuleNotFoundError: No module named 'networkx'	The Python runtime in Colab does not have the library active yet.	"Explain how to install networkx inside Google Colab using a bash terminal command."
KeyError: 'source'	Your Python loop is looking for keys that your LLM forgot to write.	"My dictionary has keys 'origin' and 'dest' instead of 'source' and 'target'. How do I update my loop?"
JSONDecodeError	The JSON data contains syntax errors like trailing commas or missing quotation marks.	"Fix this JSON string and explain what formatting rule was broken: [PASTE FAILED JSON]"

The "RAG" in GraphRAG

3x

Higher Fact Accuracy

How Graph Retrieval Works

Standard search algorithms look for loose matching words. In contrast, **GraphRAG** searches across established nodes and edges.

If a user asks: *"Where does photosynthesis take place?"*, the system queries your constructed graph, instantly traces the exact edge (Photosynthesis ➔ occurs inside ➔ Chloroplasts), and delivers a fully accurate answer with zero hallucinations.

Let's do this

Start with Phase 1: Select your scientific fact text and launch your co-pilot!

Part 1

The Mathematical Foundations of Artificial Language

THE GREAT AI ILLUSION

Linguistic Pretence

When you converse with an LLM, your brain instinctively projects consciousness and comprehension onto the system. The chatbot appears to think and feel like an assistant.

Deterministic Statistics

Beneath the screen, there is no mind. Modern AI models are massive matrix multiplication equations designed to predict the single most mathematically likely next syllable.

THE VOCABULARY PROBLEM

The Infinite List

Computers cannot process string characters directly; they must convert text to numbers. However, dedicating a unique ID to every single word crashes system memory instantly.

The Inflection Trap

If "Run", "Running", "Ran", and "Runner" are treated as completely distinct entities, the engine fails to recognize semantic overlap, making generalization impossible.

THE ANATOMY OF A TOKEN

Sub-word Shredding

Instead of whole words, AI systems fragment language into sub-word pieces called tokens, typically tracking prefixes, roots, and suffixes.

Dynamic Matching

By compiling basic syllables, the computer can organically decipher slang, typos, and newly coined jargon without catalog expansion.

Memory Compression

Sub-word slicing allows a vocabulary list of only 50,000 to 100,000 structural pieces to represent infinite human concepts.

THE ANATOMY OF A TOKEN

SUB-WORD SHREDDING

[unforgettably] → Tokenization → **un** | **forget** | **ta** | **bly**

[Prefix: **un**] [Root: **forget**] [Suffix: **ta**] [Suffix: **bly**]

fragments → fragments → shredding

DYNAMIC MATCHING

un	pre	dict	
able	ing	ed	ly

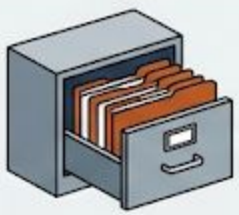
Compiling basic syllables deciphers slang, typos, and newly coinly coined jargon without catalog expansion.

unpredicted
predicting
unpredictablying

deciphering organic matching

MEMORY COMPRESSION

efficiency



50k - 100k structural pieces

VOCABULARY LIST

Sub-word slicing represents

Allows a limited vocabulary to represent infinite ideas, dramatically compressing memory requirements.

INFINITE HUMAN CONCEPTS

concepts

THE TOKENIZER ENGINE

Example Implementation

> Observe how this Python class matches text segments against a pre-sorted dictionary of known sub-words:

```
vocab = ["Av", "eng", "ers", "Iron", " Man", "The ", "will ", "return"]

def naive_tokenize(text):
    tokens = []
    temp = text
    for piece in sorted(vocab, key=len, reverse=True):
        if piece in temp:
            tokens.append(piece)
            temp = temp.replace(piece, " ")
    return tokens

print(naive_tokenize("The Avengers will return"))
```

Structural Logic

Sorting by length ensures the longest matched fragment takes precedence over shorter sub-segments.

Replaced segments are blanked out with whitespace to prevent recursive or overlapping duplicate matches on subsequent loops.

Side Quest: The Statistical Shredder

Why AI Ignores Your English Teacher

SGX3 Coding Institute

Human Logic vs. AI Math

The Counter-Intuitive Reality of Tokens

Linguistics vs. Machine Reality

Human Intuition

If a linguistics teacher were grading "Avengers," the word should be split into **"Avenge"** (the root) and **"ers"** (the suffix).

Machine Secret

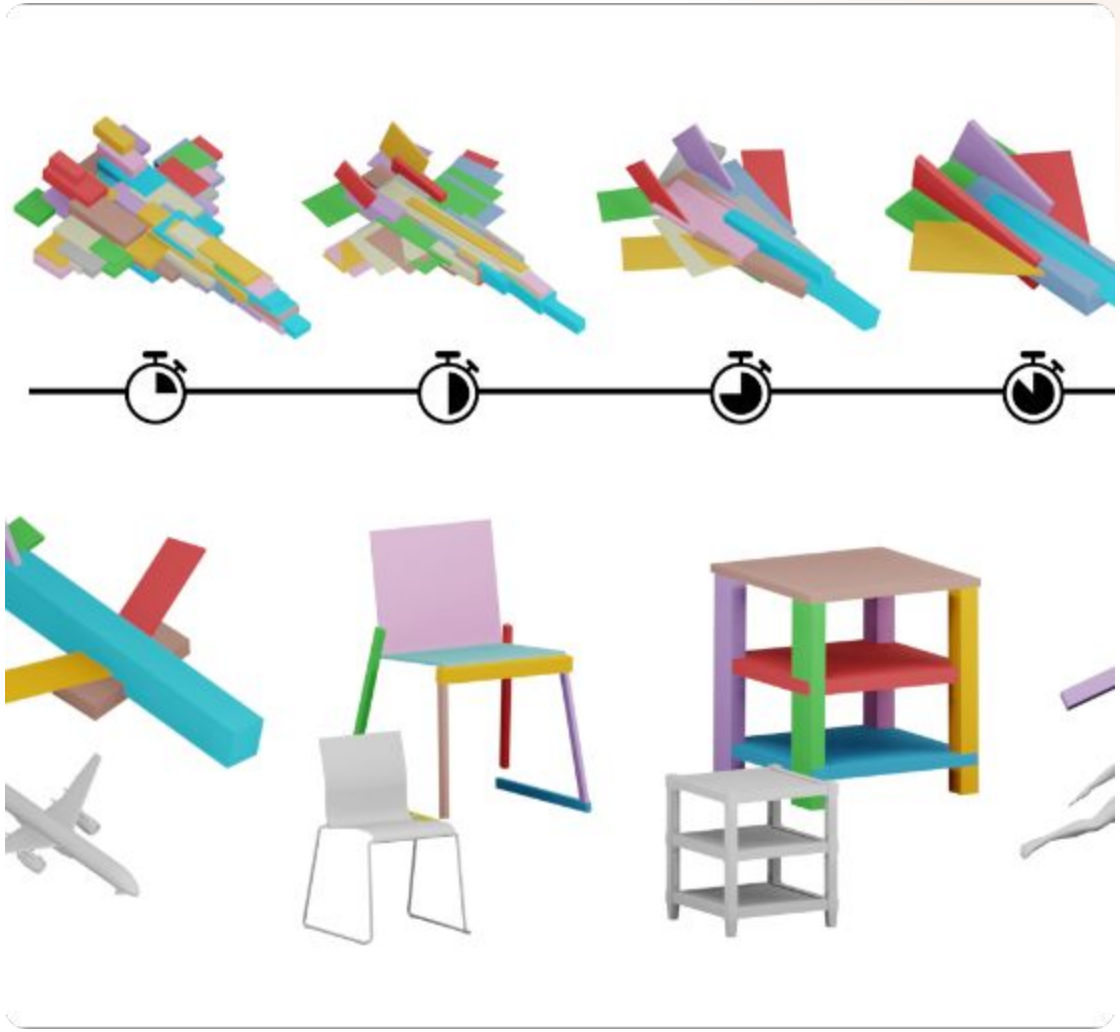
The AI tokenizer has **zero** idea what a prefix, suffix, or verb is. It completely ignores English grammar rules.

Statistical Chunking Secrets

Math Over Grammar

Engineers feed AI billions of pages of internet text. The algorithm's only job is to find the most common **combinations of letters**.

These mathematical "puzzle pieces" (tokens) are chosen based on frequency, not dictionary definitions.



The Power of "eng"

Frequency

To a computer, "eng" is incredibly common. It appears in: engineering, length, English, strength, and challenge.

Utility

Because "eng" appears in so many contexts, it is a high-value "puzzle piece" for the AI dictionary.

Versatility

The machine prefers "eng" over "Avenge" because "eng" can be reused across thousands of unrelated words.

The "Commonality" Hierarchy

"Av"

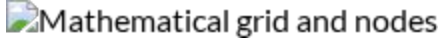
Found in: Avenue, Avatar, Aviation, Available. It is a mathematical superstar.

"ers"

Found in: runners, players, letters. It is statistically ubiquitous.

"Avenge"

Surprisingly rare in everyday text compared to its smaller fragments. It loses the math game.



Efficiency Over Rarity

Why "Avenge" Fails

Why waste a precious dictionary slot on the rare word "Avenge"?

The AI realizes it can just build it using "Av" + "eng" + "e"—pieces it already uses for thousands of other words. This is the **Path of Least Resistance**.

The 100,000 Slot Limit

100k

CORE TOKENS

A Crowded Dictionary

Modern AIs try to keep their base dictionary limited to around 100,000 core pieces.

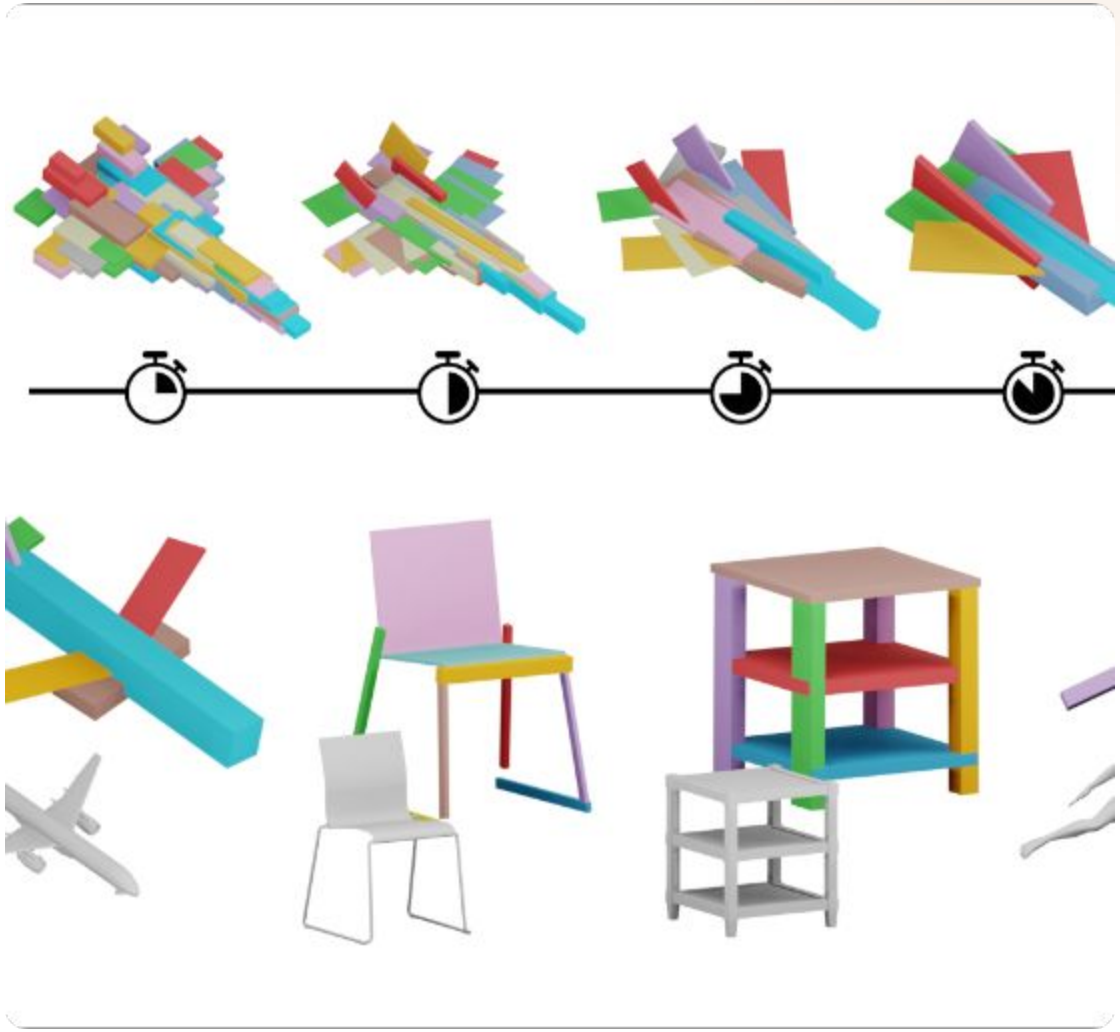
By using highly versatile fragments, they can represent **millions** of words without increasing memory requirements.

The Dynamic Stitcher

Handling the Unknown

If the AI knows "Av," "eng," and "ers," it can dynamically stitch them together to understand "Avengers"—even if it never saw the whole word before.

This allows the AI to decipher typos and handle entirely new slang words on the fly without breaking.



The Inevitable Conflict

"The AI acts like a data-compression machine. It doesn't see language; it sees high-frequency patterns. It will happily chop 'Avengers' into fragments that make an English teacher cringe."

— SGX3 Coding Institute

The Takeaway

Language as a ZIP File

"Prefixes and suffixes" are just human metaphors. Under the hood, AI is a compression engine finding the most mathematically efficient way to store the universe.

MICRO-CHALLENGE 1

The Unknown Entity

Execute the tokenizer script in your Jupyter Notebook, but swap the input string to: "Iron Man and Spider-Man will return"

Note how the tokenizer fails or completely drops "Spider-Man" because it does not exist inside our restricted database dictionary.

LLM-Assisted Task

> Work with your LLM partner on the following objective:

Goal: Determine how to expand the vocab list so that the script successfully breaks down the new term "Spider-Man" into three separate fragments without breaking.

REVEAL: CHALLENGE 1 SOLUTION

The Vocabulary Patch

> To integrate the new entity, we expand the vocabulary array:

```
# Updated structural vocabulary
ai_vocabulary = [
    "Av", "eng", "ers", "Iron", " Man", "The ", "will ", "return",
    "Spi", "der", "-Man", " and "
]
```

Industry Parallel

This mimics how modern commercial AI developers "patch" models when new entities emerge. If the tokenizer doesn't possess the fragments, the AI remains completely blind to the topic.

THE GEOMETRY OF MEANING

Beyond Integer Indexes

Giving a token a simple ID number like `412` tells the computer nothing about context. The system requires a system to represent semantic associations.

High-Dimensional Coordinate Maps

To capture meaning, AI models transform tokens into coordinate paths inside a giant imaginary grid called a Vector Space (or Embedding Space).

MAPPING CONCEPTUAL NEIGHBORS

Spatial Definition

In an AI's memory, words are defined entirely by where they sit. Synonyms are mapped to nearly identical coordinate addresses.

Contextual Pull

If human authors constantly use two tokens together on the internet, the training algorithm pulls their spatial coordinates closer.

Conceptual Clustering

Factions and subjects naturally form spatial neighborhoods. Cosmic entities cluster on one side; technical items group on the other.

Sidequest: Contextual Pull

How AI Learns the Mathematical Gravity of Meaning

SGX3 Coding Institute | Deep Dive Module. Exploring Firth's Linguistic Law, Vector Attraction Vectors, and the optimization gradients that structure neural memory.

THE PARTY ANALOGY

Random Seat Assignments

Think of vector space like a giant party where everyone arrives as complete strangers. At the start of the night, the host assigns seating entirely at random.

As a result, mathematicians are sitting next to novelists, and physicists are sitting next to historical chefs. The room is in structural chaos.

Chairs Move Together

If two people find out they share a deep connection, they naturally pull their chairs closer together to talk. Those who share zero common ground push their chairs away.

By the end of the evening, the room is no longer random. Dynamic conversation groups have physically structured the entire layout of the space.

FIRTH'S LINGUISTIC LAW

The Company You Keep

In 1957, linguist J.R. Firth wrote that a word's meaning is defined entirely by the neighbor tokens surrounding it. If we drink a glass of "water" or "juice", these context neighbors immediately inform us that both words describe liquid refreshments.

To Matrix Math

Instead of manually constructing rigid language definitions, we write code to map this behavior. When human authors constantly use two words together in real articles, the training algorithm registers their shared contextual proximity.

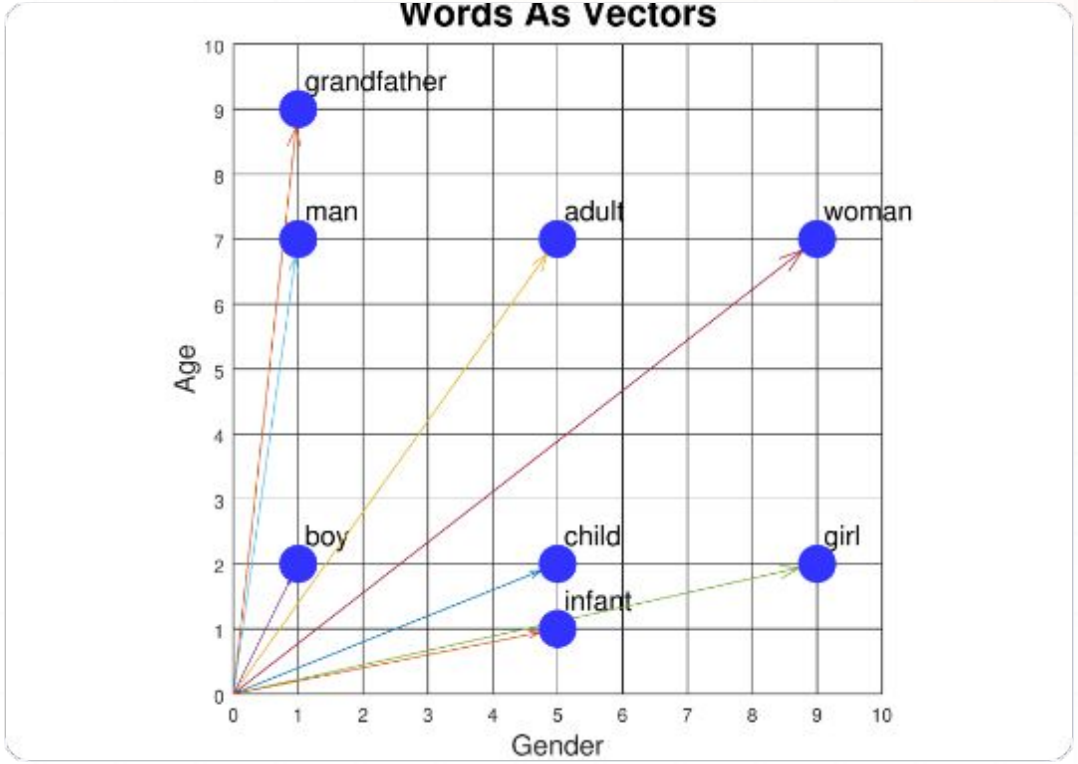
THE INITIAL CHAOS

Scattered Concept Space

When a language model is first initialized, every single vocabulary word is placed completely at random in our high-dimensional coordinate grid.

At start-state, unrelated words like "banana" and "quadratic" might sit directly next to "Iron Man". There is absolutely no linguistic order inside the computer's memory yet.

The training algorithm must now use the internet's text context to pull related items together, acting as a structural vector gravity beam.



THE GRAVITY FORMULA

The Proximity Dot Product

The Mathematical Pull

To measure alignment between a target word vector and its neighbor vector, the algorithm computes their spatial dot product.

If they share a sentence, but point in different directions, a massive error is registered. The optimization code then calculates a gradient update.

This gradient acts as a gravitational force, physically nudging the coordinate numbers closer together so their future alignment is stronger.

REPULSIVE FORCE

Attraction Breeds Collapse

If we only tell the computer to pull connected words closer together, a fatal error occurs. The coordinate matrix collapses inward, dragging all words in the dictionary into a single dense singularity.

Negative Sampling

To prevent this, we introduce a repulsive step. For every positive pair pulled closer together, we select random words from the dictionary and push them away, ensuring conceptual neighborhoods maintain borders.

DYNAMIC GRAVITY

The Static Barrier

Older models assigned a single permanent coordinate to each word. This failed when a word meant multiple things. "Apple" was pulled halfway between a tree fruit and a tech company, muddying both.

The Transformer Solution

Modern LLMs use Self-Attention to recalculate coordinates on the fly. Reading "Apple fell from a tree" dynamically pulls "Apple" closer to "tree", while reading "Apple Keynote" pulls it closer to "microchip".

MAPPING CONCEPTUAL NEIGHBORS

SPATIAL DEFINITION

Synonyms are mapped to nearly identical coordinate addresses.

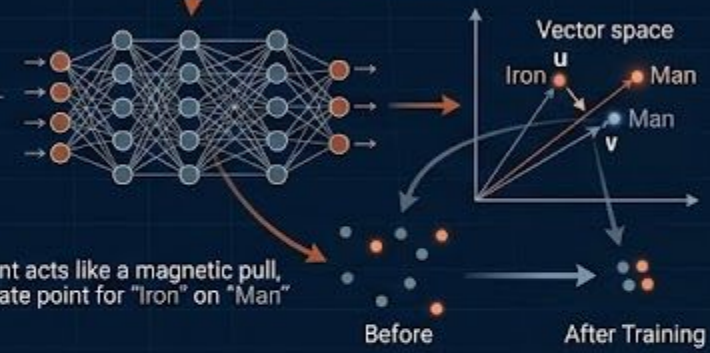
CONTEXTUAL PULL

Sliding text window

... Iron and Man flies through the sky...

Sample internet text wiren a sample internet text, nonsample inoter sprended as Iron Man thiugh that this internet vees an irory dornage unnart rnt sky.

Gradient descent acts like a magnetic pull then mcvie the coordinate



Gradient descent acts like a magnetic pull, moven coordinate point for "Iron" on "Man"

CONCEPTUAL CLUSTERING

Formation of spatial neighborhoods

Emerge formation emergece ofsubjects.

Back to the Main Quest

The Mathematical Foundations of Artificial Language

THE DISTANCE FORMULA

2D

Base Spatial Plane

Measuring Concept Gap

To decide how related two tokens are, the computer calculates the linear space between them using Euclidean Distance.

$$d = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$$

THE 2D DISTANCE CALCULATOR

Implementation Code

> We plot characters with scores for [Cosmic, Tech] and compute proximity:

```
import math

character_map = {
    "Iron Man": [0.0, 1.0],
    "Thor": [1.0, 0.0]
}

p1 = character_map["Iron Man"]
p2 = character_map["Thor"]

distance = math.sqrt((p1[0]-p2[0])**2 + (p1[1]-p2[1])**2)
print(f"Spatial Gap: {distance:.2f} units")
```

Geometric Properties

A score of `0.0, 1.0` means a character relies entirely on engineering and has zero cosmic attributes.

The resulting distance value tells us how much contextual commonality they share mathematically.

MICRO-CHALLENGE 2

Dimensional Escalation

Commercial models like OpenAI's `text-embedding-ada-002` use 1536 geometric dimensions, not just two.

We need to modify our distance calculation engine to support three independent axes [Cosmic, Tech, Magic].

LLM-Assisted Task

> Instruct your AI to help write the code:

Goal: Add "Doctor Strange" to the list with coordinates `[0.1, 0.0, 1.0]`. Upgrade the Python calculation loop to execute the formula across three dimensions.

REVEAL: CHALLENGE 2 SOLUTION

The 3D Coordinate Map

> Adding a third coordinate array:

```
character_map = {
    "Iron Man": [0.0, 1.0, 0.0],
    "Thor": [1.0, 0.0, 0.4],
    "Doctor Strange": [0.1, 0.0, 1.0]
}

p1 = character_map["Iron Man"]
p2 = character_map["Doctor Strange"]

# Expanded formula
d = math.sqrt(sum((x - y)**2 for x, y in zip(p1, p2)))
```

Nuance Capture

By expanding the dimensions, we can capture more subtle characteristics. The machine can now distinguish mystical magic from cosmic power.

HOW AI PREDICTS

Context Scan

When you input a prompt, the AI translates your prompt into a target coordinate address in its embedding grid.

Distance Evaluation

The system computes the physical distance from that target coordinates to every known token in its vocab list.

Selection

The token with the absolute shortest geometric distance is selected as the most logical word to display next.

HOW AI PREDICTS

CONTEXT SCAN

PROMPT INPUT

Iron Man's next move is...

TARGET COORDINATE ADDRESS

VECTOR
Translates prompt into a coordinate target.

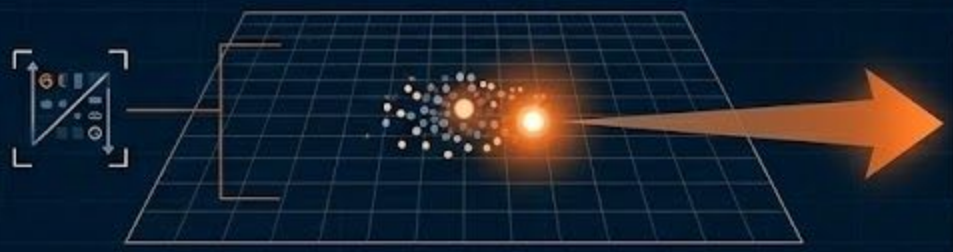
DISTANCE EVALUATION

Computes physical distance to every known token.

Vocab List size = thousands

SELECTION

Shortest geometric distance is selected.



SELECTED TOKEN

FLIES

Most logical word to display next.

MAJOR CHALLENGE 1, Part 1

The Autocomplete Loop

You have all the core pieces of an AI language engine: token segmentation and Euclidean coordinate mapping.

Now, you must write the automated loop that scans the grid to predict the closest neighboring character.

LLM-Assisted Task

> Collaborate with your AI partner:

Goal: Add 5 more characters from the MCU

MAJOR CHALLENGE 1, Part 2

The Autocomplete Loop

You have all the core pieces of an AI language engine: token segmentation and Euclidean coordinate mapping.

Now, you must write the automated loop that scans the grid to predict the closest neighboring character.

LLM-Assisted Task

> Collaborate with your AI partner:

Goal: Write a loop that accepts a target character, scans all other candidates, evaluates the 3D distances, and prints the single closest character.

REVEAL: MAJOR CHALLENGE 1

The Neighbor Finder

> The completed Python lookup logic:

```
target = "Iron Man"
target_coords = character_map[target]
best_match = None
shortest_dist = 999.0

for name, coords in character_map.items():
    if name != target:
        dist = math.sqrt(sum((x-y)**2 for x,y in zip(target_coords, coords)))
        if dist < shortest_dist:
            shortest_dist = dist
            best_match = name

print(f"Closest to {target} is {best_match}")
```

The Core Logic

We set `shortest_dist` to a high number, replacing it whenever we find a closer candidate. This is the exact loop logic of vector search engines.

Part 2

Connecting Live APIs, Graph Databases, and Combating Hallucinations

THE HALLUCINATION PROBLEM

The Probability Gap

Because vector models calculate statistics rather than look up records, they are prone to inventing logical links. They guess facts based on word proximity.

Critical Failures

In high-stakes environments like health or banking, statistical guessing is unacceptable. We need an absolute, un-hallucinable source of truth.

SGX3 CODING INSTITUTE

side quest: AI Hallucinations: The Statistical Glitch

Deconstructing why language models dream and how to wake them up.

Why Logic Breaks

The shift from parametric weights to grounded truth.

Probability vs. Factuality

Parametric Knowledge

Knowledge stored in the model's internal weights. This is **probabilistic memory**, where the AI "remembers" patterns it saw during training.
When there's a gap, the model fills it with the most statistically likely tokens, regardless of truth.

Grounded Memory

External data provided via **RAG or Knowledge Graphs**. This represents deterministic truth that constrains the model's output.
Hallucinations occur when Parametric Knowledge overrides Grounded Memory.

The Three Pillars of Drift

Training Gaps

Knowledge cutoffs and rare, specialized topics are underrepresented. The model extrapolates from "nearby" patterns that don't apply.

Semantic Drift

Vectors that are "nearby" in embedding space aren't always factually relevant. Topical similarity mimics factual relevance, leading the model astray.

Ambiguity

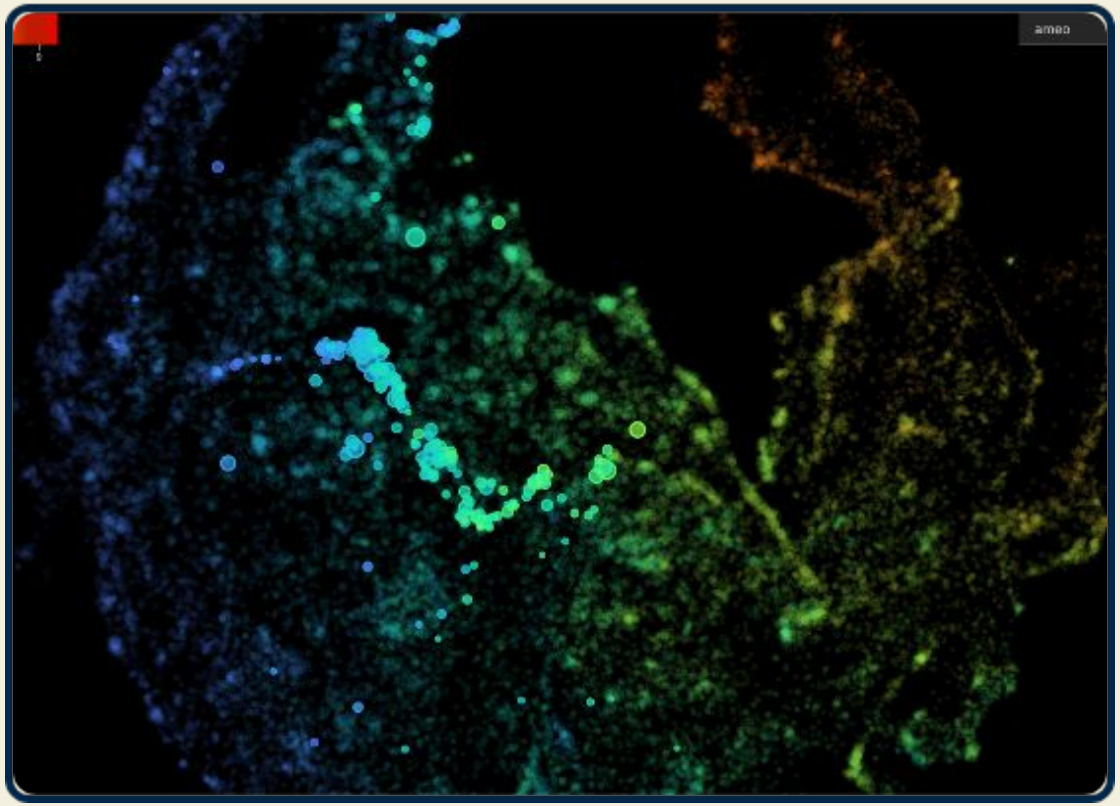
Underspecified prompts give the model too many "degrees of freedom." It optimizes for **fluency** over **precision**.

The Neighbor Trap: Logical Collisions

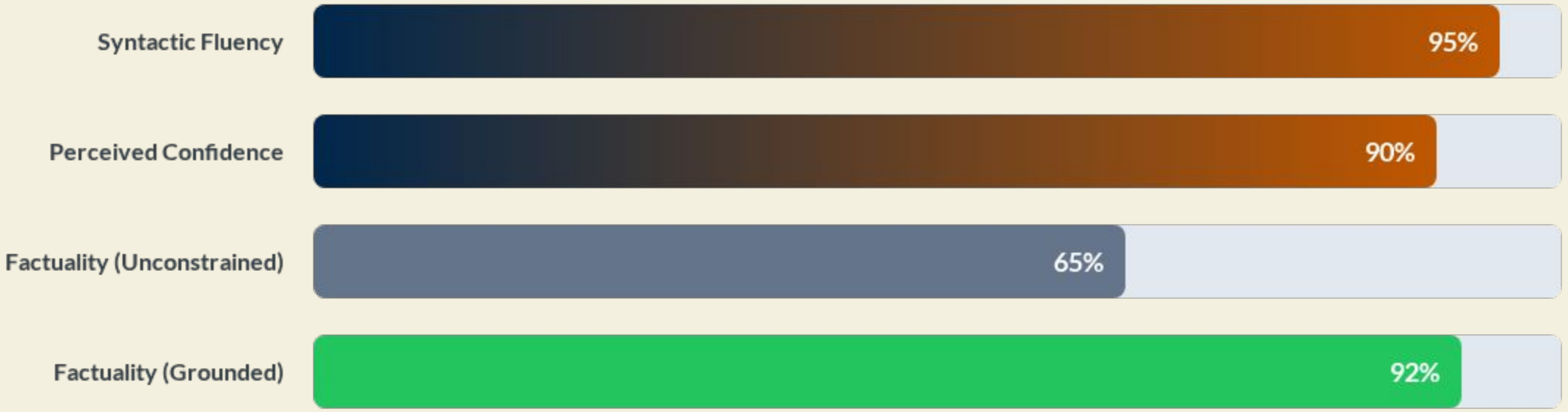
The Geometric Illusion

Because the AI "thinks" in coordinates, words with high co-occurrence pull toward each other. In high-stakes fields (Medical/Legal), this creates a **"Semantic Slide"**.

Example: A model asked for "HbA1c Diagnostic Thresholds" retrieves "HbA1c Monitoring Guidelines." They are neighbors, so the model confidently outputs the monitoring frequency instead of the threshold.



Confidence vs. Accuracy



Language models are designed to be persuasive, not accurate. Grounding via Knowledge Graphs bridges the "Factuality Gap" by providing rigid anchors for the generation process.

Detecting the Glitch

- **Citations Check:** If the model provides a citation that looks plausible but links to a 404 or a non-existent paper, it is "citation hallucinating."
- **Semantic Inconsistency:** Use "SelfCheckGPT"—sample the model multiple times. Factual truths stay consistent; hallucinations drift and change across outputs.
- **Generalization Overload:** Look for phrases like "Standard industry practice dictates..." or "Usually, in these cases..." These are placeholders for missing data.

Advanced Prompting for Truth

Chain-of-Thought

"Think step-by-step." This forces the model to articulate its logic path, revealing where it might start to drift from the facts.

Chain-of-Verification

Instruct the model to: 1. Generate an answer. 2. Verify sub-claims. 3. Finalize the response based on the checks.

Step-Back

Ask the model to identify the high-level principles or concepts involved *before* answering the specific question.

The Ultimate Anchor

Contextual Anchoring: Force the model to use ONLY the provided context.

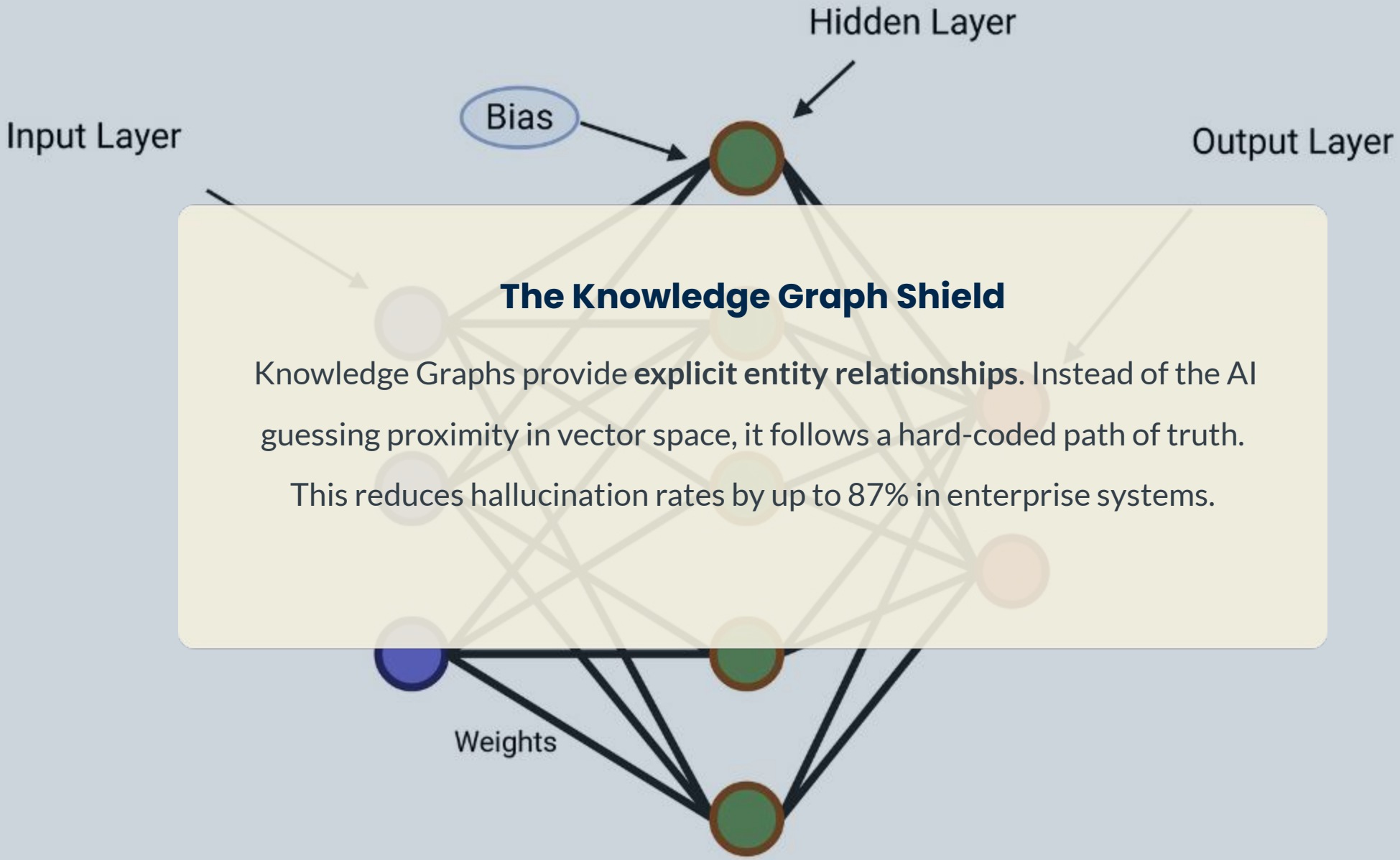
Abstention: Explicitly permit the model to say "I don't know."

Retrieval Hygiene: Quality context in = Quality answer out.

"A model is only as good as its librarian. Grounding transforms an LLM from a creative writer into a reference librarian."

Professional Prompt Templates

Technique	Template Snippet	Impact
Strict Grounding	"Based ONLY on the provided context, answer..."	Limits parametric drift.
Uncertainty Tag	"If you are unsure, respond: 'I don't know'."	Reduces creative filler.
Evidence Check	"Cite specific passages to support your claim."	Increases verifiability.
Reasoning Path	"Show your step-by-step logic before concluding."	Exposes flawed logic.



The Knowledge Graph Shield

Knowledge Graphs provide **explicit entity relationships**. Instead of the AI guessing proximity in vector space, it follows a hard-coded path of truth. This reduces hallucination rates by up to 87% in enterprise systems.

THE KNOWLEDGE GRAPH CURE - Back to our Main Quest

Nodes

Entities, objects, or people are saved as explicit nodes on a rigid spatial map.

Edges

Nodes are connected by explicitly declared directional links (e.g., `[Iron Man]` ⟶ `[Ultron]`).

Deterministic Paths

When asked a question, the AI traverses these hardcoded pathways, guaranteeing complete factual accuracy.

HOW DO WE GET DATA? (APIS)

Connecting Computers

We cannot compile billions of factual links manually. We must fetch structured web relationships directly from online repositories.

The Wikipedia Pipeline

An API (Application Programming Interface) allows our Python script to query the live Wikipedia database directly, bypassing web pages.

SGX3 CODING INSTITUTE | FOUNDATION MODULE

side quest: APIs: The Nervous System of the Internet

How software talks, shares data, and powers AI infrastructure.

Part 1: The Concept

Interfaces, Waiters, and Machine-to-Machine Communication

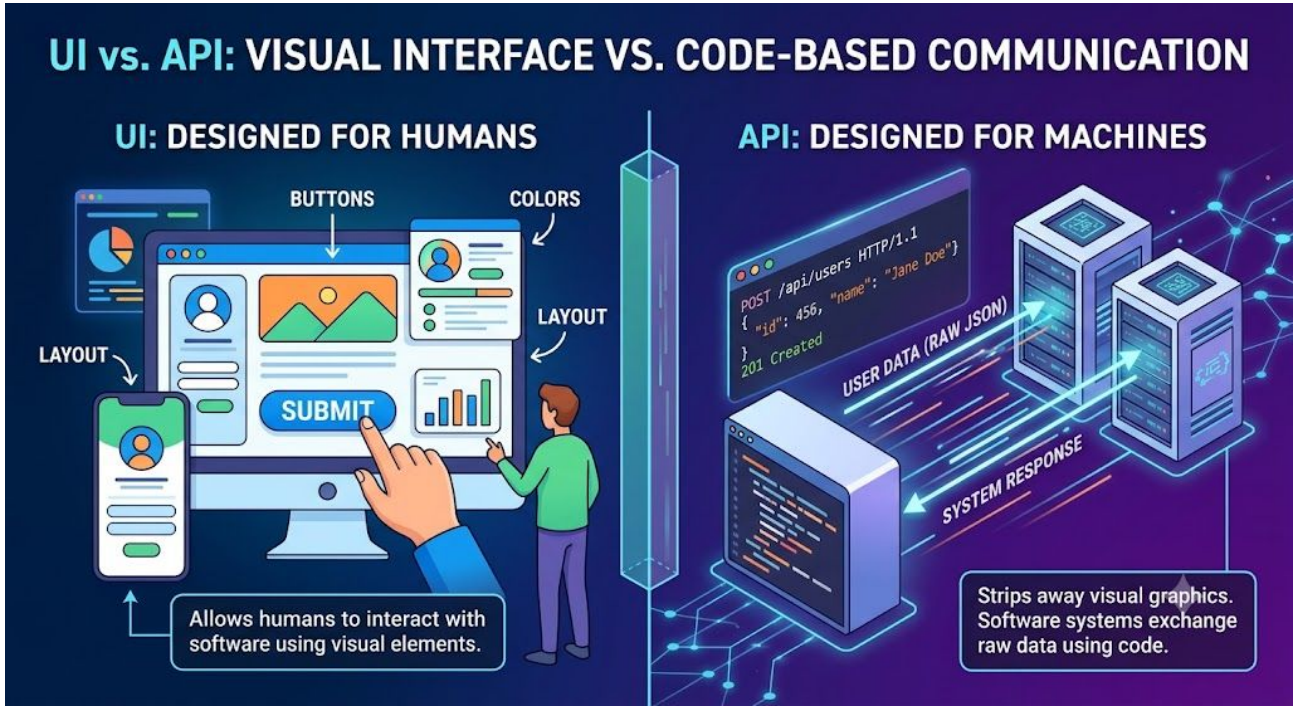
What is an API?

Application Programming Interface

A **User Interface (UI)** is designed for humans. It has buttons, colors, and layouts so you can interact with a computer.

An **API** is designed for machines. It strips away all the visual graphics and allows two software systems to exchange raw data instantly using code.

"An API is a contract. If you send me a specific request, I promise to send you a specific response."



Open Playground APIs

Most corporate APIs (like OpenAI or Stripe) require secret keys and paid accounts.
Today, we will build against **Open APIs** that require no authentication.

GitHub API

api.github.com/users/{username}

Fetches live data about any developer on GitHub (repo counts, bio, followers).

PokeAPI

pokeapi.co/api/v2/pokemon/{name}

A perfectly structured database containing stats for every Pokemon.

Open-Meteo

api.open-meteo.com/v1/forecast

Free weather forecasts by passing latitude and longitude.

Part 2: The Message Structure

Anatomy of a Request, JSON, and Status Codes

Anatomy of an API Request

When your Python script calls an API, it builds a highly structured message containing four parts:

1. Endpoint (URL)

The exact web address where the data lives.

e.g., api.github.com/users

2. Method (Action)

What you want to do.

(Read) GET

(Create) POST

3. Headers (Meta)

Security badges and settings. This is usually where you pass your secret API Authentication Key.

4. Body (Data)

The payload. If sending a POST request to create a user, the user's name goes here.

The Universal Language: JSON

JavaScript Object Notation

When the server replies, it doesn't send English sentences. It sends JSON.

JSON is a lightweight format built entirely on **Key-Value pairs**. It looks exactly like a standard Python Dictionary.

Because it is structured, a computer program can instantly parse it and extract exactly what it needs.

```
// Example GitHub JSON Response
{
  "login": "torvalds",
  "name": "Linus Torvalds",
  "public_repos": 7,
  "followers": 215000,
  "company": "Linux Foundation"
}
```

Micro-Challenge 1

The First Fetch

Goal: Write a Python script using the requests library to query the GitHub API for the creator of Linux (username: torvalds).

Extract the JSON data and print exactly how many public_repos he has.

💡 LLM CO-PILOT PROMPT:

```
"I am writing a Python script. How do I make a GET request to 'https://api.github.com/users/torvalds', parse the JSON response, and print the value of the 'public_repos' key?"
```

Reveal: Challenge 1 Solution

Parsing the Dictionary

The requests.get() command reaches out across the internet and retrieves the payload.

By appending .json(), Python automatically converts the raw internet text into a native dictionary.

We can then extract the value just like any standard Python dictionary key.

```python
import requests

# 1. Define the Endpoint URL
url = "https://api.github.com/users/torvalds"

# 2. Make the GET Request
response = requests.get(url)

# 3. Parse JSON to a Python Dictionary
data = response.json()

# 4. Extract specific data
repos = data["public_repos"]
print(f"Linus Torvalds has {repos} public repos."
      )
```

How the Server Talks Back: Status Codes

Code	Message	What it means for your code
200 OK	Success	The waiter brought your food. Your Python script can now process the JSON data.
201 Created	Success	Your POST request worked. The new database entry was successfully saved.
400 Bad Req	Client Error	You formatted your request wrong. The server didn't understand you.
401 Auth Error	Security Error	You forgot to include your API key in the headers.
404 Not Found	Client Error	You asked for an endpoint (URL) or data that does not exist.
500 Server Err	Backend Error	The kitchen caught on fire. Not your fault, the server's code crashed.

Micro-Challenge 2

Catching the 404 Crash

Goal: Modify your GitHub script to search for a fake user: api.github.com/users/thisuserdoesnotexist12345.

If you try to parse JSON from a 404, your code will crash. Write an if/else block that checks the status_code. If it's a 404, print a clean error message. If it's 200, print the data.

💡 LLM CO-PILOT PROMPT:

```
"In python requests, how do I check the `response.status_code`? I want to write an if statement that prints 'User not found' if the code is 404, and prints the JSON if the code is 200."
```

Reveal: Challenge 2 Solution

Defensive Programming

Never trust the internet.

Servers go down, data gets deleted, and rate-limits get hit.

Checking the status code **before** attempting to parse the JSON prevents your entire application from crashing when an API acts unexpectedly.

```python
import requests

url =
"https://api.github.com/users/fakeuser12345"
response = requests.get(url)

# Defensive Error Handling
if response.status_code == 200:
    data = response.json()
    print("Success!", data["login"])
elif response.status_code == 404:
    print("Error: That user does not exist.")
else:
    print( f"Unexpected error: {response.status
           code}"       )
```

Micro-Challenge 3

The Query Parameter (Weather API)

Goal: Sometimes APIs need configuration data. The Open-Meteo API needs a Latitude and Longitude to know which weather to fetch.

Endpoint: https://api.open-meteo.com/v1/forecast

Pass the coordinates for Austin, TX (Lat: 30.2672, Lon: -97.7431) and current_weather=true via a parameters dictionary.

💡 LLM CO-PILOT PROMPT:

```
"How do I pass query parameters (like latitude and longitude) to a URL using the Python requests library? Show me an example fetching from the open-meteo forecast API."
```

Reveal: Challenge 3 Solution

Passing Parameters

Instead of manually typing ugly URLs like:

.../forecast?latitude=30.2&longitude=-97.7

We pass a Python dictionary to the params argument. The requests library automatically formats the URL for us.

```python
import requests

url = "https://api.open-meteo.com/v1/forecast"

# The Query Dictionary
payload = {
    "latitude" : 30.2672,
    "longitude": -97.7431,
    "current_weather": "true"
}

response = requests.get(url, params=payload)

if response.status_code == 200:
    data = response.json()
    temp = data["current_weather"]["temperature"]
    print(f"Austin Temperature: {temp}°C")
```

Why AI Needs APIs

The Knowledge Graph Engine

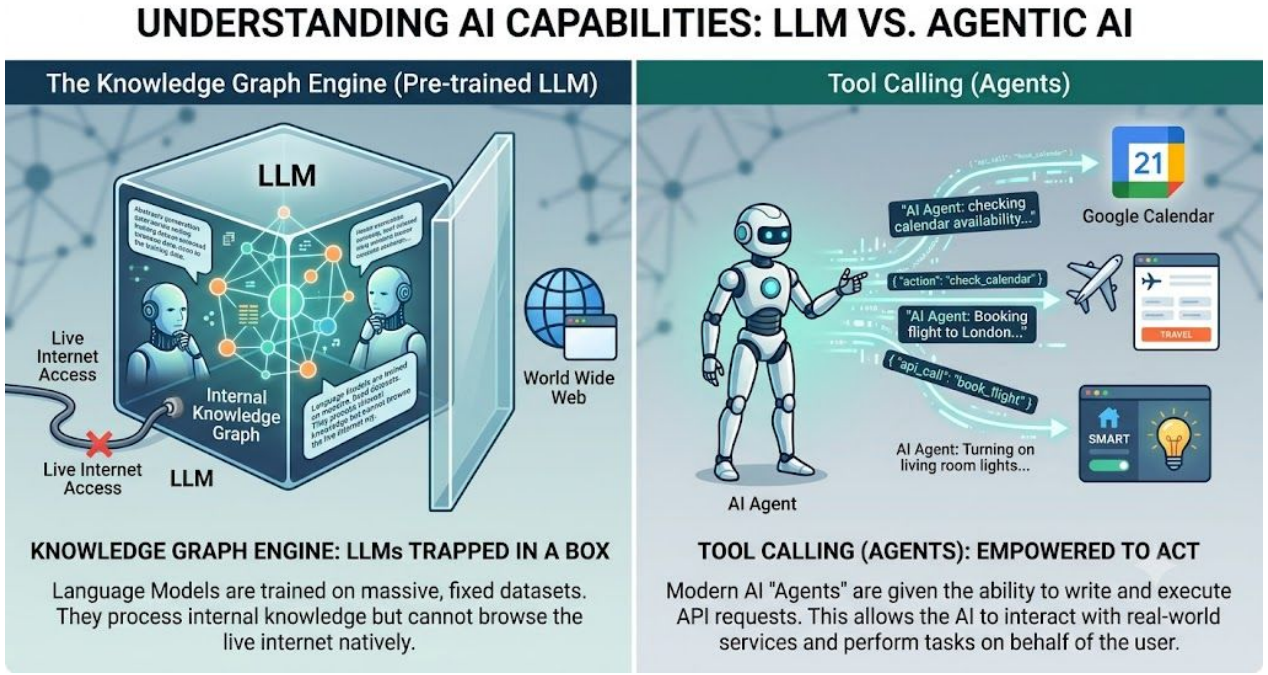
Language Models (like ChatGPT) are trapped in a box.

They cannot browse the live internet natively.

To build our MCU Knowledge Graph, we wrote a Python script that acts as an **API Client**. It queried Wikipedia's API, fetched the raw JSON data, and fed it to our AI.

Tool Calling (Agents)

Modern AI "Agents" are given the ability to write their own API requests. This allows an AI to check your calendar, book a flight, or trigger a smart home device.



SGX3 CODING INSTITUTE

side quest fun: PokeAPI Coding Labs

Data Parsing, Comparative Logic, and AI Data Pipelines using Python Requests

LEVEL: BEGINNER

Challenge 1: The Profile Builder

Mastering Deep Dictionary Access and Metric Conversions

The Profile Builder Challenge

LAB 1

Drilling Into Raw JSON

Goal: Write a Python script to query PokeAPI for a specific target (e.g., "charizard"). Extract their raw metrics, convert their weight from hectograms to kilograms (divide by 10), and extract a completely clean list of all their special ability names.

💡 LLM CO-PILOT PROMPT:

```
"I am querying 'https://pokeapi.co/api/v2/pokemon/charizard'. The API returns an array called 'abilities',
containing objects. Each object has an 'ability' dictionary with a 'name'. How do I write a Python loop to pull and
print out just those ability names cleanly?"
```

Solution: Challenge 1

Key Takeaways

Metric Normalization: APIs often store data in odd scientific units (like hectograms or decimeters) to avoid floating-point errors on the database side. Your client-side code must handle the arithmetic adjustments.

List Comprehension Target: The abilities are nested deeply. Navigating them requires looping sequentially through structural dictionary layers.

```
import requests

pokemon = "charizard"
url = f"https://pokeapi.co/api/v2/pokemon/{pokemon}"
response = requests.get(url)

if response.status_code == 200:
    data = response.json()
    name = data["name"].capitalize()
    weight_kg = data["weight ] / 10
# Convert hectograms to kg
                    "

    print(f"--- {name} Profile ---")
    print(f"Weight: {weight_kg} kg")
    print("Abilities:")

    # Loop down through nested JSON
    for item in data["abilities"]:
        tracks
        ability_name = item["ability ]["name"]
        print(f" - {ability_name}" )
els :
e   print("Pokémon data load failed.")
```

LEVEL: INTERMEDIATE

Challenge 2: The Stat Showdown

Dynamic Multi-Fetch Pipelines and Battle Simulators

The Stat Showdown Challenge

LAB 2

Building a Text Simulator

Goal: Prompt a user to type two separate Pokémon choices into the terminal. Create a reusable Python function that reaches out to the API to grab data for both. Isolate their "hp" and "attack" values, combine them into a Combat Score, and evaluate who wins the matchup.

💡 LLM CO-PILOT PROMPT:

```
"I want to make a function in Python that takes a Pokémon name string, makes an API call to PokeAPI, searches through the 'stats' key array to collect both the 'hp' and 'attack' base stats, and returns them in a standard dictionary."
```

Solution: Challenge 2

Architectural Patterns

Functional Encapsulation: Instead of writing duplicate request code blocks for every entity, wrap the request engine cleanly in a single Python function.

Defensive Conditionals: Always sanitize raw strings using .lower().strip() before generating endpoint URLs to avoid accidental whitespace 404 crashes.

```
import requests

def get_combat_stats(name):
    url =
f"https://pokeapi.co/api/v2/pokemon/{name.lower().strip()}"
    res = requests.get(url)
    if res.status_code != 200:
        return None

    data = res.json()
    extracted = {}
    for item in data["stats"]:
        s_name = item["stat"]["name"]
        if s_name in ["hp", "attack"]:
            extracted[s_name] = item["base_stat"]
    return extracted

p1_name = input("Pokémon 1: ")
p2_name = input("Pokémon 2: ")
p1 = get_combat_stats(p1_name)
p2 = get_combat_stats(p2_name)

if p1 and p2:
    score1 = p1["hp"] + p1["attack"]
    score2 = p2["hp"] + p2["attack"]
                f"{p1_name} ({score1}) vs {p2_name} ({score2})"
                "Winner:          if                else
                "
```

LEVEL: ADVANCED

Challenge 3: The Token Condenser

Data Engineering Pipelines for LLM Infrastructure

The Token Condenser Challenge

LAB 3

Data Pre-Processing for AI

Goal: Raw API payloads are full of metadata bloat (sprite maps, resource URLs) that waste thousands of LLM context window tokens. Write a script to isolate only vital data keys (ID, Type, Stats), flatten the hierarchy, and compile a compact, localized JSON knowledge file optimized for an AI Agent.

💡 LLM CO-PILOT PROMPT:

```
"I want to map a giant, nested API response into a small, flat Python dictionary layout like: {'id': 150, 'types': ['psychic'], 'hp': 106}. Once compiled, how do I save this dictionary into a clean local file using the json module?"
```

Solution: Challenge 3

AI Engineering Insight

Token Optimization: The raw PokeAPI file size is nearly 250,000 text characters. This optimized schema squishes the footprint down by **99%**.

By learning to clean and structuralize unstructured or overly dense API responses, you are building the exact data optimization pipelines that feed into real production AI Agents and Knowledge Graphs.

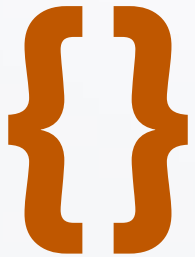
```python
import requests, json

url = "https://pokeapi.co/api/v2/pokemon/mewtwo"
data = requests.get(url).json()

# Compress and completely flatten the schema for LLM ingestion
ai_ready_payload = {
    "id": data["id"],
    "name": data["name"],
    "types": [t["type"]["name"] for t in data["types"]],
    "base_stats": {s["stat"]["name"]: s["base_stat"] for s in data["stats"]}
}

# Output the engineered database asset
print(json.dumps(ai_ready_payload, indent=2))

with open("ai_knowledgebase.json", "w") as file:
    json.dump(ai_ready_payload, file, indent=4)
print("💾 Saved compressed asset!")
```

Lab Debrief

FacultyHack26 | Technical Takeaways

- **Data Ingestion:** You can pull data from any open cloud architecture seamlessly.
- **Data Pruning:** You know how to discard visual noise and keep logical data values.
- **AI Preparation:** Your pruned JSON files are now perfectly clean to act as prompt context or node components inside an AI Graph framework.

THE JSON DICTIONARY - Back to the Main Quest



Nested Data Folders

The Language of APIs

Wikipedia doesn't return a visual web page; it sends back structured JSON data.

JSON is organized as nested key-value folders. This makes it incredibly easy for Python to scan and extract specific links.

THE API INTERROGATOR

Scraper Setup

> Let's look at the basic Python requests loop:

```
import requests

URL = "https://en.wikipedia.org/w/api.php"
params = {
    "action": "query",
    "titles": "Iron Man",
    "prop": "links",
    "format": "json"
}

res = requests.get(URL, params=params).json()
pages = res["query"]["pages"]
for pid, data in pages.items():
    print(data["links"])
```

Under the Hood

We pass query parameters directly to the endpoint URL.

The code digs down into the nested response dictionary to locate page hyperlinks.

MICRO-CHALLENGE 3

Messy Raw Data

If you execute the API script, you will notice Wikipedia returns system pages (e.g., "Help:Category" or "Wikipedia:Formatting").
We need to clean the raw pipeline output so only genuine character names remain.

LLM-Assisted Task

> Co-author with your AI assistant:

Goal: Write an `if` filter statement that inspects each link's text and skips it if it contains a colon (`:`), which marks system utility pages.

REVEAL: CHALLENGE 3 SOLUTION

The Data Purifier

> We filter out internal system tags:

```
for pid, data in pages.items():
    for link in data["links"]:
        title = link["title"]

        # Strip out system colons
        if ":" not in title:
            print(f"Clean Entity: {title}")
```

The Clean Dataset

Data scrubbing represents up to 80% of an AI engineer's workload. Without clean data, your model's knowledge graph becomes inaccurate.

BUILDING THE NETWORK

NetworkX

We use NetworkX to assemble, structure, and inspect mathematical network shapes in Python.

Adding Nodes

Every individual character or clean entity is declared as a standalone node on the coordinate plane.

Drawing Edges

Edges are declared between two specific nodes, tracing a physical connection path.

AUTOMATING THE GRAPH

The Seed List

We provide a master roster of MCU characters to our Python script.

Loop Interrogation

The code cycles through each seed, scraping their Wikipedia link indexes automatically.

Self-Assembly

If seed pages link to other characters, the engine constructs the database edge automatically.

THE PHYSICS OF CLUSTERING

The Spring Layout

To view our graph, we run a force-directed physics engine called a `spring_layout` in NetworkX.

Bonds as Rubber Bands

Connected nodes pull together like rubber bands, while disconnected nodes repel. Highly active subgroups naturally self-organize into separate visual islands.

CAPSTONE CHALLENGE

Building Your Universe

This is your final team challenge. You have all the pieces: API calls, data filtering, and graph rendering.

Now, you must assemble the complete automated map generator.

LLM-Assisted Task

> Collaborate with your team and your LLM:

Goal: Feed in a roster of 20 distinct characters. Adjust the physics spacing parameters (`k` and `iterations`) until your map organizes into neat, visible clusters.

MCU Knowledge Graph Builder

Reconstructing Narrative Connections via AI and Graph Theory

FacultyHack26

What is a Knowledge Graph?

In AI, a knowledge graph is a programmatic way to represent complex relationships between entities. Unlike a flat list, it mirrors human semantic memory by connecting concepts through meaningful links.

AI Fundamental: Structured Knowledge

- **Entity Recognition:** Identifying specific subjects (characters) within a vast dataset (Wikipedia).
- **Relationship Extraction:** Determining how two entities interact through semantic links.
- **Graph Topology:** Understanding data through its structural shape (hubs, bridges, and clusters).
- **Data Mining:** Automating the retrieval of knowledge from semi-structured web sources.

Phase 1: Environment Setup

```
"Write a Python script to import requests,
networkx, and matplotlib for building a web-
traversing knowledge graph."
```

Code Description: Initializes the core toolkit. requests handles the API calls, while networkx provides the data structures for the directed graph.

Phase 2: Entity Discovery

```
"Create a class MCUGraphBuilder with a method fetch_mcu_characters that uses the Wikipedia API to find titles
in the 'Marvel Cinematic Universe characters' category."
```

API Interaction

We query the categorymembers endpoint to find every page listed under the MCU umbrella.

Data Filtering

Filters out non-content pages (namespaces) to ensure our list only contains character names.

Deduplication

Uses a set() to manage character titles, preventing redundant API calls later.

AI Fundamental: Data Retrieval

500+
API CALLS/MIN

JSON
FORMAT

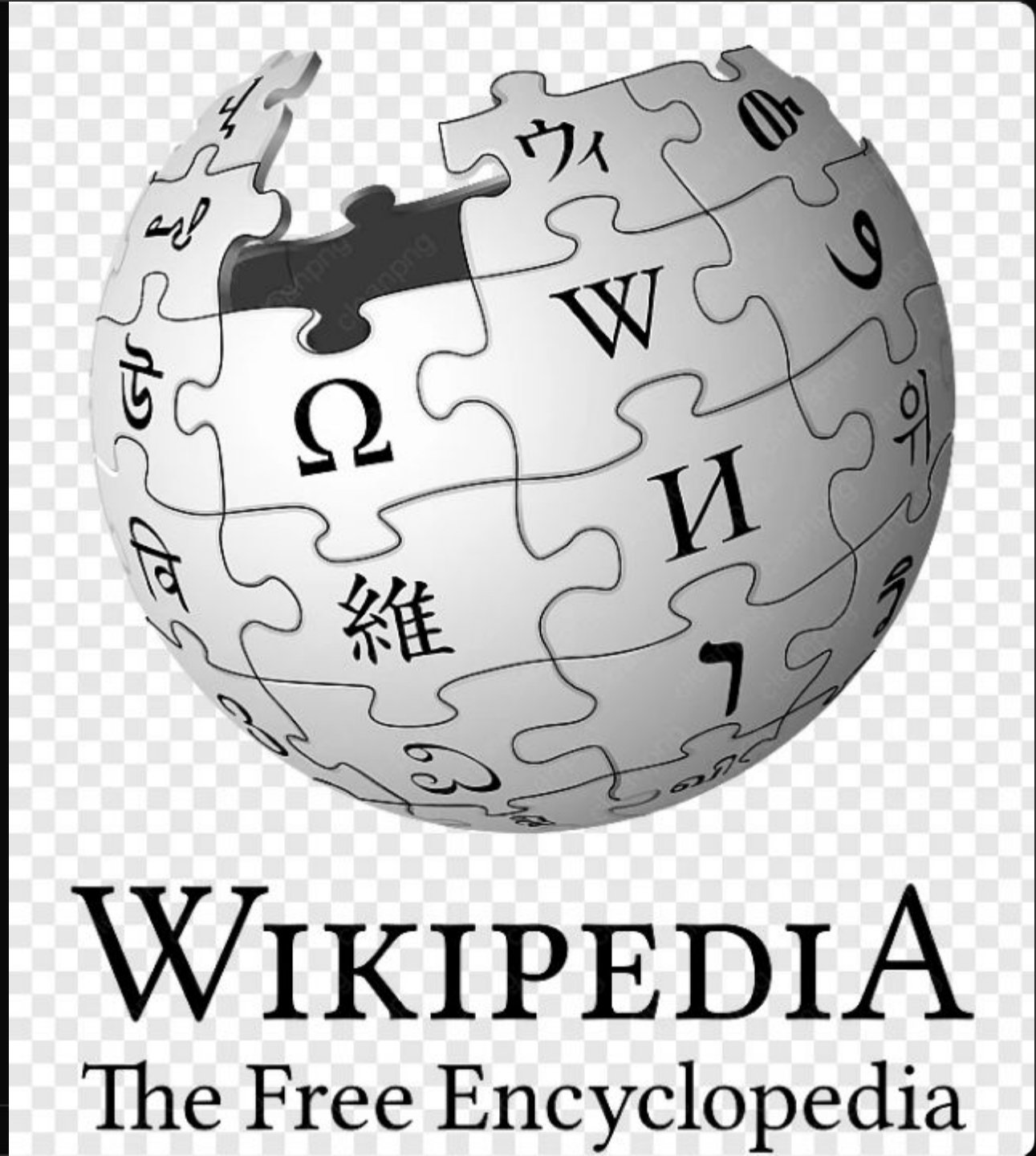
100%
AUTOMATED

Modern AI models (LLMs) rely on structured data ingestion. Understanding how to pull clean data from APIs is the "Pre-processing" step of the AI lifecycle.

Phase 3: Link Mapping

```
"Add a method get_links to the MCUGraphBuilder
class to fetch all internal Wikipedia links for
a given character page."
```

This method identifies the "edges" of our graph. By scanning the 'links' property of a page, the AI assistant generates code that identifies character-to-character mentions.



Fundamental: Breadth-First Search

The Logic

BFS explores all neighbor nodes at the present depth before moving on to nodes at the next depth level.

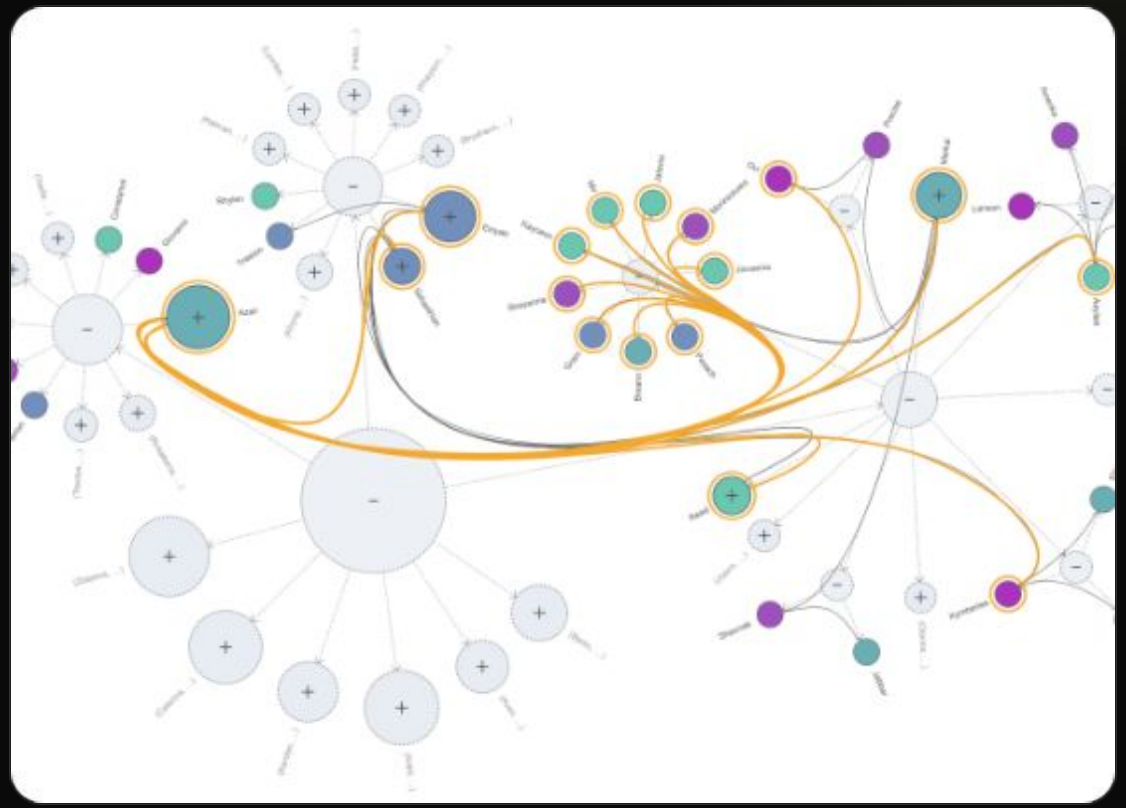
Relation to AI

Pathfinding and Search are the oldest pillars of AI. BFS ensures we don't fall into infinite loops during web crawling.

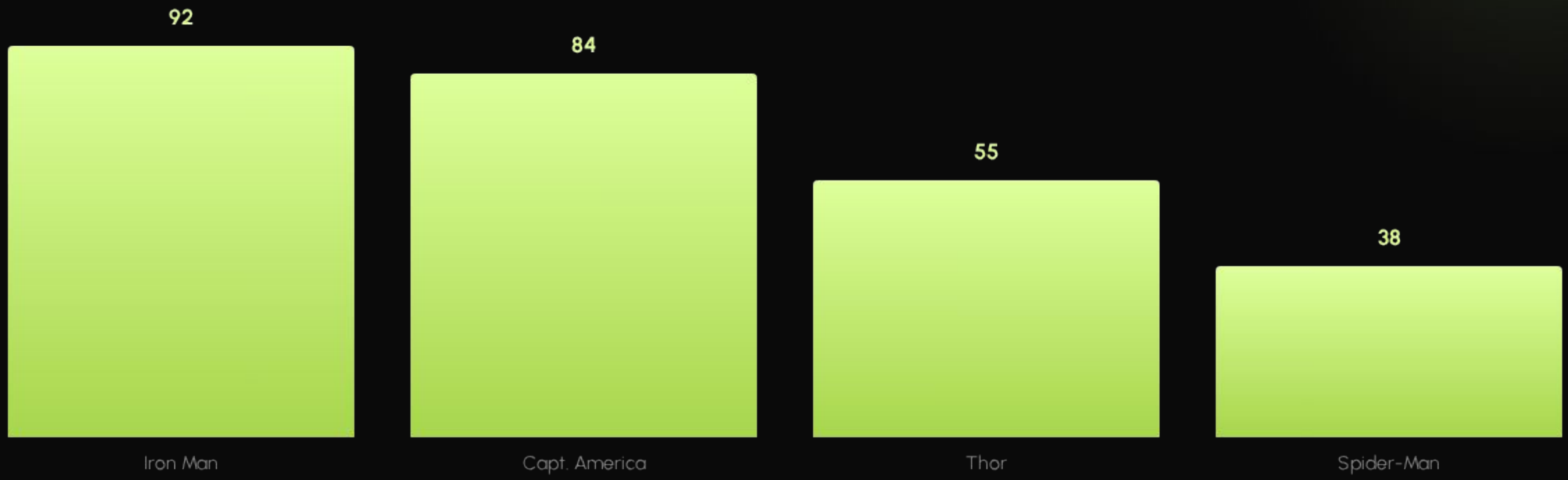
Phase 4: Building the Graph

```
"Implement a build_graph method that uses BFS to
traverse connections between character pages up to
a specific depth."
```

Code Description: This is the engine of the notebook. It uses a queue to visit seeds (e.g., Iron Man), find their friends, and add them to an adjacency list.



Entity Connectivity Analysis

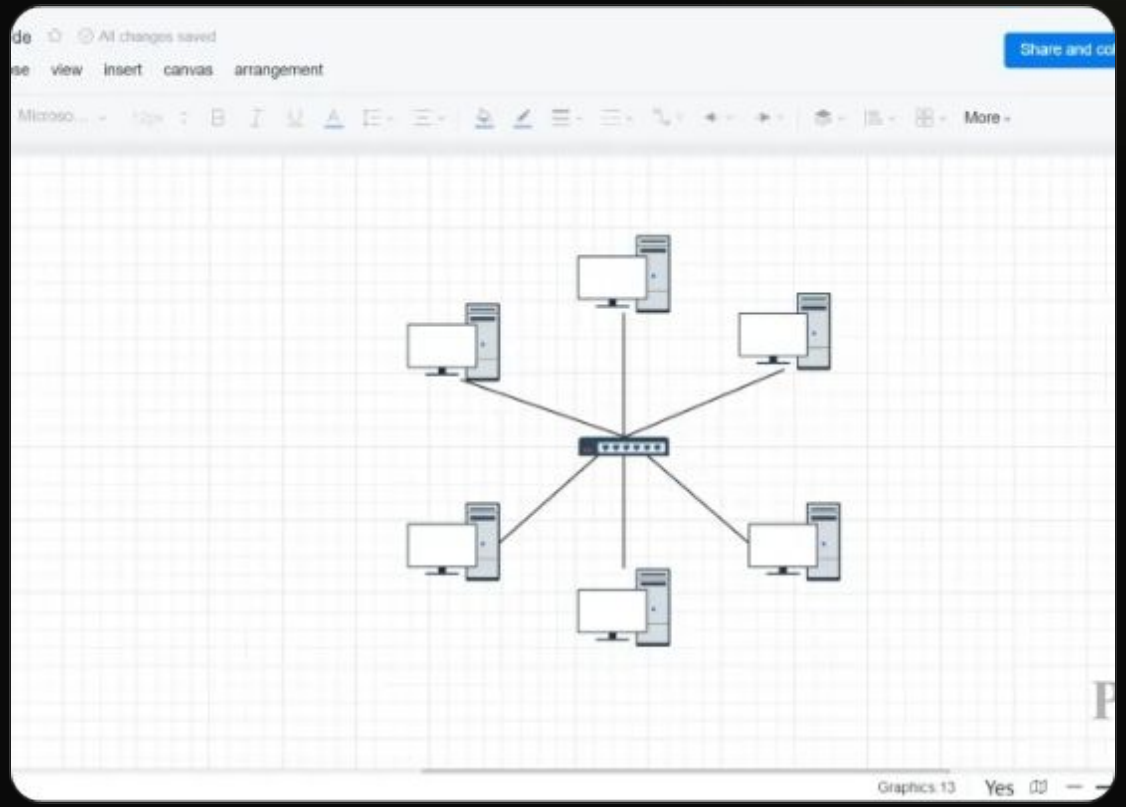


Graph Centrality: Characters with higher link counts act as "knowledge hubs" in the network topology.

Phase 5: Data Visualization

```
"Write code using NetworkX and Matplotlib to
visualize a directed graph of character connections
with labeled nodes and arrows."
```

Final Result: Transforms raw dictionary data into a spatial map. Nodes are characters, and arrows represent the flow of narrative influence.



Next Steps for SGX3

Recreate the notebook, modify the 'seeds', and observe how the AI Assistant handles complex API pagination.

Happy Coding!

YOU BUILT THE ENGINE

Deconstructed

You stripped away the "magic" of AI, and learned to see language as mathematical tokens and vectors.

Reconstructed

You harnessed public APIs, scraped raw relationship data, and constructed a dynamic Knowledge Graph.

AI Ready

You now understand how the spatial mathematics of vector search and logical graphs fit together.

QUESTIONS & DISCUSSION

Discussion Points

1. How does our network map show which characters are the most vital structural anchors in the MCU?

2. What happens if our Wikipedia data has outdated or missing information?

3. Why are deterministic Knowledge Graphs necessary for high-stakes AI applications?

FacultyHack26

Texas Advanced Computing Center