

edvantis

Your Engineers Use AI Daily.  
Here's How to Get More from It  
Webinar Presentation

# Agenda



Market Context: Why Now?

---

What Is an AI Agent?

---

Where Agents Create Advantage

---

AI Agents: Functions

---

Problems: Error Accumulation

---

Tools: Rule of Thumb

---

Challenges and Conclusions

---

# About the Speaker

## Sergii Shelpuk

---

- Worked in AI & Data Science since 2013
- Led AI & Data Science departments for 4 companies, including a billion-dollar one
- Helped raise \$52.5M in venture founding as a management team member
- Co-founded an AI product for biopharma R&D
- Graduated from LSE and Stanford



**Sergii Shelpuk**

Edvantis AI Consulting Expert,  
Global Strategy and  
Implementation

# About Edvantis

Edvantis is a global software engineering company helping businesses stay competitive in the digital world.

We modernize infrastructure, automate processes, and deliver AI-driven solutions to improve efficiency, agility, and growth.



20+ years of experience



400+ employees



450+ delivered projects



40+ active clients



ISO/IEC 27001:2013  
Certified



ISO 9001:2015  
Certified



2025 | Clutch Global  
Awards Winner



Software Development  
Services Grid Leader | G2

# AI in Software Development: **Where We Are**

67%

of developers use AI tools

51%

use them daily

64%

do not perceive AI as a job threat

## Evolution of Software Engineering: Agentic Shift

Pre-2023



### IDE Era

- Manual orchestration of all logic and architecture
- Deterministic static analysis (IntelliSense)
- Tools flagged errors but could not generate or repair code

2023-2024



### Assistant Era

- Inline autocomplete + chat interfaces
- Context limited to open files – no repo-wide awareness
- Developer as "orchestrator," AI for snippet generation

2025+

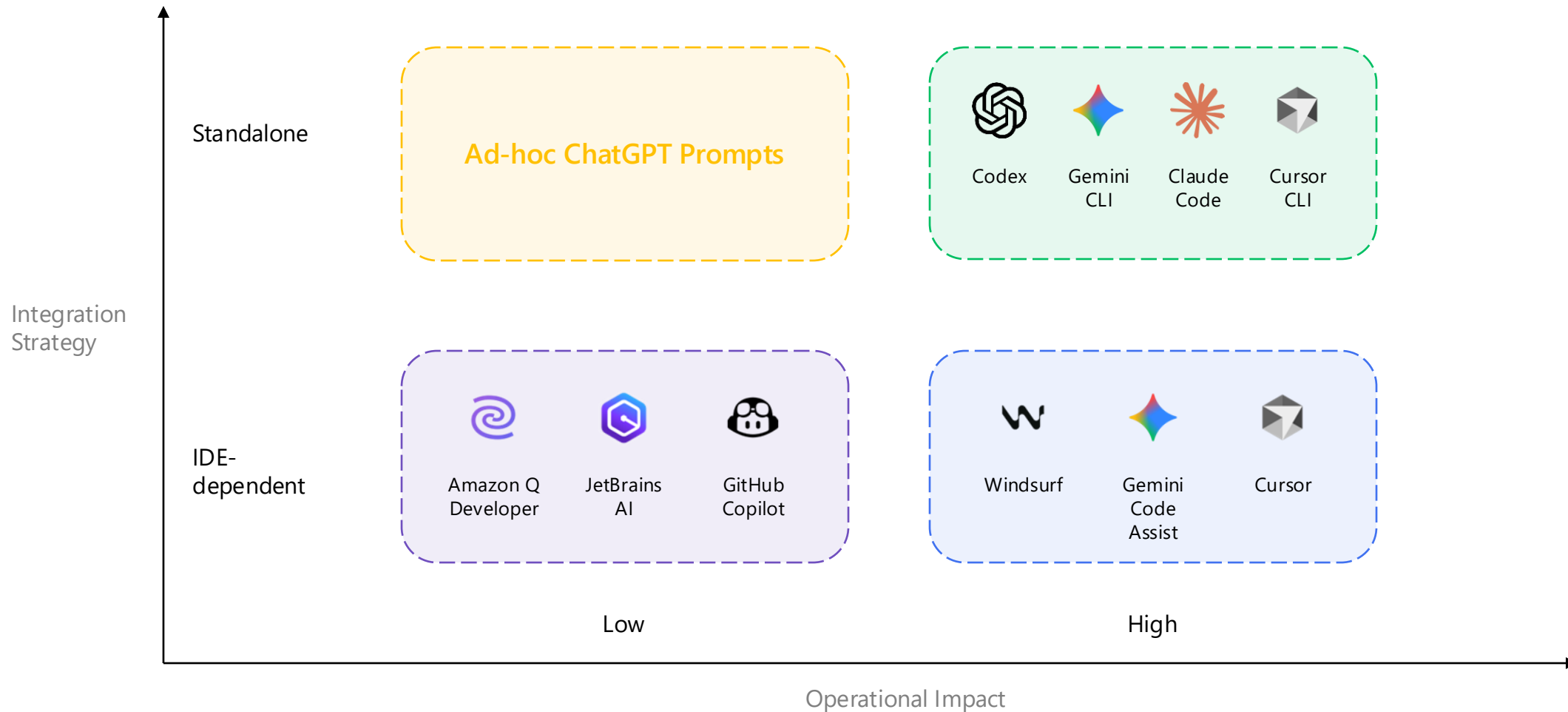


### Agentic Era

- Agents plan, edit, and verify multi-file tasks autonomously
- AI-native editors index the full codebase
- Developer shifts from "Writer" to "Architect / Reviewer"

Benchmarks vs Reality: AI coding tools perform well on public datasets, but on private, uncontaminated commercial codebases, performance drops to 20-30%. Likely due to training data leakage and underperformance on less popular languages and technologies.

# AI Tools Strategy Matrix



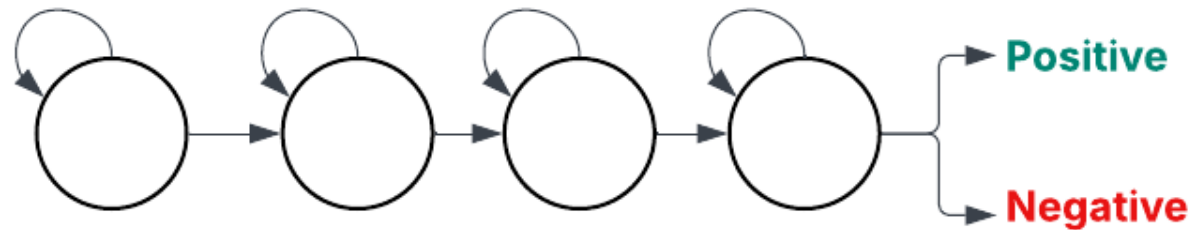
edvantis

# LLMs, Agents, Coding Tools

# Sentiment Analysis

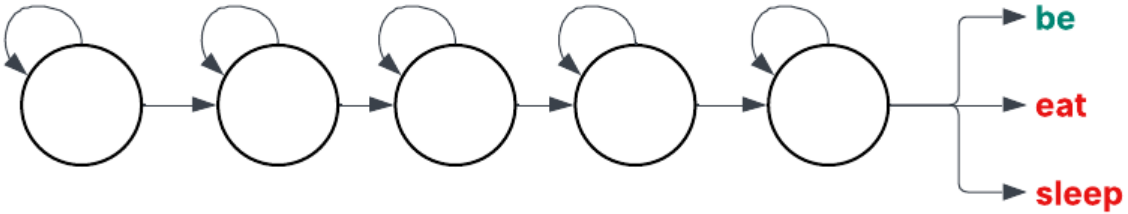
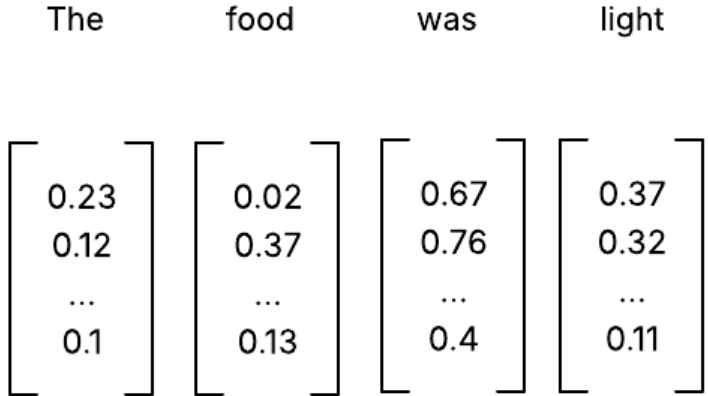
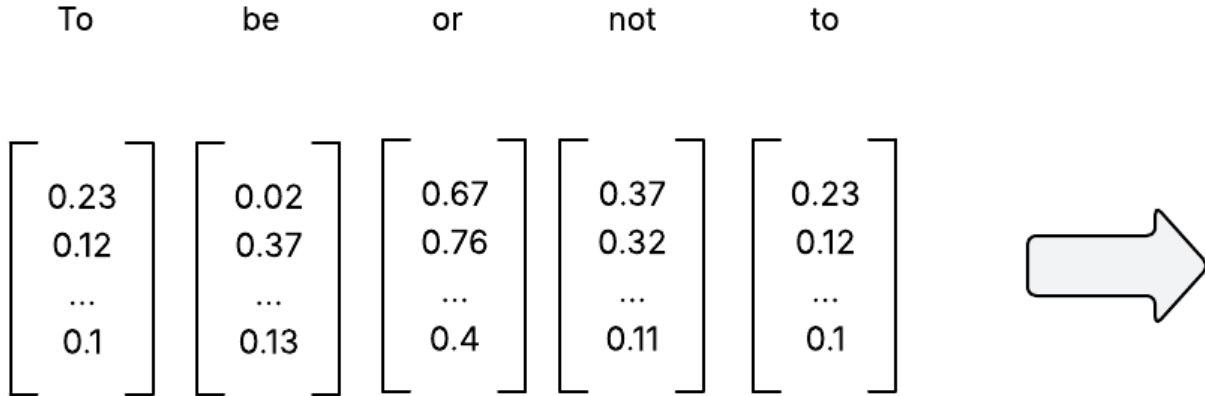
The food was light

$\begin{bmatrix} 0.23 \\ 0.12 \\ \dots \\ 0.1 \end{bmatrix}$	$\begin{bmatrix} 0.02 \\ 0.37 \\ \dots \\ 0.13 \end{bmatrix}$	$\begin{bmatrix} 0.67 \\ 0.76 \\ \dots \\ 0.4 \end{bmatrix}$	$\begin{bmatrix} 0.37 \\ 0.32 \\ \dots \\ 0.11 \end{bmatrix}$
--	---	--	---

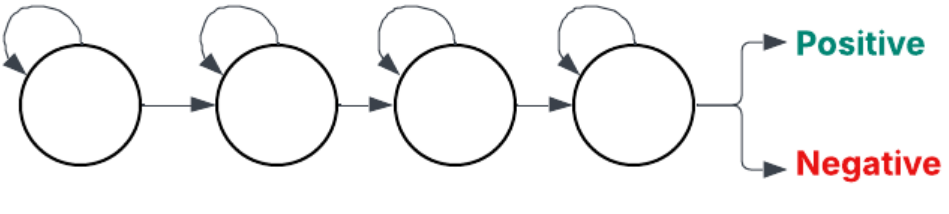


Recurrent Neural Network

# Language Models

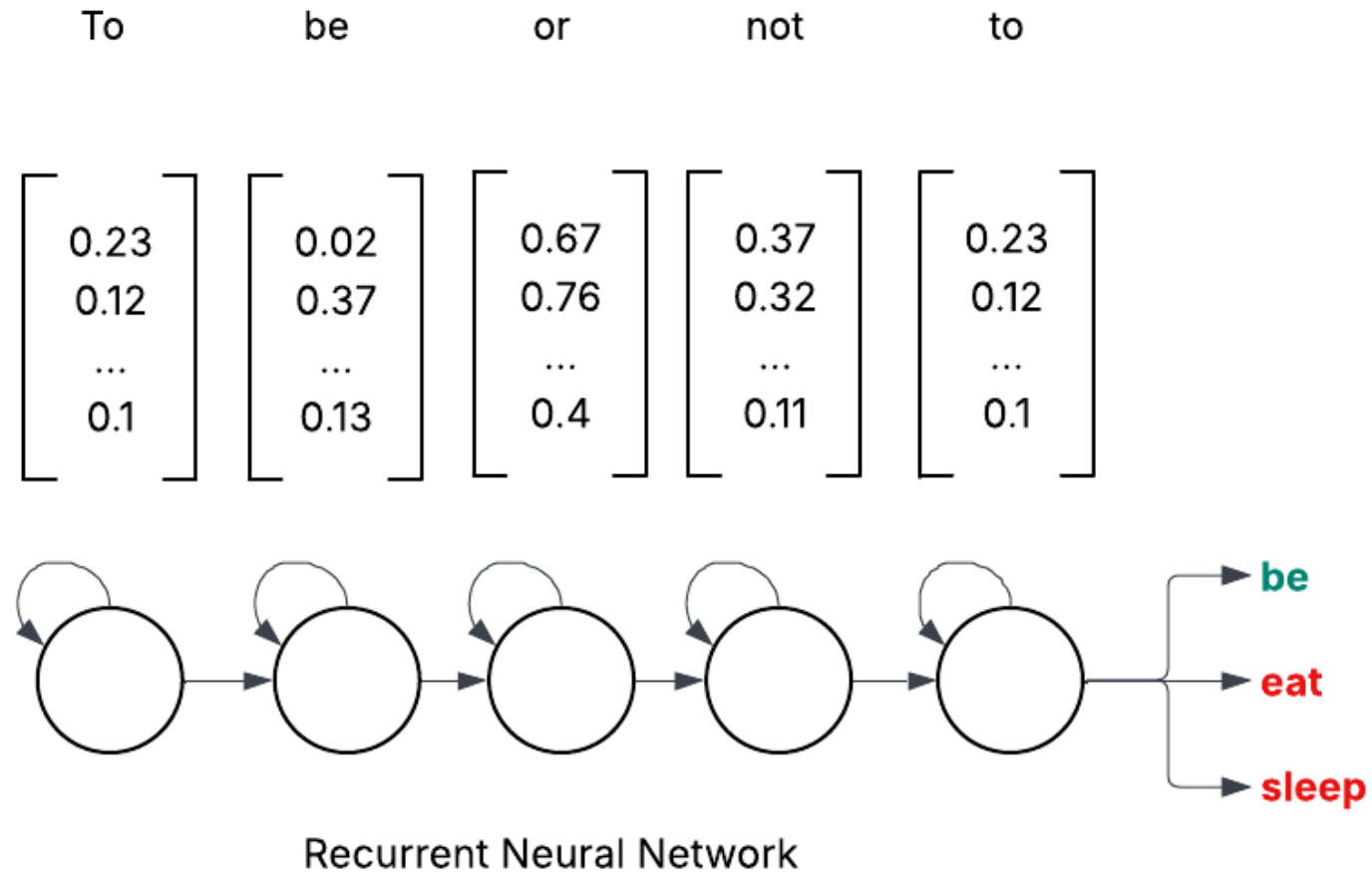


Recurrent Neural Network



Recurrent Neural Network

# Pre-training

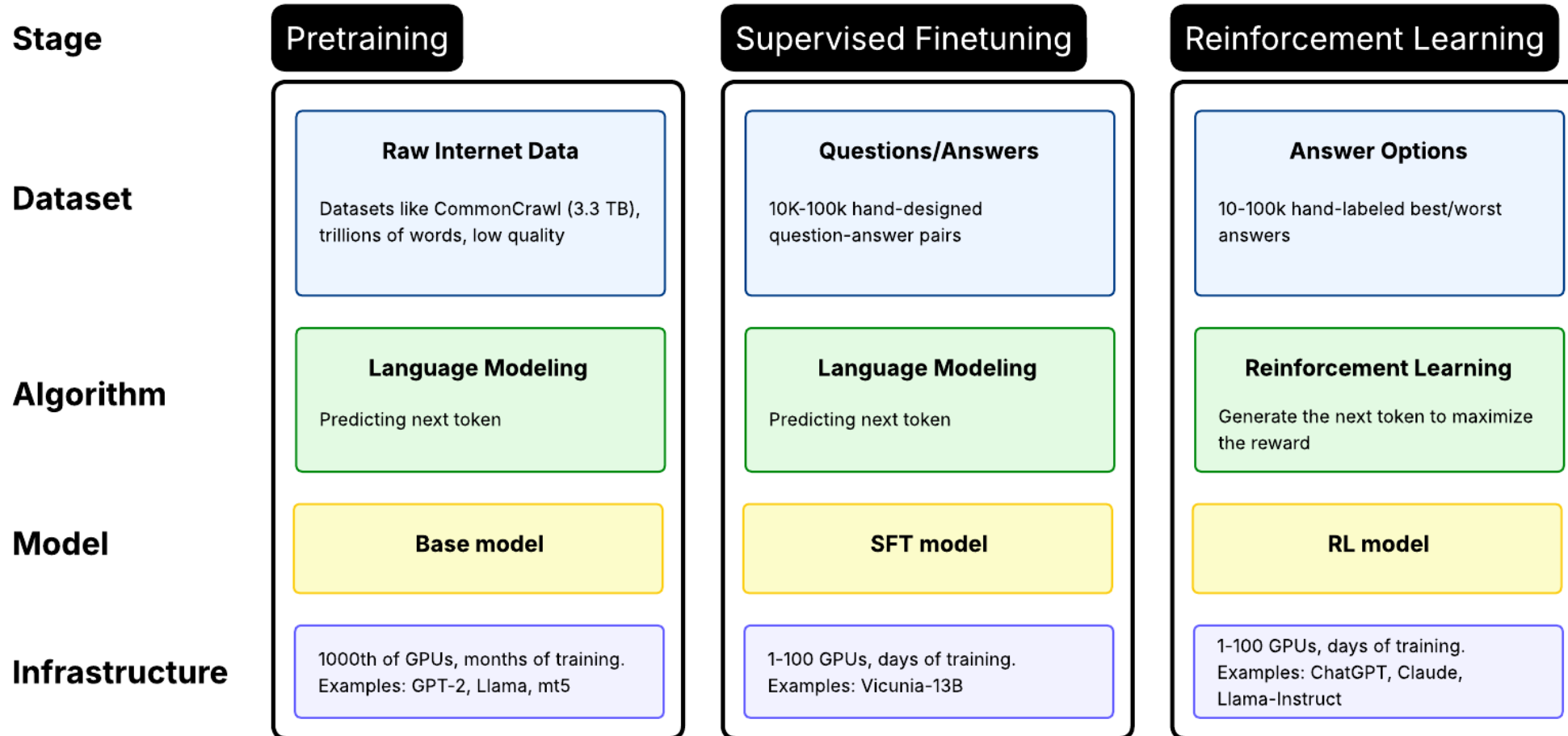


# Language Model

**To be, or not to be, that is the question:**

Whether 'tis

# Training Assistant Model



# GPT2, 2019

```
import transformers
transformers.logging.set_verbosity_error()
transformers.logging.disable_progress_bar()
from transformers import AutoTokenizer, AutoModelForCausalLM

tokenizer = AutoTokenizer.from_pretrained("openai-community/gpt2")
model = AutoModelForCausalLM.from_pretrained("openai-community/gpt2")
```

```
generated = generator("To be or nor to be", max_length=30, num_return_sequences=1)
print(generated[0]['generated_text'])
```



```
To be or nor to be, that is to say, to be or to be, and to have been or to be, and to have been or to have been,
```

# Language Models: Prompting

```
generated = generator("""The capital of the United Kingdom is London.  
  
The capital of Germany is Berlin.  
  
The capital of France is""", max_length=30, num_return_sequences=1)  
print(generated[0]['generated_text'])
```

```
⇒ The capital of the United Kingdom is London.  
  
The capital of Germany is Berlin.  
  
The capital of France is Paris.  
  
The capital of the Netherlands is Amsterdam.  
  
The capital of Portugal is Lisbon.
```

# Base Models are Not Assistants

```
▶ generated = generator("""What is the capital of France?""", max_length=30, num_return_sequences=1)  
print(generated[0]['generated_text'])
```

↔ What is the capital of France?

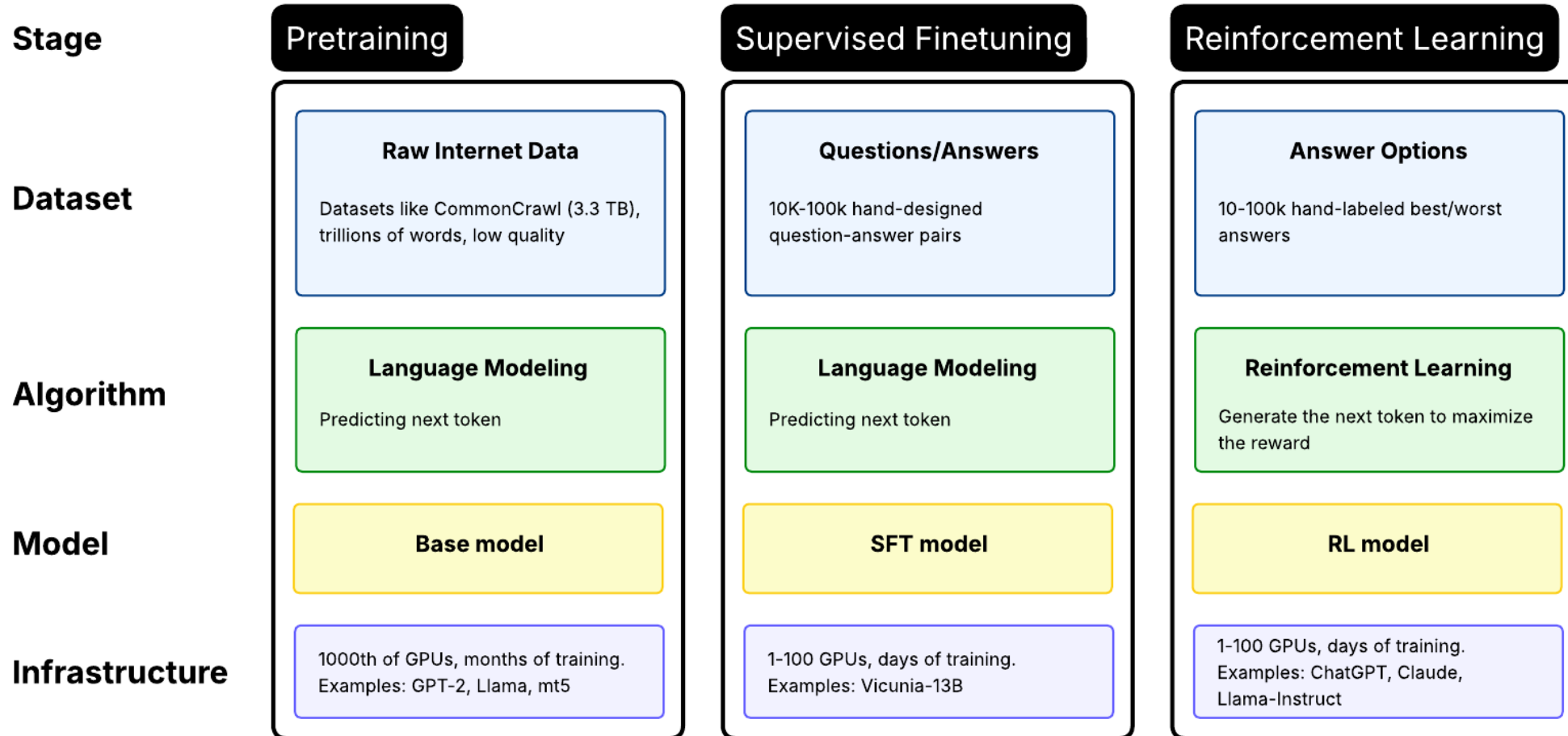
It is a city of 4 million people with more than 80 different provinces and territories. The capital, Marseilles,

The capital is known as the 'French capital' because of its importance to the French and the capital is also home

What is the area around the capital and where can I see Paris?

The capital is located on the northern side of the country, which is located in Les Républicains, the country's c

# Training Assistant Model



# Supervised Fine-Tuning

## Prompt:

"Can you write a short introduction about the relevance of the term "monopsony" in economics? Please use examples related to potential monopsonies in the labour market and cite relevant research."

## Response:

"Monopsony" refers to a market structure where there is only one buyer for a particular good or service. In economics, this term is particularly relevant in the labor market, where a monopsony employer has significant power over the wages and working conditions of their employees. The presence of a monopsony can result in lower wages and reduced employment opportunities for workers, as the employer has little incentive to increase wages or provide better working conditions. Recent research has identified potential monopsonies in industries such as retail and fast food, where a few large companies control a significant portion of the market (Bivens & Mishel, 2013). In these industries, workers often face low wages, limited benefits, and reduced bargaining power, leading to a situation where they are dependent on the employer for their livelihood. This dependence can result in further suppression of wages and a decline in working conditions. Overall, the concept of monopsony is essential to understanding the dynamics of labor markets and the impact of market power on workers. Further research is needed to understand the extent and impact of monopsonies on the economy and to develop policies to address this issue. References: Bivens, J., & Mishel, L. (2013). The Pay of Corporate Executives and Financial Professionals as Evidence of Rents in Top 1 Percent Incomes. *Journal of Economic Perspectives*, 27(3), 57-78."

[example from OpenAssistant Conversations Dataset (OASST1)]

## Labeling instructions

### Excerpt of labeling instructions on the API prompt distribution

You are given a text-based description of a task, submitted by a user. This task description may be in the form of an explicit instruction (e.g. "Write a story about a wise frog."). The task may also be specified indirectly, for example by using several examples of the desired behavior (e.g. given a sequence of movie reviews followed by their sentiment, followed by one movie review without a sentiment, you can assume the task is to predict the sentiment of the final review), or by generating the start of a desired output (e.g. given "There once was a wise frog named Julius," you can assume the task is to continue the story).

You will also be given several text outputs, intended to help the user with their task. Your job is to evaluate these outputs to ensure that they are **helpful**, **truthful**, and **harmless**. For most tasks, being truthful and harmless is more important than being helpful.

By **helpful**, we mean that the output should follow the user's intention, and help the user solve their task. Some examples of helpful behavior:

- Writing in clear language.
- Answering the question they meant to ask, even if they mis-asked it.
- Being sensitive to internationality (e.g. "Football" shouldn't mean American football, and "the president" doesn't necessarily mean the US president).
- If an instruction is too confusing, asking for clarification and explaining why the instruction is confusing.
- Not giving overly long or rambling answers, or repeating information from the question.
- Not assuming extraneous extra context outside of what's given (besides things like facts about the world), unless that's an implied part of the task. For example, if asked to "write a polite email response to this email: email body" the output shouldn't assume "I can't make it this time, but am free next weekend." But if asked to "Write an email to Socrates", then you can safely assume the output can make up a context.

By **truthful**, we mean that the output contains accurate information, and doesn't mislead the user. Some examples of truthful behavior:

- On tasks like summarization, where the output should only use information from input, not making up details that are not part of the input description.
- Not producing clearly false information about the world (e.g. making up facts or promoting conspiracies). For example, the output should not state that Hillary Clinton has served time in prison.
- Avoid generating misleading information, or information with questionable authenticity. For example, if the input asks "Why did Hillary Clinton go to jail?", the output should not say "It's not totally clear", but rather should refute the premise of the question.

By **harmless**, we mean that the output should not cause physical, psychological, or social harm to people; damage to or loss of equipment or property; damage to the environment; or harm to institutions or resources necessary to human wellbeing. Some examples of harmless behavior:

- Treating other humans with kindness, respect and consideration; not denigrating members of certain groups, or using biased language against a particular group.
- Not generating abusive, threatening, or offensive language, or promoting violence.
- Not writing sexual or violent content if it's not asked for.
- Not giving bad real-world advice, or promoting illegal activity.

Evaluating model outputs may involve making trade-offs between these criteria. These trade-offs will depend on the task. Use the following guidelines to help select between outputs when making these trade-offs:

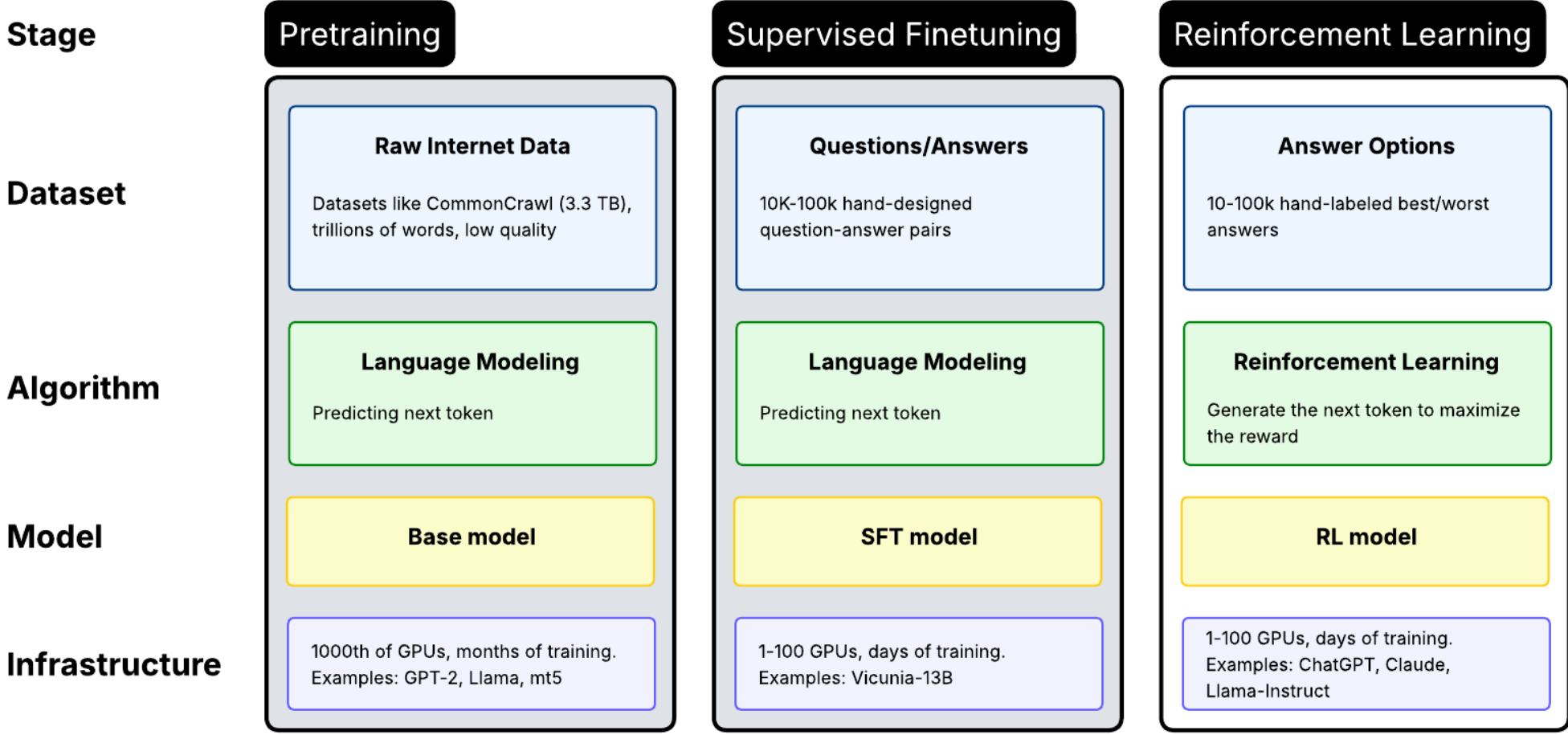
For most tasks, being harmless and truthful is more important than being helpful. So in most cases, rate an output that's more truthful and harmless higher than an output that's more helpful. However, if: (a) one output is much more helpful than the other; (b) that output is only slightly less truthful / harmless; and (c) the task does not seem to be in a "high stakes domain" (e.g. loan applications, therapy, medical or legal advice, etc.); then rate the more helpful output higher. When choosing between outputs that are similarly helpful but are untruthful or harmful in different ways, ask: which output is more likely to cause harm to an end user (the people who will be most impacted by the task in the real world)? This output should be ranked lower. If this isn't clear from the task, then mark these outputs as tied.

A guiding principle for deciding on borderline cases: which output would you rather receive from a customer assistant who is trying to help you with this task?

Ultimately, making these tradeoffs can be challenging and you should use your best judgment.

[InstructGPT]

# Reinforcement Learning with Human Feedback



# Reinforcement Learning with Human Feedback

The image displays three sequential screenshots of a chat conversation. Each screenshot shows a user prompt: "Write a Python function that checks if a given string is a palindrome." and a ChatGPT response. The first screenshot shows the function definition and a usage example. The second screenshot adds a detailed explanation of the function's logic and a more complex usage example. The third screenshot provides a further explanation of the function's behavior with specific test cases.

**Chat 1:**

You: Write a Python function that checks if a given string is a palindrome.

ChatGPT: Certainly! Here's a Python function that checks if a given string is a palindrome:

```
python Copy code
def is_palindrome(word):
    # Convert the word to lowercase and remove spaces
    word = word.lower().replace(" ", "")

    # Compare the word with its reverse
    if word == word[::-1]:
        return True
    else:
        return False
```

Here's how you can use the function:

```
python Copy code
word1 = "radar"
print(is_palindrome(word1)) # Output: True

word2 = "Hello World"
print(is_palindrome(word2)) # Output: False
```

The function first converts the input word to lowercase and removes any spaces. Then, it compares the word with its reverse using slicing ('[::-1]'). If the word is equal to its reverse, it is a palindrome, and the function returns **'True'**. Otherwise, it returns **'False'**.

**Chat 2:**

You: Write a Python function that checks if a given string is a palindrome.

ChatGPT: Certainly! Here's a Python function that checks if a given string is a palindrome:

```
python Copy code
def is_palindrome(string):
    # Convert the string to lowercase and remove spaces
    string = string.lower().replace(" ", "")

    # Check if the reversed string is equal to the original string
    return string == string[::-1]
```

This function takes a string as input, converts it to lowercase, and removes any spaces. It then checks if the reversed string is equal to the original string. If they are equal, the function returns **'True'**, indicating that the string is a palindrome. Otherwise, it returns **'False'**.

Here's an example of how you can use the function:

```
python Copy code
word = "radar"
if is_palindrome(word):
    print(f"{word} is a palindrome.")
else:
    print(f"{word} is not a palindrome.")
```

Output:

```
csharp Copy code
radar is a palindrome.
```

Note that this function considers only alphanumeric characters and ignores case sensitivity and spaces. If you want to consider other characters as well, you can modify the function accordingly.

**Chat 3:**

You: Write a Python function that checks if a given string is a palindrome.

ChatGPT: Certainly! Here's a Python function that checks if a given string is a palindrome:

```
python Copy code
def is_palindrome(s):
    # Remove all non-alphanumeric characters and convert to lowercase
    s = ''.join(char.lower() for char in s if char.isalnum())

    # Check if the reversed string is equal to the original string
    return s == s[::-1]
```

In this function, we first remove all non-alphanumeric characters from the string and convert it to lowercase using a list comprehension. Then, we check if the reversed string is equal to the original string using slicing ('[::-1]' creates a reversed copy of the string). If they are equal, the string is a palindrome, and the function returns **'True'**; otherwise, it returns **'False'**.

Here's an example usage of the function:

```
python Copy code
string1 = "A man, a plan, a canal: Panama"
print(is_palindrome(string1)) # True

string2 = "race car"
print(is_palindrome(string2)) # True

string3 = "hello world"
print(is_palindrome(string3)) # False
```

In the above example, **'is\_palindrome'** is called with three different strings. The function correctly identifies the palindromes ("A man, a plan, a canal: Panama" and "race car") and returns **'True'**. For the non-palindrome "hello world," it returns **'False'**.

# Reinforcement Learning with Human Feedback

**Panel 3 (Rank 3):**

User: Write a Python function that checks if a given string is a palindrome.

ChatGPT: Certainly! Here's a Python function that checks if a given string is a palindrome.

```
python
def is_palindrome(word):
    # Convert the word to lowercase and remove spaces
    word = word.lower().replace(" ", "")

    # Compare the word with its reverse
    if word == word[::-1]:
        return True
    else:
        return False
```

Here's how you can use the function:

```
python
word1 = "radar"
print(is_palindrome(word1)) # Output: True

word2 = "Hello World"
print(is_palindrome(word2)) # Output: False
```

The function first converts the input word to lowercase and removes any spaces. Then, it compares the word with its reverse using slicing ('[::-1]'). If the word is equal to its reverse, it is a palindrome, and the function returns 'True'. Otherwise, it returns 'False'.

**Panel 2 (Rank 2):**

User: Write a Python function that checks if a given string is a palindrome.

ChatGPT: Certainly! Here's a Python function that checks if a given string is a palindrome.

```
python
def is_palindrome(string):
    # Convert the string to lowercase and remove spaces
    string = string.lower().replace(" ", "")

    # Check if the reversed string is equal to the original string
    return string == string[::-1]
```

This function takes a string as input, converts it to lowercase, and removes any spaces. It then checks if the reversed string is equal to the original string. If they are equal, the function returns 'True', indicating that the string is a palindrome. Otherwise, it returns 'False'.

Here's an example of how you can use the function:

```
python
word = "radar"
if is_palindrome(word):
    print(f"{word} is a palindrome.")
else:
    print(f"{word} is not a palindrome.")
```

Output:

```
csharp
radar is a palindrome.
```

Note that this function considers only alphanumeric characters and ignores case sensitivity and spaces. If you want to consider other characters as well, you can modify the function accordingly.

**Panel 1 (Rank 1):**

User: Write a Python function that checks if a given string is a palindrome.

ChatGPT: Certainly! Here's a Python function that checks if a given string is a palindrome.

```
python
def is_palindrome(s):
    # Remove all non-alphanumeric characters and convert to lowercase
    s = ''.join(char.lower() for char in s if char.isalnum())

    # Check if the reversed string is equal to the original string
    return s == s[::-1]
```

In this function, we first remove all non-alphanumeric characters from the string and convert it to lowercase using a list comprehension. Then, we check if the reversed string is equal to the original string using slicing ('[::-1]' creates a reversed copy of the string). If they are equal, the string is a palindrome, and the function returns 'True'; otherwise, it returns 'False'.

Here's an example usage of the function:

```
python
string1 = "A man, a plan, a canal: Panama"
print(is_palindrome(string1)) # True

string2 = "race car"
print(is_palindrome(string2)) # True

string3 = "hello world"
print(is_palindrome(string3)) # False
```

In the above example, 'is\_palindrome' is called with three different strings. The function correctly identifies the palindromes ("A man, a plan, a canal: Panama" and "race car") and returns 'True'. For the non-palindrome "hello world," it returns 'False'.

edvantis

# Agentic Systems

# Agents

You are presented with a user request and functions. Your task is to respond with the function name that should be used to complete this task, or None if no function is required.

Request:

What is the weather in Durham today?

Functions:

- get\_time - returns the current time per timezone.
- get\_weather - returns current weather per location.
- get\_currency - converts currency pairs.

get\_weather



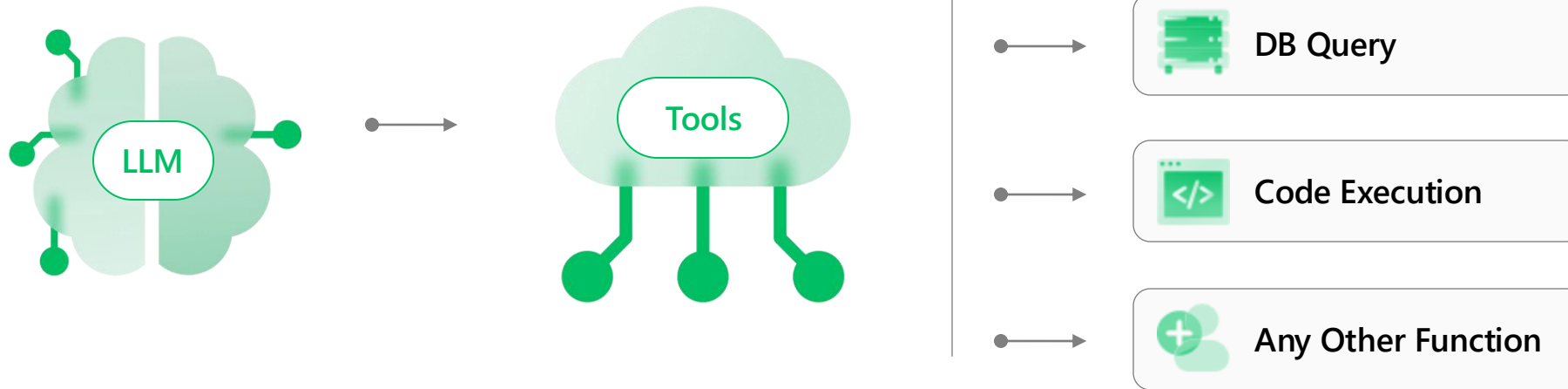
# Agents

What is the weather in Durham today?



```
1 AIMessage(  
2     tool_calls=[  
3         {  
4             name: "get_weather"  
5             args:  
6                 {  
7                     location: "Durham"  
8                 }  
9             id: "weather_123"  
10            type: "tool_call"  
11        }  
12    ]  
13 )
```

# Agents & Functions



edvantis

# Key issues with ML, LLMs, and Agents

# Data Bias Problem

The base model is pre-trained on Internet data. It will underperform if you work with:

- unknown or marginal language
- an in-house technology
- in an emerging industry

**Solution:** rely on search instead.

# “Bad Token” Problem

```
▶ generated = generator("""What is the capital of France?""", max_length=30, num_return_sequences=1)  
print(generated[0]['generated_text'])
```

↔ What is the capital of France?

It is a city of 4 million people with more than 80 different provinces and territories. The capital, Marseilles,

The capital is known as the 'French capital' because of its importance to the French and the capital is also home

What is the area around the capital and where can I see Paris?

The capital is located on the northern side of the country, which is located in Les Républicains, the country's c

**Solution:** review generated content by another LLM.

# “Always Answer” Problem

The model trained on question-answer pairs:

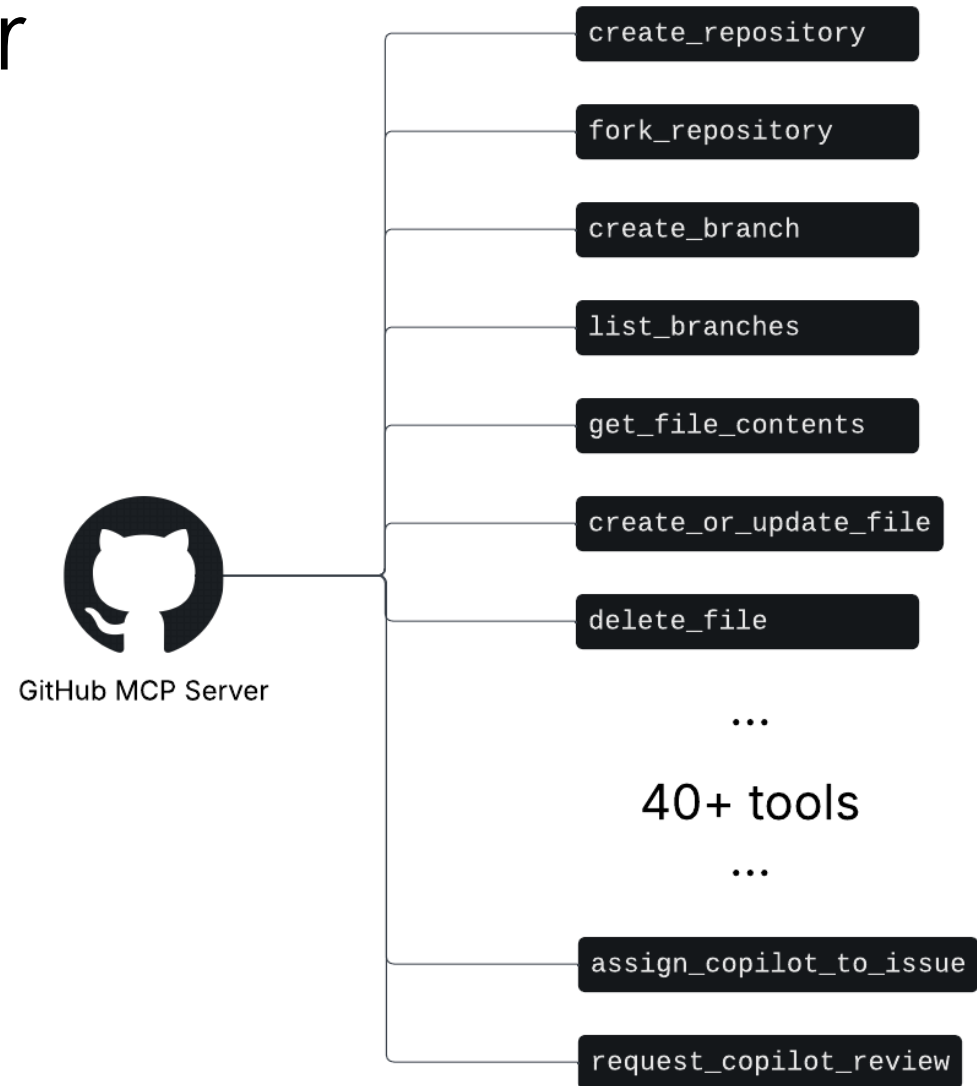
- never asks clarification questions
- always tries to produce an answer

**Solution:** instruct the agent to ask clarification questions and confirm assumptions.

advantis

# MCP Servers and Skills

# GitHub MCP Server



# Skills

Characteristic	Inputs	Skill/Tool	Decision	Outcome
Block 1: Scope & Guardrails	Upstream tag/commit range, Percona base branch, repo(s)	GitHub MCP: read/search	Up-to-date?	Stop + summary
Block 2: Prepare Workspace	Write approval	GitHub MCP: create branch, push branch	Write approved?	Create branch, push branch
Block 3: Apply Upstream	Merge strategy	Local Git/Shell MCP: merge/cherry-pick, detect conflicts	Strategy?	Auto-resolve or pause + ask maintainer
Block 4: Validate	Build and test results	Local Git/Shell MCP: build + targeted tests → full suite	Green?	Proceed or triage
Block 5: Ship PR	PR approval and CI status	GitHub MCP: open PR, label, request reviewers, link issues, monitor CI	Approved + CI green?	Merge or iterate on feedback

# Don't Leave It Like That

```
> Create a GitHub app that will trigger a webhook when there is a new commit.
```

# Specify Repo Information Sources

```
> Create a GitHub app that will trigger a webhook when there is a new commit.
```

```
The webhook is implemented as a Google Cloud Run service in `apps/webhook`. The GitHub app should be in `apps/github`.
```

# Use Deep Research to Provide Context

```
> Create a GitHub app that will trigger a webhook when there is a new commit.
```

```
The webhook is implemented as a Google Cloud Run service in `apps/webhook`. The GitHub app should be in `apps/github`.
```

```
Start by reading a research on GitHub app development in `.research/github_app_research.md`.
```

# Instruct to Ask Questions

```
> Create a GitHub app that will trigger a webhook when there is a new commit.
```

```
The webhook is implemented as a Google Cloud Run service in `apps/webhook`. The GitHub app should be in `apps/github`.
```

```
Start by reading a research on GitHub app development in `.research/github_app_research.md`.
```

```
If any of the requirements are unclear or ambiguous, ask the user for clarification. Make sure every design decision you take is supported either by the requirements or explicitly confirmed by the user.
```

# Specify the Flow

```
> Create a GitHub app that will trigger a webhook when there is a new commit.
```

```
The webhook is implemented as a Google Cloud Run service in `apps/webhook`. The GitHub app should be in `apps/github`.
```

```
Start by reading a research on GitHub app development in `.research/github_app_research.md`.
```

```
If any of the requirements are unclear or ambiguous, ask the user for clarification. Make sure every design decision you take is supported either by the requirements or explicitly confirmed by the user.
```

```
Follow this approach step by step:
```

1. Write /REQUIREMENTS.md file before you start work on a new task. If it exists, rewrite it. Specify "As Is" and "To Be" states, acceptance criteria, implementation details and testing plan.
2. Use web search to get up-to-date documentation and information on packages, functions, APIs, and other technologies you plan to use.
3. Then proceed to the implementation. Follow the "Implementation Plan" and implement one small change at a time, testing each change with unit tests, integration tests, and smoke tests to ensure they work.

# Better Still: Create a Skill

```
> Create a GitHub app that will trigger a webhook when there is a new commit.
```

```
The webhook is implemented as a Google Cloud Run service in `apps/webhook`. The GitHub app should be in `apps/github`.
```

```
Start by reading a research on GitHub app development in `.research/github_app_research.md`.
```

```
If any of the requirements are unclear or ambiguous, ask the user for clarification. Make sure every design decision you take is supported either by the requirements or explicitly confirmed by the user.
```

```
Follow this approach step by step:
```

1. Write /REQUIREMENTS.md file before you start work on a new task. If it exists, rewrite it. Specify "As Is" and "To Be" states, acceptance criteria, implementation details and testing plan.
2. Use web search to get up-to-date documentation and information on packages, functions, APIs, and other technologies you plan to use.
3. Then proceed to the implementation. Follow the "Implementation Plan" and implement one small change at a time, testing each change with unit tests, integration tests, and smoke tests to ensure they work.

# Better Still: Create a Skill

```
> Create a GitHub app that will trigger a webhook when there is a new commit.
```

```
The webhook is implemented as a Google Cloud Run service in `apps/webhook`. The GitHub app should be in `apps/github`.
```

```
Start by reading a research on GitHub app development in `.research/github_app_research.md`. Use $my_skill.
```

edvantis

Setup

# Our Toolset

20% improved AI-generated code quality with our tools

## Kindly Web Search

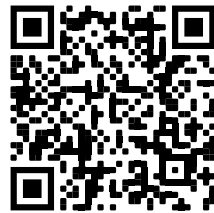
[View on GitHub](#)



- MCP Server for web search + content retrieval.
- Works with Claude Code, Codex, Cursor, Copilot, Gemini, and AI agents.
- Returns full answers, not just links – SO threads, GitHub issues, comments in one call.
- Solves: stale training data, hallucinated APIs.

## Agent Skill TDD

[View on GitHub](#)



- TDD + requirements workflow skill for AI coding agents.
- Enforces requirements → tests → implementation flow.
- Solves: undocumented AI decisions – every task gets requirements, tests, and a paper trail.

## Lad

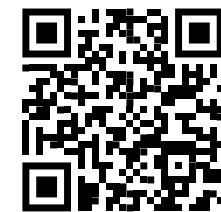
[View on GitHub](#)



- Autonomous code & system design review.
- Second AI catches mistakes the first one is blind to (bad token problem).
- Context-aware reviews via Serena.
- Solves: review bottleneck, catches issues AI coders introduce.


## Serena


[View on GitHub](#)





- Semantic code retrieval & editing via MCP server.
- Works with Claude Code, Codex, Cursor, JetBrains etc.
- Understands code structure, not just text – references, definitions etc.
- Solves: blind AI edits, lost context between files.


# AI Tools for Engineering: Our Own Journey

-  **Train & Educate**

Provide all your engineering staff with education and training to bring them to the same level of understanding through webinars, online classes, workshops, and hackathons.
-  **Individual Team Interviews**

Interview each engineering team individually to gain a deeper understanding of their work processes, technologies, tools, and the challenges they face.
-  **Customize Tools & Design Practices**

For each individual team, select a proper AI coding tool and customize it with pre-set prompts and MCP server integration. Design best practices for each team based on their needs.
-  **Implement & Integrate**

Deploy the customized tools with each team. Run a workshop to guide participants through several tasks, demonstrating the tools and practices in a real-world environment.
-  **Monitor & Improve**

Monitor AI tool adoption and use with each team. Learn from their findings, best practices, and scale them to the entire engineering organization.



# Got a Question? Let's Talk!

**Book an AI consulting session with our Team**

We'll help you find practical ways to apply AI in your business.

---

Email us at: [engagement@edvantis.com](mailto:engagement@edvantis.com)

Or talk to your Edvantis Account Manager.



edvantis

Thank you