MCP Workshop

From Data to Intelligent Agents

nacorm



Nabil SADEG
CTO, Nacorm

Experience in **software development**, specialized in **high-performance graphics**. Expertise in **artificial intelligence and data processing**.

Nacorm develops and applies advanced AI technologies for AECO, helping organizations move from experimentation to reliable, operational systems.

Training & Awareness
Engagement in the industry through **conferences** and **workshops**

AI & Data Consulting
Strategic consulting and **R&D** with leading AECO partners

Software Development
Custom design of **algorithms and AI-powered tools**

AU 2025 · BIM DAY CVA · BIMWORLD JUMEAUX NUMÉRIQUES · AU 2024 · IHF INGÉNIEURS HOSPITALIERS DE FRANCE · AUTODESK DevCon · TECH FOR CONSTRUCTION

The Foundations of Intelligent Systems

What You Need to Know Before Building AI Agents

nacorm



DATA

Understanding and labelling your data

DATA



AI

Identify AI applications



DATA



AI



API

Connect systems, expose capabilities



DATA

AI

API



MCP

Standardize AI-to-tool communication



DATA



AI



API



MCP



DATA

Understanding and labelling your data

DATA

Why data is the foundation of AI

- AI is entirely dependent on **Data**
- **Data quality = AI Quality**
- AEC data often **incomplete, heterogeneous**



DATA

6 main data types



Numeric



Text



Audio



Visual (2D)



Video



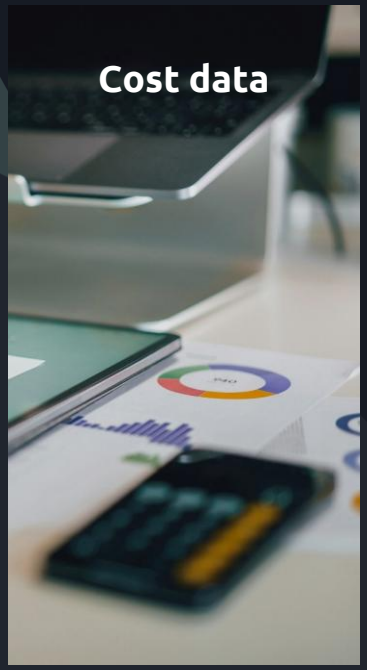
Geometric (3D)

nacorm

DATA

Numeric data

Information expressed as numbers for measurement or calculation.







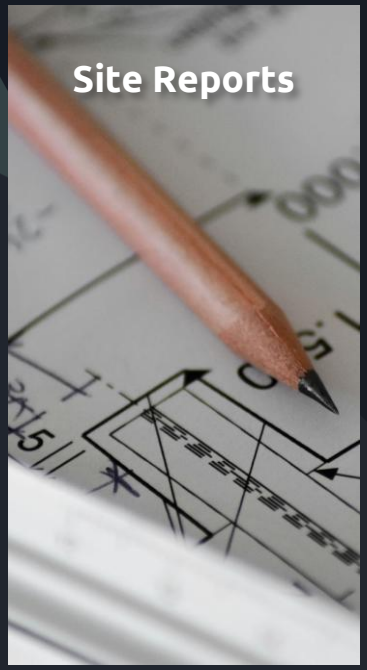


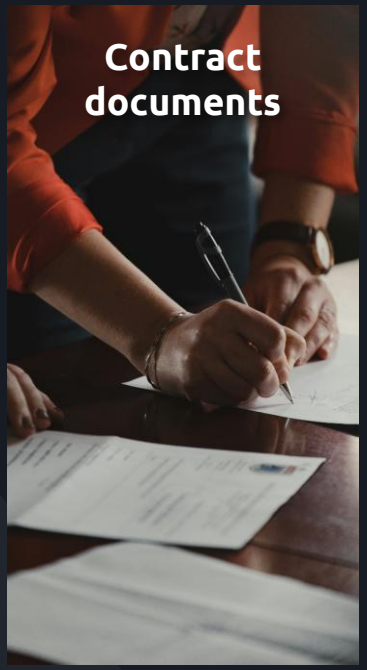
nacorm

DATA

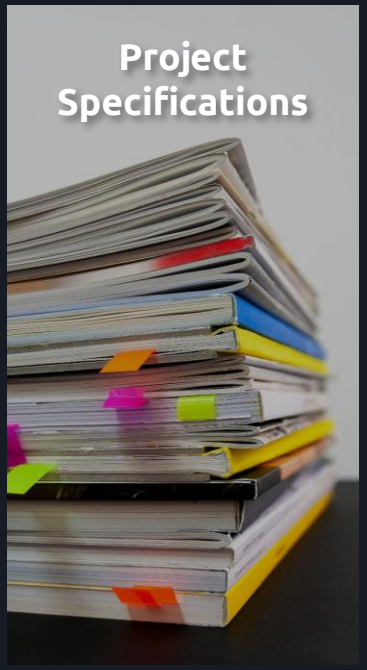
Text data

Information conveyed by sequences of characters or words.









nacorm

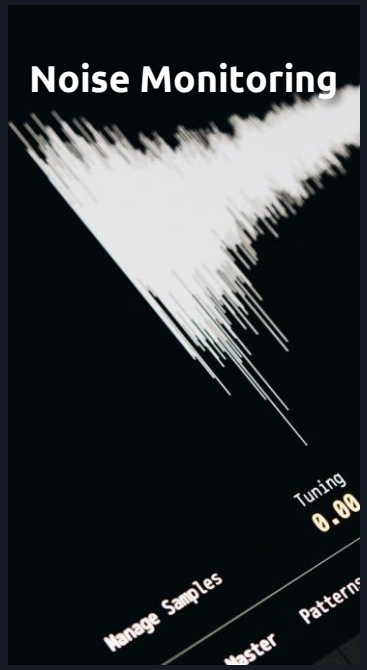
DATA

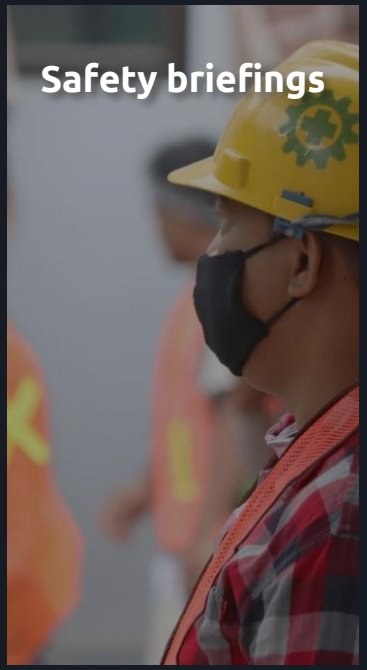
Audio data

Information captured as variations in sound over time.







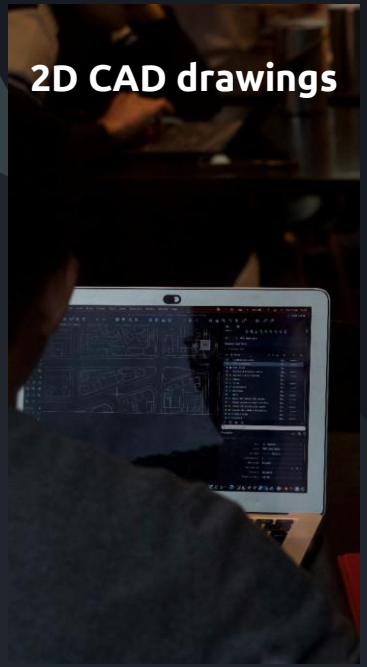


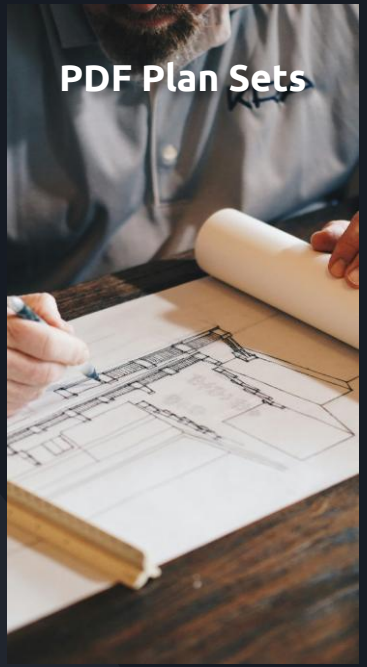
nacorm

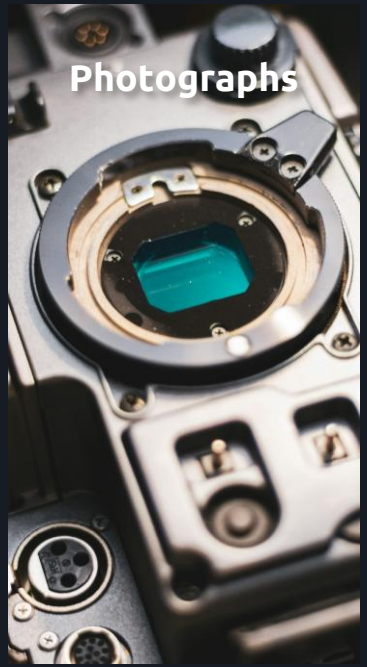
DATA

Visual data (2D)

Information represented as a flat grid of pixels.









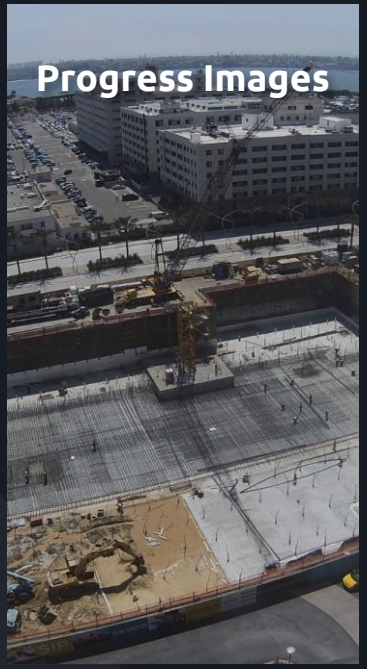
nacorm

DATA

Video data

Information represented by a sequence of images played over time to represent motion.









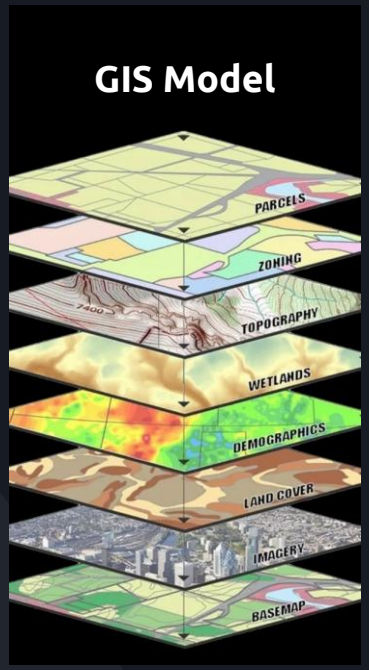
nacorm

DATA

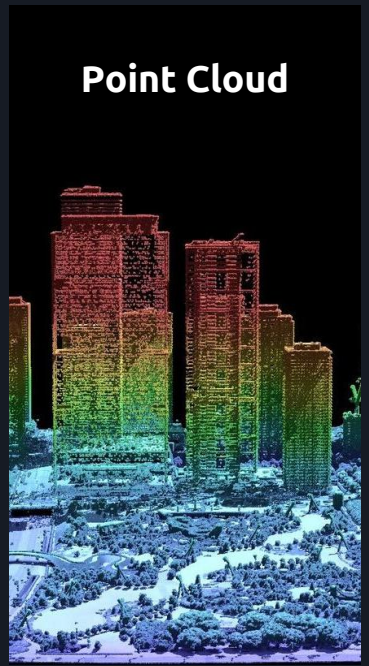
Geometric data (3D)

Information defined by points or shapes in three-dimensional space.









nacorm



DATA

4 Data Quality Attributes

- Accuracy
- Consistency
- Completeness
- Accessibility

nacorm



DATA

4 Data Quality Attributes

Accuracy

Consistency

Completeness

Accessibility

nacorm



DATA

4 Data Quality Attributes

Accuracy

Consistency

Completeness

Accessibility

nacorm



DATA

4 Data Quality Attributes

Accuracy

Consistency

Completeness

Accessibility

nacorm



DATA

4 Data Quality Attributes

Accuracy

Consistency

Completeness

Accessibility

nacorm

DATA

Accuracy

The data correctly reflects real-world facts or events, free of errors or distortions.



Accurate dataset



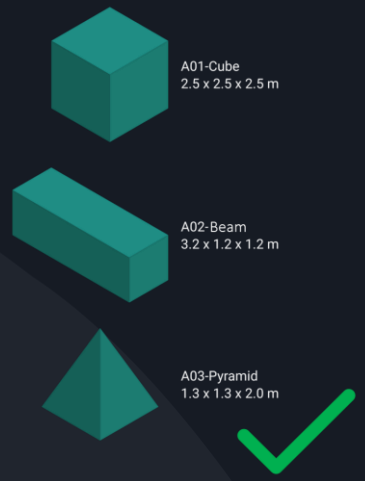
Precise dataset

nacorm

DATA

Consistency

Ensuring data is represented identically across systems in both value and format.



Consistent dataset



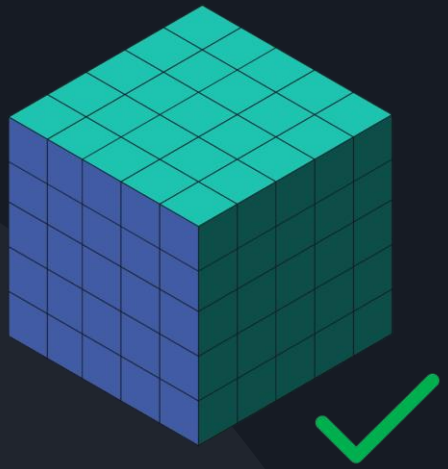
Inconsistent dataset

nacorm

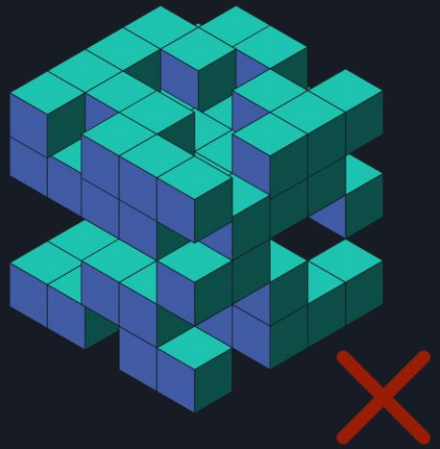
DATA

Completeness

All required data is captured with no necessary parts missing.



Complete dataset



Incomplete dataset

nacorm

DATA

Accessibility

The ease with which authorized users can locate, retrieve and use data when they need it.



Accessible



Inaccessible

nacorm

DATA

Internal vs. external quality

Internal data quality

Evaluate the data contained in **your organization's systems**.

- You have **full control** over internal data management.

External data quality

Evaluate the quality of your **data interface with external systems**.

- Comply with **industry standards** and partner specifications.

nacorm

DATA

Structured vs. Unstructured Data

Structured data

- Organized in a defined format (tables, columns, fields)
- Easy to query and connect across systems

Examples: Excel table, SQL database, sensor measurements, inventories, costs, dates

Unstructured data

- No fixed format, difficult to analyze automatically
- Often rich in context or semantics

Examples: Emails, PDF documents, images, videos, conversations, scanned drawings/plans

Transformation

Unstructured data can be analyzed to extract structured elements:

- Automatic document classification
- Object detection in an image
- Entity recognition (names, places, numbers) in free text

nacorm

DATA

Data Quantity

Risks

- Too little data limits analysis and the reliability of results
- Too much data makes management, storage, and access more complex

Assessing data quantity

Total volume: overall size (GB, TB) and number of files

Number of records: rows, objects, measurements, images, documents

Frequency: collection or sampling rate over time

Granularity: level of detail of each record

Coverage: range of cases or situations represented

Reuse rate: share of data that is actually usable

A database with 1 million records can be less useful than a sample of 100,000 that covers all observed situations.
Data quantity should always be evaluated against its coverage and intended use.

nacorm

DATA

Noise in data

Noise refers to **erroneous, inconsistent,** or **random values** that do not represent the observed reality.It hides real trends and reduces the reliability of analyses.

Possible sources

- Data entry or measurement errors
- Miscalibrated or faulty sensors
- Incomplete or poorly synchronized data
- Incorrect conversion or transmission
- External disturbances (temperature, movement, interference)

Noise reduction

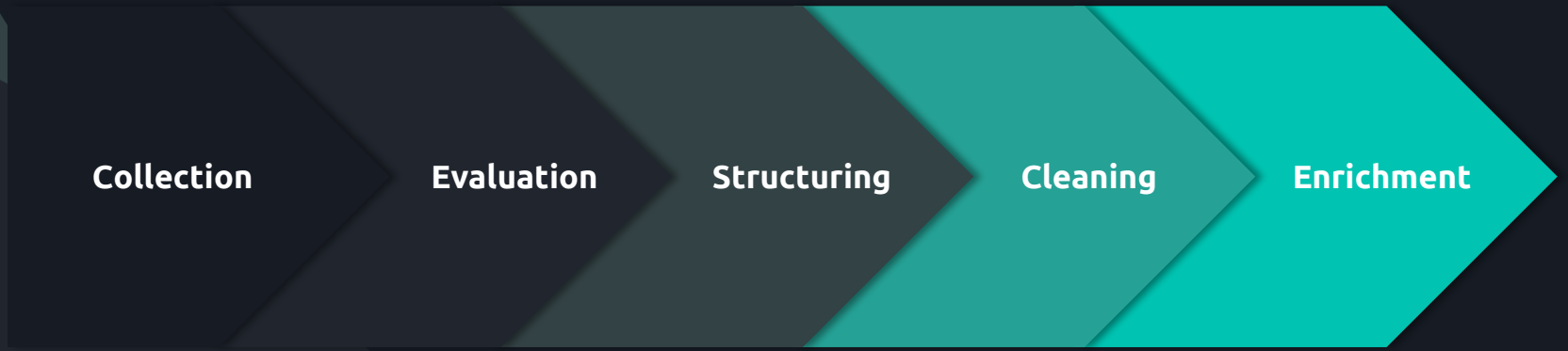
- Regular verification and calibration of sources
- Detection and handling of outliers
- Averaging or filtering depending on the context
- Using domain knowledge

Noise never disappears completely, but it can be greatly reduced by combining technical tools and domain expertise.

nacorm

DATA

AEC Data Pipeline



nacorm



DATA



AI



API



MCP

DATA



AI

Identify AI applications

Algorithm vs. AI

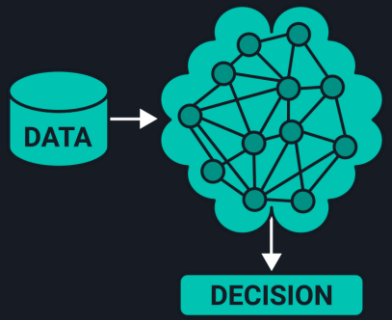
Algorithm

- An algorithm follows a **fixed series of steps** to solve a task.
- It always behaves in the same way when it is provided with the same inputs.



Artificial intelligence

- Artificial intelligence **uses data** to improve over time.
- AI can **adapt** to new situations and make **decisions** without explicit instructions.



AI

Human Abilities Framework



See



Hear and Speak



Understand Language



Act



Learn



Find



Reason

nacorm

Human Abilities Framework



See



Hear and Speak



Understand Language



Act



Learn



Find



Reason

AI

See



Computer Vision

Definition

Interpret images and 3D environments to extract useful information and detect patterns.

Input Data



Goal

Information extraction, pattern & object recognition

Use Cases

Automated Quality Control

Computer vision analyzes 3D BIM models and site photos to detect deviations and ensure compliance.

Object Recognition and Classification

AI identifies and classifies construction elements (doors, windows, pipes, etc.) in a BIM model or site scan.

nacorm x KLAI

© 2026 Nacorm x KLAI. All rights reserved.

nacorm

AI

Hear & Speak



Speech To Text & Text To Speech

Definition

Enable voice synthesis and transcription, converting text to speech and vice versa.

Input Data



Goal

Transcription & speech synthesis

Use Cases

Automated BIM Model Documentation

STT transcribes spoken notes from inspectors and architects into BIM data, reducing manual input.

Virtual BIM Assistants

TTS and STT enable smart assistants to answer questions about BIM, materials and compliance.

nacorm x KLAI

© 2026 Nacorm x KLAI. All rights reserved.

nacorm

AI

Understand Language

Natural Language Processing

Definition

Enables interpretation, analysis and extraction of information from text data.

Input Data



Goal

Extract meaning, behavior & emotion

Use Cases

Automated Compliance Checking

NLP analyzes regulations and compares them to BIM models to flag non-compliance.

Issue Tracking & Reporting

The model processes incident reports (emails, chats, field notes) and links them to BIM elements for faster resolution.

nacorm x KLAI

© 2026 Nacorm x KLAI. All rights reserved.

nacorm

AI

Act



Robotics & Automation

Definition

Automate repetitive or complex tasks using intelligent systems and robots.

Input Data



Goal

Perform physical & virtual tasks, make decisions

Use Cases

Automated Prefabrication & Assembly

Robots use BIM data to cut, assemble and transport components, reducing waste and boosting efficiency.

AI-Powered Site Inspections

Drones and robots navigate sites, compare progress to BIM models and detect real-time discrepancies.

nacorm x KLAI

© 2026 Nacorm x KLAI. All rights reserved.

nacorm

A
I

Learn



Machine Learning

Definition

Use historical data to identify patterns and generate predictive models.

Input Data



Goal

Pattern recognition, outcome prediction & rule discovery

Use Cases

Construction Risk Assessment

AI analyzes historical data and BIM models to predict delays, cost overruns and safety risks.

AI-Based Cost Estimation

Machine learning generates accurate cost estimates from BIM models, reducing budget uncertainty.

nacorm x KLAI

© 2026 Nacorm x KLAI. All rights reserved.

nacorm

A
I

Find

Retrieval-Augmented Generation

Definition

Combine information retrieval and content generation to deliver accurate, contextual responses.

Input Data

Goal

Quick retrieval of relevant information

Use Cases

Automated Specification Matching

The model compares manufacturer databases with BIM requirements to recommend optimal materials.

Smart Issue Resolution

AI analyzes past issues and suggests contextual solutions within the BIM environment.

nacorm x KLAI

© 2026 Nacorm x KLAI. All rights reserved.

nacorm

A
I

Reason



Logical Inference

Definition

Applies logical rules to make informed decisions based on available data.

Input Data



Goal

Make informed decisions based on available data

Use Cases

Design Conflict Resolution

AI analyzes conflicting requirements (e.g., structure vs. energy efficiency) and suggests optimal solutions.

Layout Optimization

The model recommends optimal spatial layouts based on functional and regulatory requirements.

nacorm x KLAI

© 2026 Nacorm x KLAI. All rights reserved.

nacorm



DATA



AI



API



MCP



DATA



AI



API

Connect systems, expose capabilities

API

What is an API?

An API (Application Programming Interface) is a **contract** between two software systems. It defines how one system can **request data** or **trigger actions** in another.

Key characteristics:

- Defined **endpoints** (URLs) for specific operations
- Authentication via **API keys** or tokens
- Structured **request/response** format (usually JSON)
- Documented and **versioned**

nacorm

API

How an API works

The request-response cycle:



1. The **client** sends a request to a specific endpoint (e.g., *GET /projects/123*)



2. The **server** processes the request, accesses its database or logic

3. The server returns a **structured response** (status code + data)

Core elements:

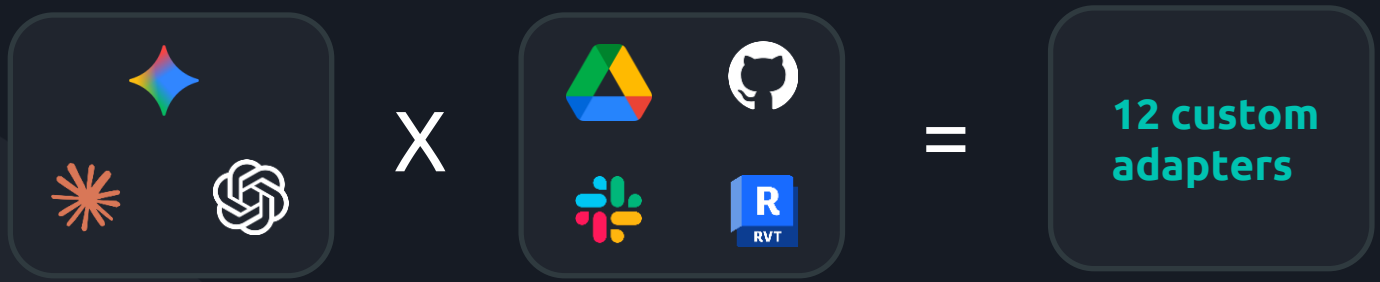
- **Endpoint:** The URL that identifies the resource
- **Method:** The action to perform (GET, POST, PUT, DELETE)
- **Headers:** Metadata (authentication, content type)
- **Body:** The payload (for POST/PUT requests)

nacorm

Weakness - The N×M problem

Without a standard, connecting **N AI models** to **M tools** requires **N × M custom integrations**.

Example:



- ✗ **Built** from scratch
- ✗ **Maintained** independently
- ✗ **Updated** when either side changes

The result: Integration cost grows **exponentially**, not linearly. Every new model or tool multiplies the work.



A
P
I

Fixed endpoints

Traditional APIs are **static by design:**

- Each capability requires a **dedicated endpoint**
- Adding a feature means **modifying the server** and **updating all clients**
- The client must **know in advance** what the API can do

Consequences for AI:

- The LLM cannot **discover** new capabilities at runtime
- Every tool change requires **rewriting integration code**
- No flexibility to adapt to **unforeseen use cases**

APIs are powerful for **structured,** predictable workflows but they struggle with the **dynamic, open-ended** nature of AI agents.

nacorm

API

API examples

AUTODESK Platform Services

Autodesk Platform Services (APS)

- Access BIM data, manage files, translate models
- RESTful endpoints for **Data Management, Model Derivative, Forma**

OpenAI / Anthropic / Gemini

- Send prompts, receive completions
- Fixed request format: model, messages, parameters

AUTODESK Forma

Forma API

- Retrieve **issues**, **RFIs**, project members
- Automate workflows across the **construction lifecycle**

nacorm



DATA



AI



API



MCP



DATA

AI

API



MCP

Standardize AI-to-tool communication

What is the MCP protocol (Model Context Protocol)?

Definition

MCP is an **open standard** for **communication** between Large Language Models (**LLMs**) and **external systems** such as tools, APIs, databases, and even other LLMs. It is often referred to as “the USB-C of AI.”

The problem it solves

Before MCP, each connection was custom-built. A developer had to create a different proprietary “adapter” for each LLM (Claude, ChatGPT, Gemini, etc.) and each tool (Google Drive, VSCode, GitHub, etc.).

The solution

MCP creates a standard “plug.” Any MCP-compatible LLM can instantly connect to any MCP-compatible tool, **making AI agents reusable and interoperable**.

MCP flexibility: Build once, connect anywhere

MCP solves the N×M problem with a **1-to-many** approach:

- Build **one MCP server** per tool → it works with **any MCP-compatible client**
- Build **one MCP client** per host → it works with **any MCP server**

What this enables:

✓ **Swap LLMs** without rewriting tool integrations

✓ **Add new tools** without modifying the AI agent

✓ **Reuse servers** across projects, teams, and organizations

The same MCP server you build for Autodesk APS today works with Claude, Copilot, or any future MCP-compatible host, **no adapter needed**.

MCP

MCP Server vs Client

MCP Client (the host side)

- Lives inside the **AI application** (IDE, chatbot, desktop agent)
- Manages the **connection** to one or more MCP servers
- Forwards **tool calls** from the LLM to the right server
- Returns **tool responses** back to the LLM

MCP Server (the tool side)

- A lightweight service that **exposes capabilities** to AI
- Registers **tools**, **resources**, and **prompts**
- Handles the **actual execution** (API calls, file reads, database queries)
- Can run **locally** (stdio) or **remotely** (HTTP/SSE)

The server doesn't need to know which LLM is calling it. The client doesn't need to know how the tool works internally. The **protocol** handles the translation.

nacorm

The 3 MCP primitives

Prompts (user-controlled)

- Pre-built **templates** for common workflows
- Example: "Analyze this Revit model for code compliance"
- Triggered by the **user**, not autonomously by the LLM

Resources (application-controlled)

- Data the server makes **available** for context
- Example: project files, BIM model metadata, user preferences
- Similar to GET endpoints, **read-only** context

Tools (model-controlled)

- Functions the LLM can **call** to perform actions
- Example: *create_issue, get_model_properties, run_clash_detection*
- The LLM decides **when** and **how** to use them

MCP

Context is everything

LLMs are powerful, but they operate in a **limited context window**. MCP helps manage what goes in:

Why context matters:

- ✗ An LLM with **no project context** gives generic answers
- ✓ An LLM with **relevant project context** gives specific, useful answers

How MCP delivers context:

- **Prompts** frame the task with the right instructions
- **Resources** inject structured data into the conversation
- **Tool results** provide real-time information from live systems

nacorm

Validation and Security

MCP introduces **structured control** over what AI can do:



Input validation

- Every tool defines a **schema** (using Zod/JSON Schema)
- Parameters are **validated before execution**, malformed requests are rejected
- The LLM cannot call a tool with **unexpected arguments**



Human-in-the-loop

- The host can require **user approval** before executing sensitive actions
- Critical operations (delete, modify, publish) go through a **confirmation step**



Transport security

- Local servers use **stdio** (no network exposure)
- Remote servers use **HTTPS + authentication**
- OAuth 2.1 support for **secure, scoped access** to external APIs

The principle: MCP doesn't give the AI **unrestricted access**. It gives it **controlled**, **validated**, **auditable** access with humans staying in the loop for high-stakes decisions.

Practical example: how GitHub Copilot works

Scenario

A developer in their IDE asks Copilot:
"The latest release broke the login feature. Run the test suite, find the failing test, and tell me what's wrong."

The "actors" (architecture)



The "host" (your IDE)

The VS Code interface where you type the command.



The "brain" (Remote LLM)

The powerful reasoning engine running in the cloud and the MCP client that decides which tools to call.



The "hand" (local MCP server)

A service running on your machine with approved access to your local terminal and file system.

MCP

Practical example: how GitHub Copilot works







nacorm

MCP

Practical example: how GitHub Copilot works

1. The **host** (IDE) sends your complex prompt ("…run the test suite…") to the remote "**brain**".

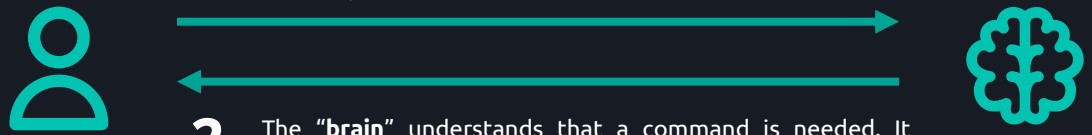




nacorm

MCP

Practical example: how GitHub Copilot works



1. The **host** (IDE) sends your complex prompt ("…run the test suite…") to the remote "**brain**".

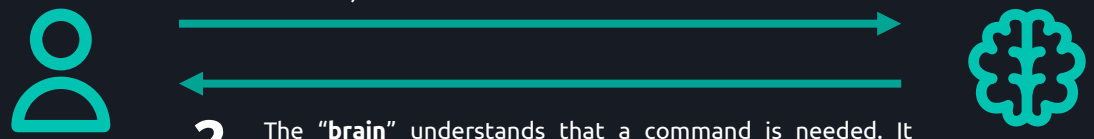
2. The "**brain**" understands that a command is needed. It sends an action plan (a standardized tool call message) back to the **IDE**, specifying:
tool_name: *run_terminal_command*
parameters: *{ command: "npm test" }*



nacorm

Practical example: how GitHub Copilot works

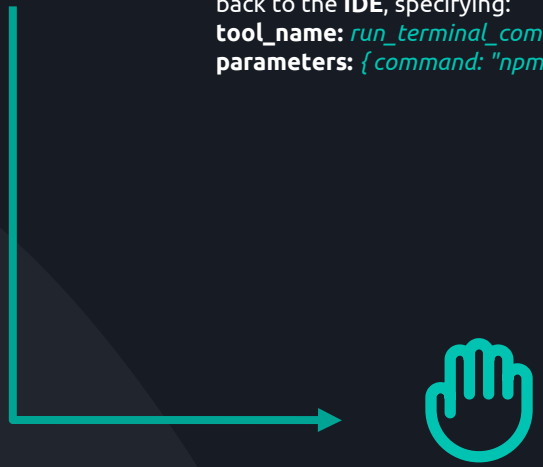
1. The **host** (IDE) sends your complex prompt ("…run the test suite…") to the remote "**brain**".



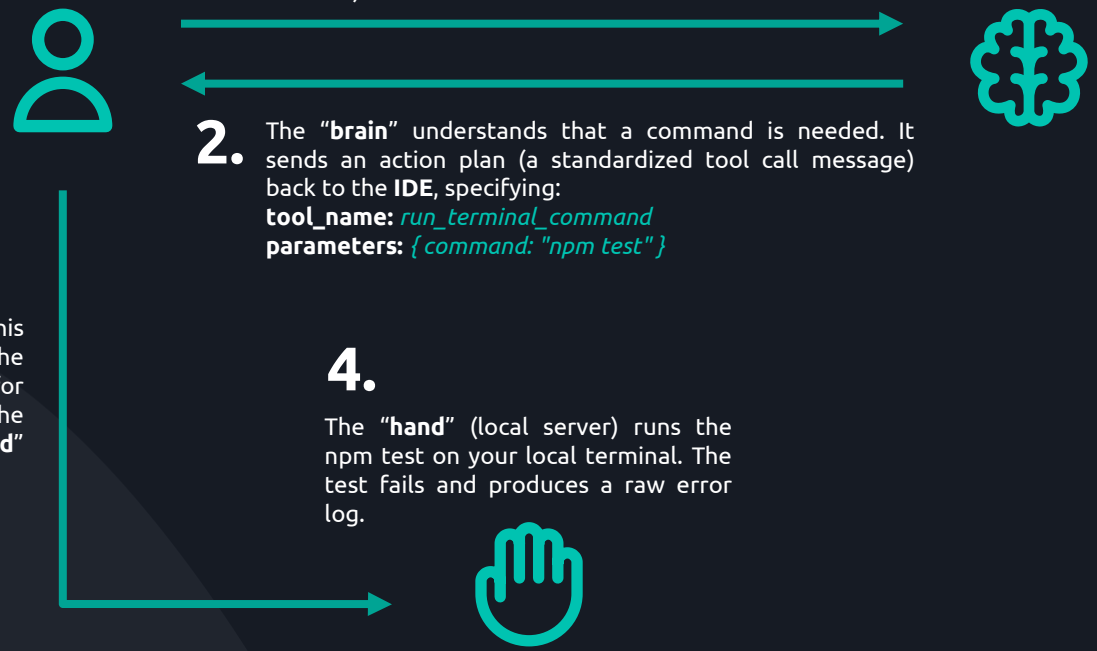
2. The "**brain**" understands that a command is needed. It sends an action plan (a standardized tool call message) back to the **IDE**, specifying:
tool_name: *run_terminal_command*
parameters: *{ command: "npm test" }*

3.

The **host** (IDE) receives this plan. It validates the command (e.g., asks for approval), then forwards the tool call to the local "**hand**" (MCP server).



Practical example: how GitHub Copilot works



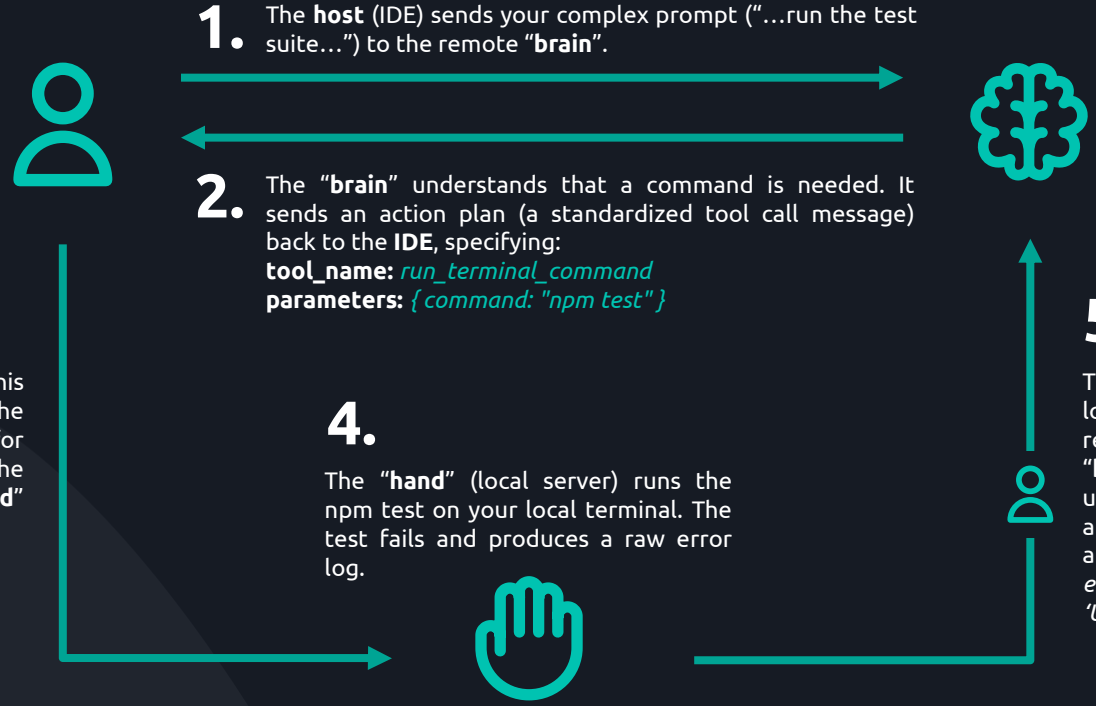
1. The **host** (IDE) sends your complex prompt ("…run the test suite…") to the remote "**brain**".

2. The "**brain**" understands that a command is needed. It sends an action plan (a standardized tool call message) back to the **IDE**, specifying:
tool_name: *run_terminal_command*
parameters: *{ command: "npm test" }*

3. The **host** (IDE) receives this plan. It validates the command (e.g., asks for approval), then forwards the tool call to the local "**hand**" (MCP server).

4. The "**hand**" (local server) runs the npm test on your local terminal. The test fails and produces a raw error log.

Practical example: how GitHub Copilot works



1. The **host** (IDE) sends your complex prompt ("…run the test suite…") to the remote "**brain**".

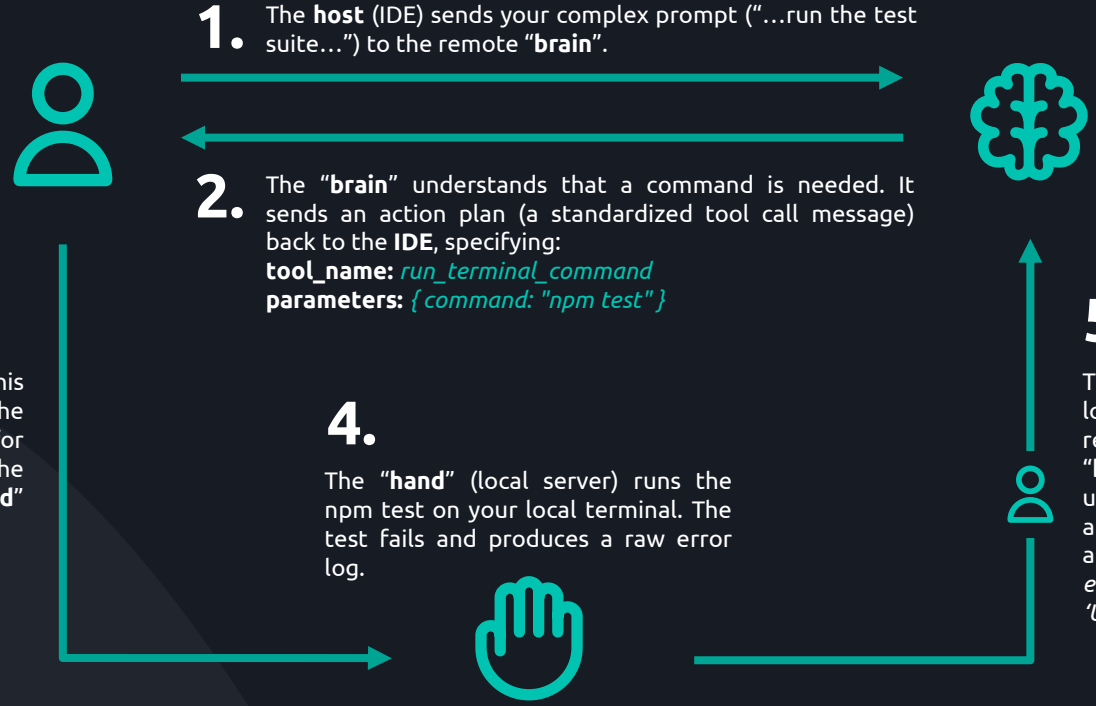
2. The "**brain**" understands that a command is needed. It sends an action plan (a standardized tool call message) back to the **IDE**, specifying:
tool_name: *run_terminal_command*
parameters: *{ command: "npm test" }*

3. The **host** (IDE) receives this plan. It validates the command (e.g., asks for approval), then forwards the tool call to the local "**hand**" (MCP server).

4. The "**hand**" (local server) runs the npm test on your local terminal. The test fails and produces a raw error log.

5. The "**hand**" captures that error log and sends it back (as a tool response) to the "brain". The "**brain**" reads the log, understands which test failed, and generates the final smart answer: *"The test suite failed. The error is in tests/auth.test.js on the 'User login' test …"*

Practical example: how GitHub Copilot works



1. The **host** (IDE) sends your complex prompt ("…run the test suite…") to the remote "**brain**".

2. The "**brain**" understands that a command is needed. It sends an action plan (a standardized tool call message) back to the **IDE**, specifying:
tool_name: *run_terminal_command*
parameters: *{ command: "npm test" }*

3. The **host** (IDE) receives this plan. It validates the command (e.g., asks for approval), then forwards the tool call to the local "**hand**" (MCP server).

4. The "**hand**" (local server) runs the npm test on your local terminal. The test fails and produces a raw error log.

5. The "**hand**" captures that error log and sends it back (as a tool response) to the "brain". The "**brain**" reads the log, understands which test failed, and generates the final smart answer: *"The test suite failed. The error is in tests/auth.test.js on the 'User login' test …"*

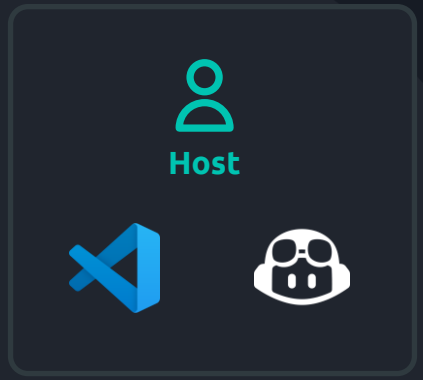
Summary: The local "**hand**" can act and see results in the real world. The remote "**brain**" provides the intelligence needed to decide what to do and interpret those results.



HANDS ON

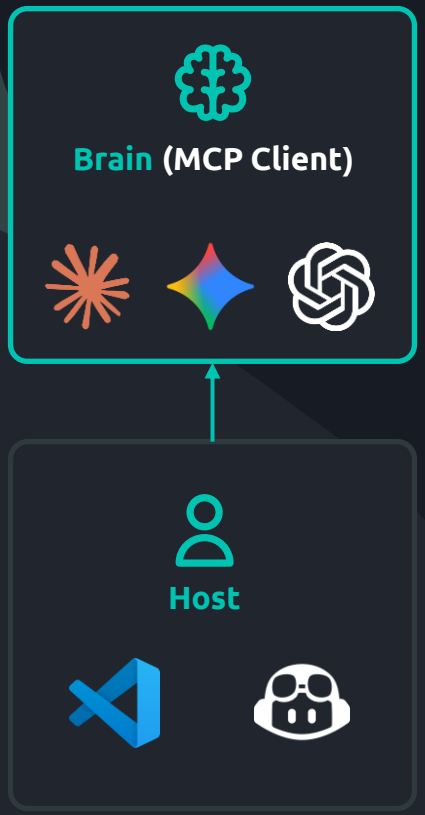
HANDS ON

1 - Establishing MCP Communication



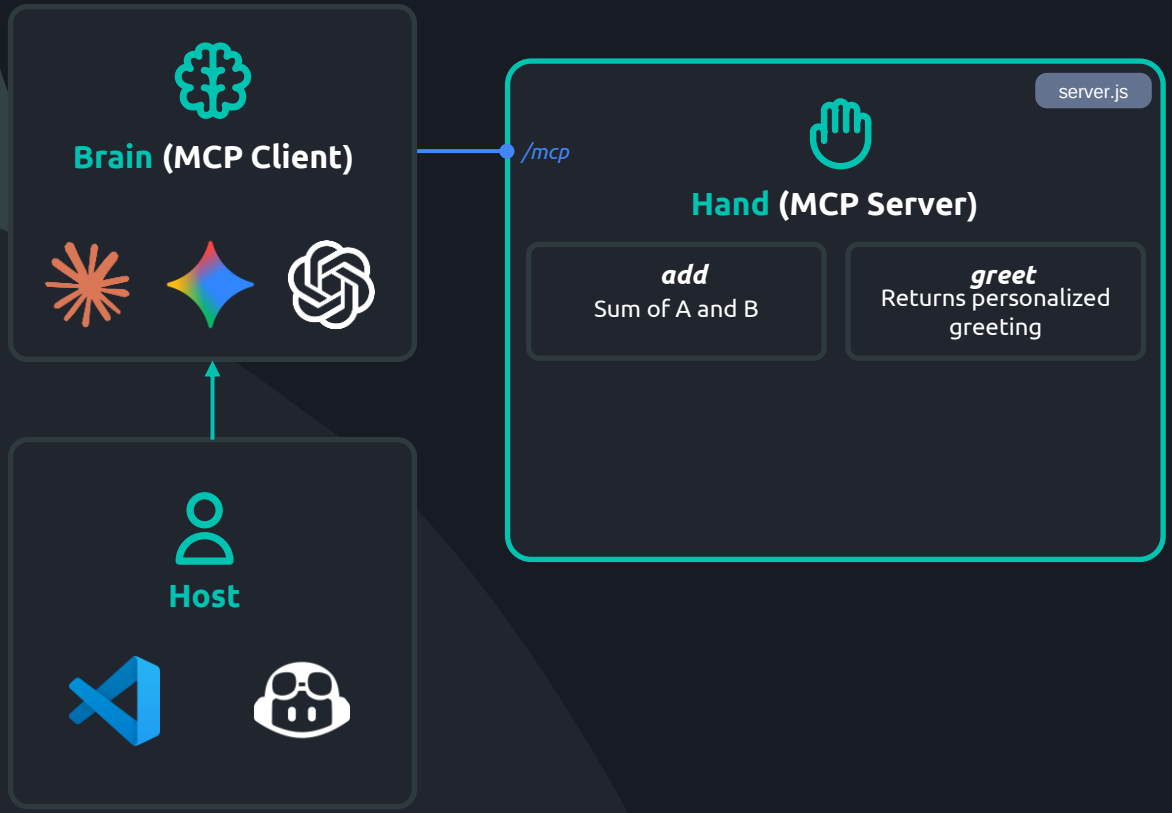
HANDS ON

1 - Establishing MCP Communication



HANDS ON

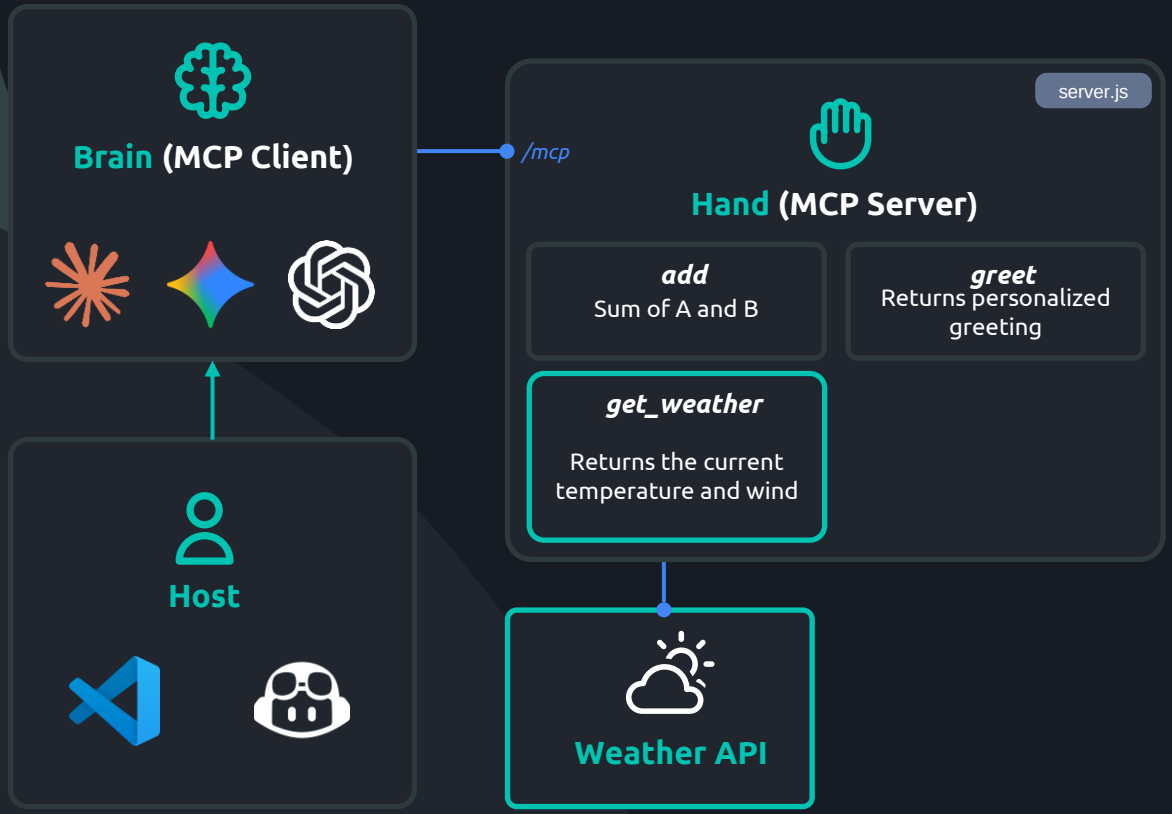
1 - Establishing MCP Communication



nacorm

HANDS ON

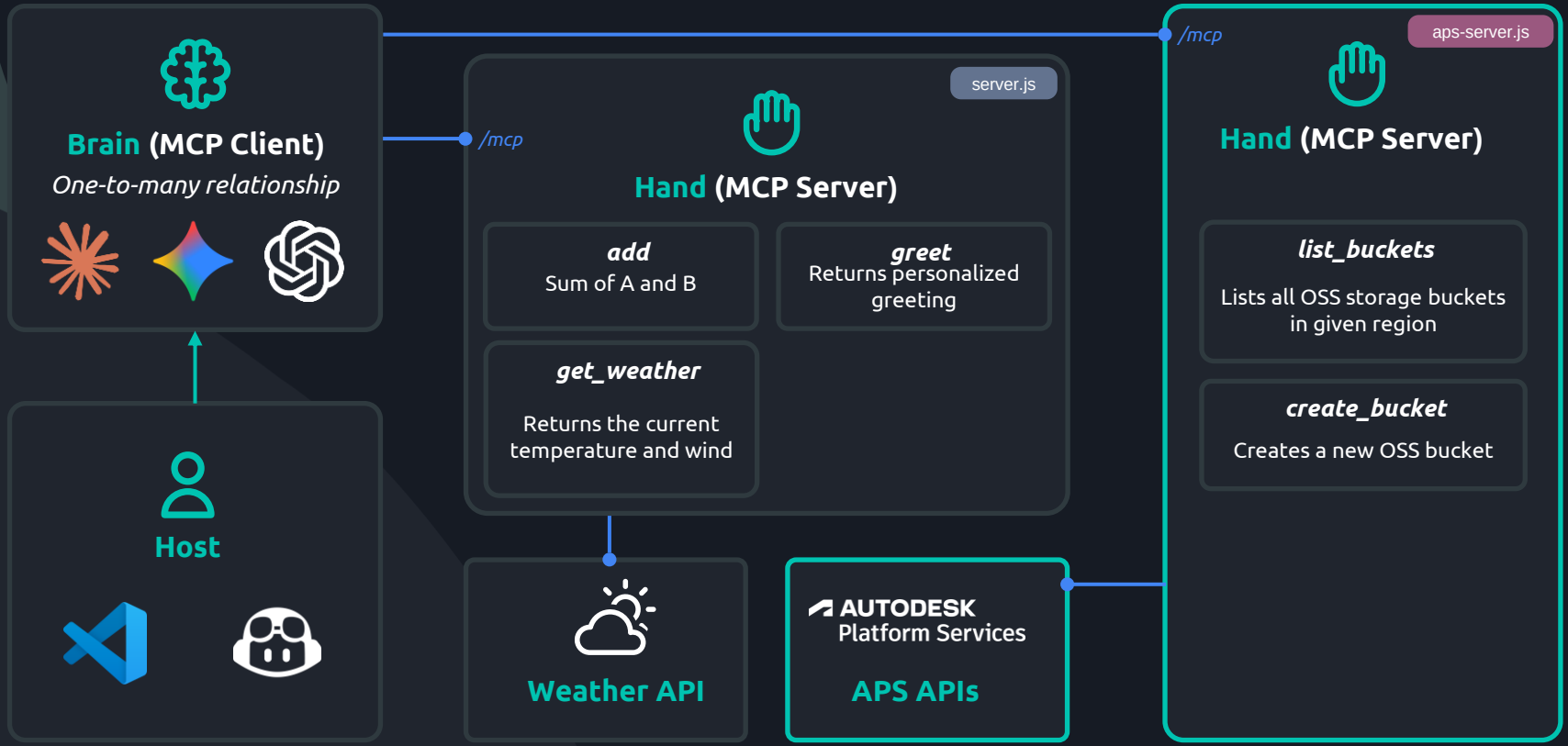
2 - Linking with External Tools



nacorm

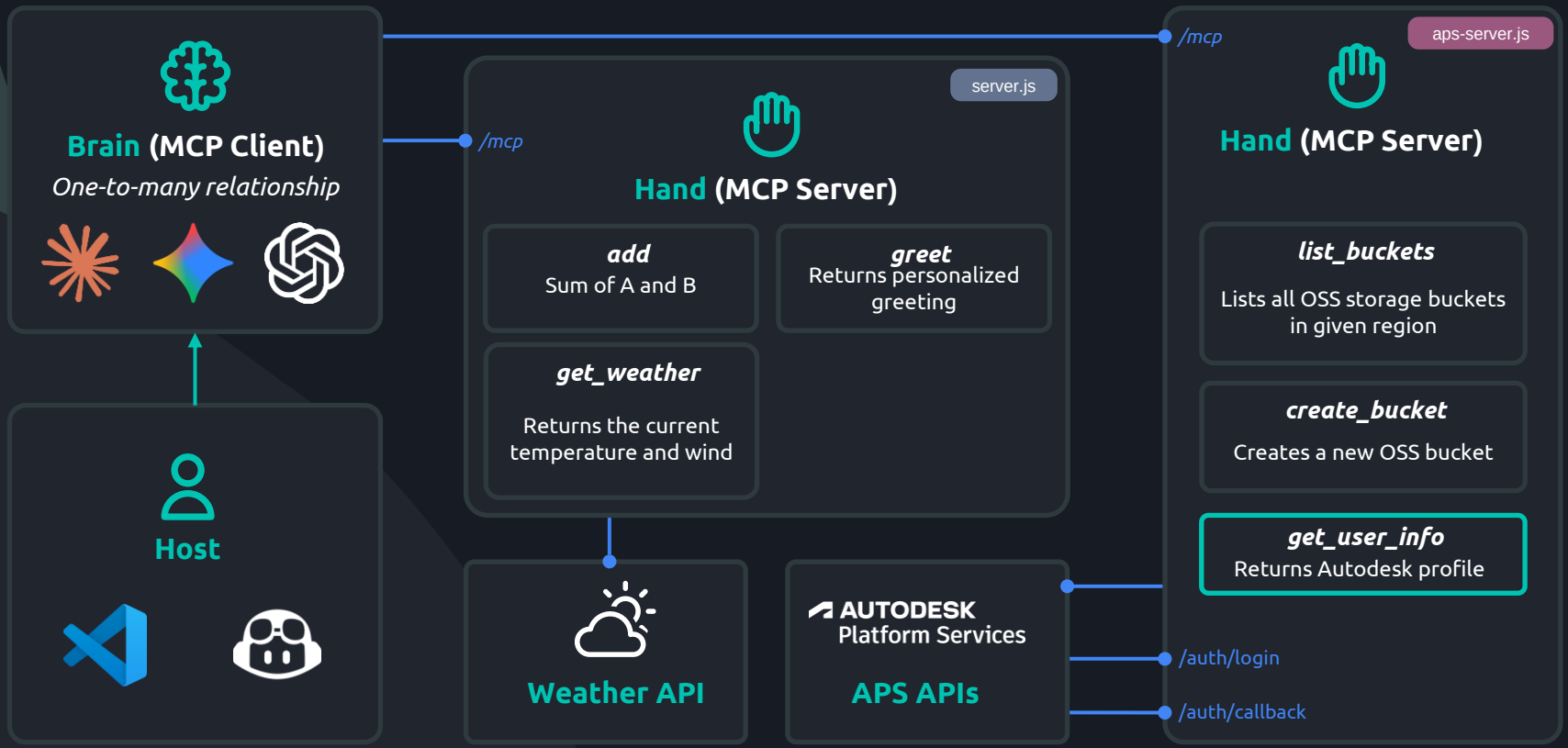
HANDS ON

3 - Connecting Autodesk APS (2-Legged)



HANDS ON

4 - User Authentication with APS (3-Legged)



Thank you



Nabil Sadeg
nabil.sadeg@nacorm.com
nacorm.com

This presentation was created by Nacorm for Autodesk DevCon MCP Workshop held on April 14, 2026. It contains illustrative examples provided for informational purposes only and does not constitute a final solution. Any reproduction, distribution, or use, in whole or in part, without prior authorization is strictly prohibited. All content is protected by copyright and remains the property of its respective rights holders.