DOULOS KnowHow TUTORIAL

Delivering KnowHow

www.doulos.com

Beginners Guide to Using AI for Hardware Engineers

Presented by Doug Smith

Where do we start?

Robotics

Deals with the design, construction, operation, and application of robots

Deep Learning

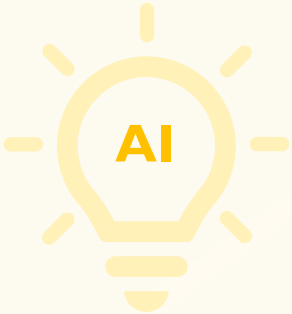
A subset of machine learning that uses neural networks with multiple layers to model and solve complex problems

Knowledge Representation and Reasoning

Focuses on representing knowledge in a form that a computer system can utilize to solve complex tasks

Neural Networks

Computer systems modeled after the neural connections in the human brain



Cognitive Computing

Mimics human thought processes and augments human cognition

Expert Systems

Software systems that emulate the decision-making ability of a human expert in a particular domain

Natural Language Processing (NLP)

Focuses on the interaction between computers and human languages

Computer Vision

Involves enabling machines to interpret and make decisions based on visual data

Machine Learning

The study of computer algorithms that improve automatically through experience and data

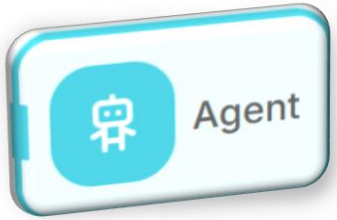
Our focus

Only a subset of AI

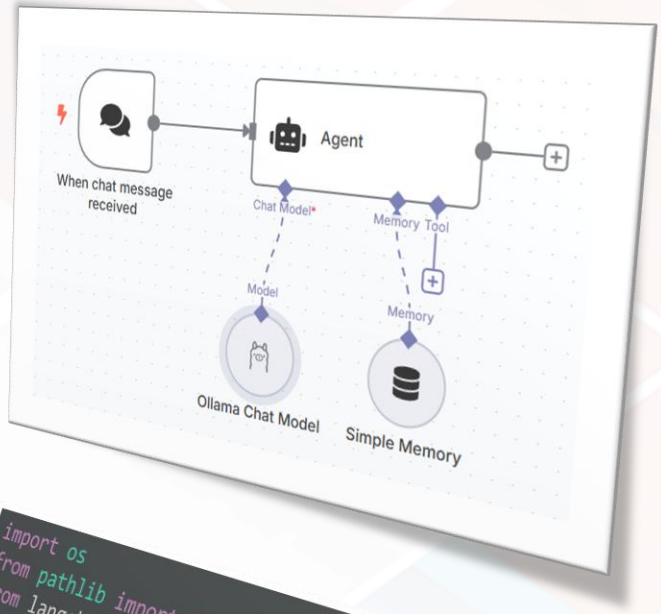
How does AI work?

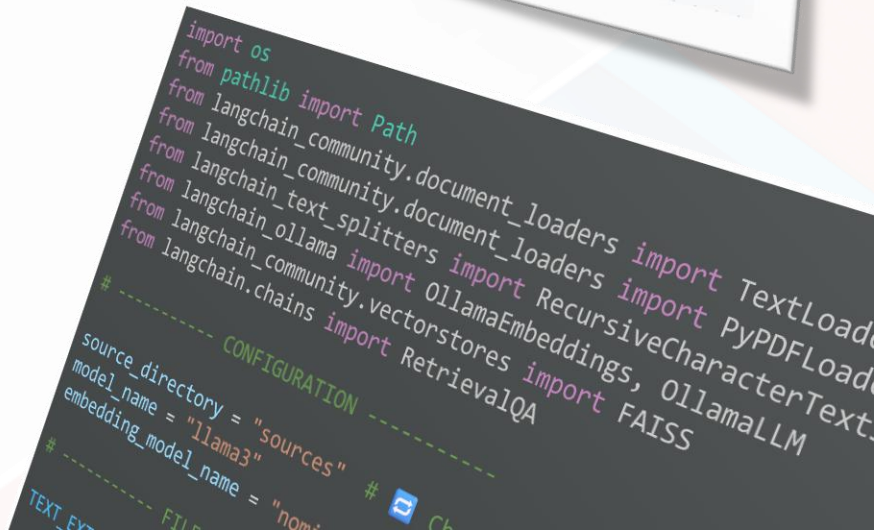
How do I work with AI?

How do I use AI in engineering?



How does AI work?

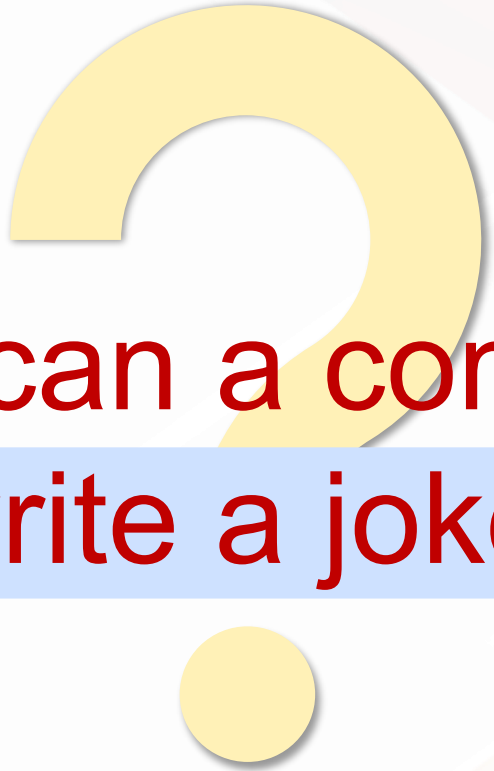




Ask AI

+ Write a funny joke. ↑

"Why couldn't the bicycle stand up by itself? It was two-tired."



How can a computer write a joke?

The fundamental problem

Computers speak in numbers, not English.

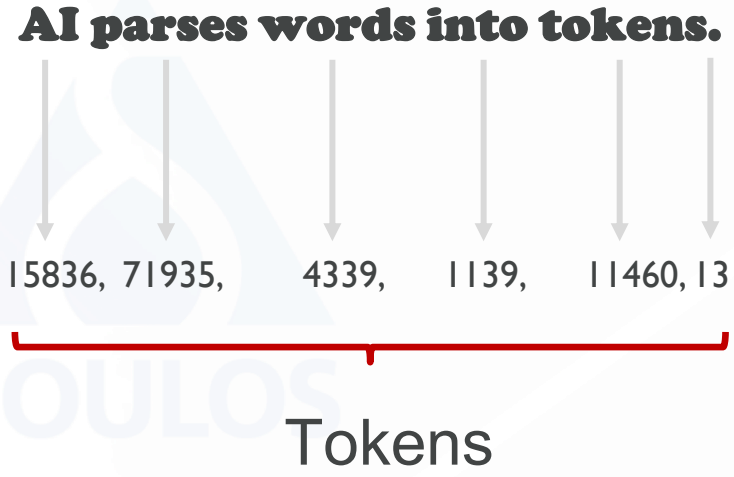
Human language has nuance and context

Natural Language Processing (NLP)

[10445, 7846, 956, 279, 36086, 2559, 709]

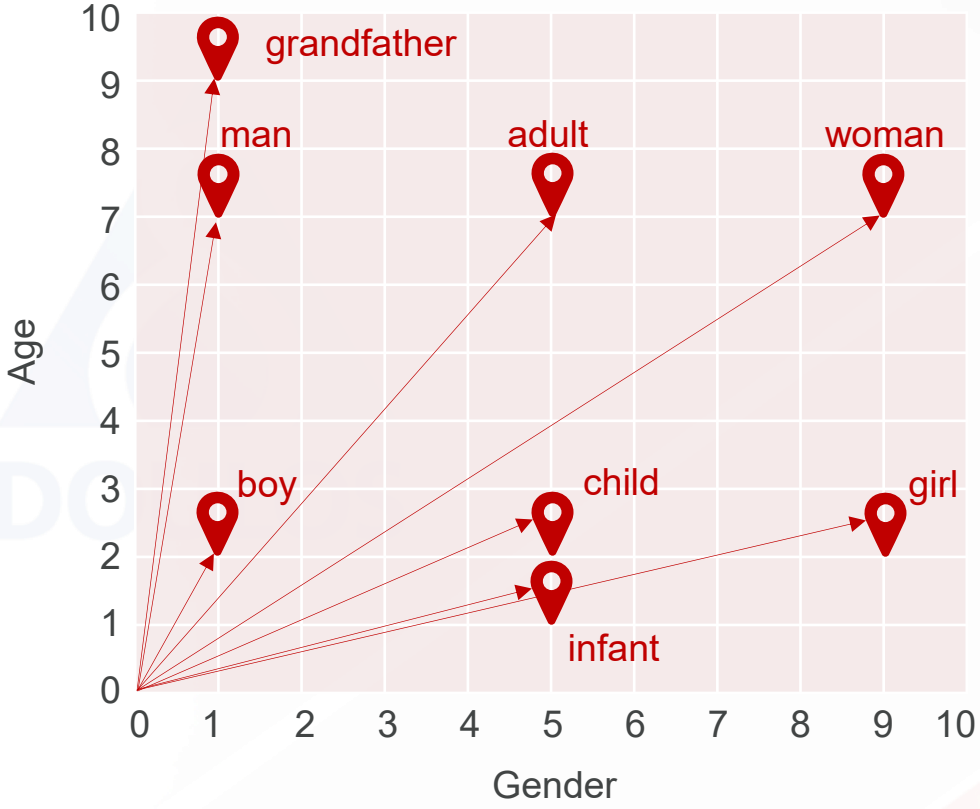
Machine language has numbers and vectors

Chopping up language



A **token** is a unit of text, like an entry in a dictionary, but it has no **meaning**.

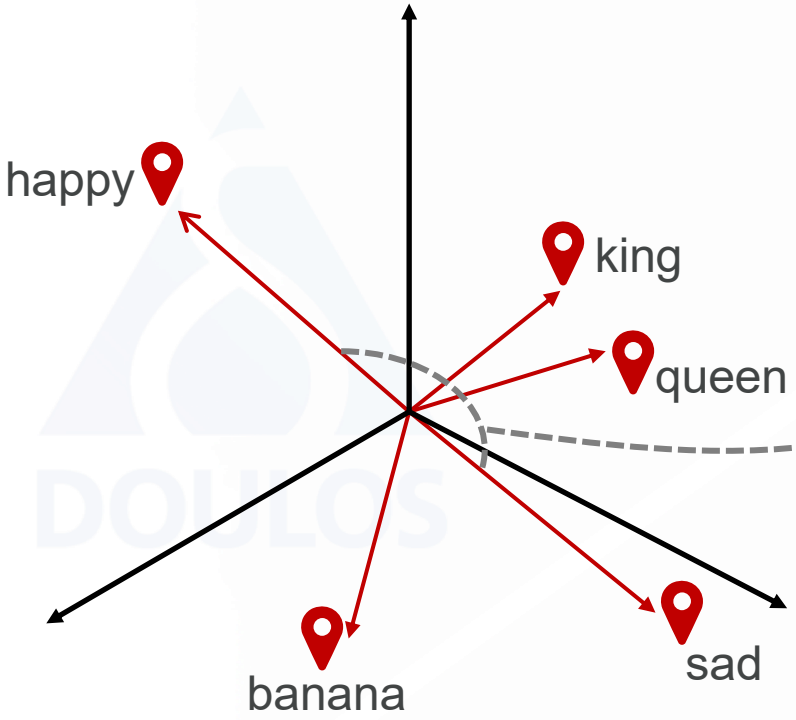
Embeddings



	Gender		Age
boy	(1,2)	woman	(9,7)
man	(1,7)	girl	(9,2)
adult	(5,7)	infant	(5,1)
child	(5,2)	grandfather	(1,9)

An **embedding** is a token's address into '**meaning space**'.

Word relationships



Angle and direction between vectors encode **similarity** and **relationships**

Score	Meaning	Similarity	
1	Same direction	High	king & queen
0	Unrelated	None	king & banana
-1	Opposite direction	Low	happy & sad

Attention to context

1

(noun)

She rode the train.

She trained the dog.

(verb)

2

She broke the record.

She broke the record player.

(related)

"Attention" means looking at a word's **context** in a sentence.

Meaning has many dimensions

She rode the train .

Tokens: 8100, 41761, 279, 5542,13

Embeddings: [0.39886564016342163, -0.404699176549911 5, -3.179293632507324, 0.20834633708000183, 1.0488201379776, …] […] […] […] [-1.7465770244598389, 1.24246084690094, -4.810368537902832, 0.3165740668773651, 0.9620928764343262, …]

(768 length vector)

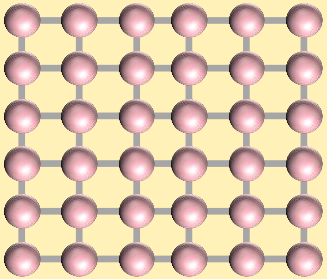
Large language models

She rode the ...

Embeddings

[0.39886564016342163,
-0.404699176549911 5,
-3.179293632507324,
0.20834633708000183,
1.0488201379776,
1.414209246635437,
…]

LLM
(Transformer)



Predicted probability

train	0.08003556728363037
bus	0.07108145207166672
bike	0.024848591536283493
elevator	0.019790327176451683
back	0.01585303619503975

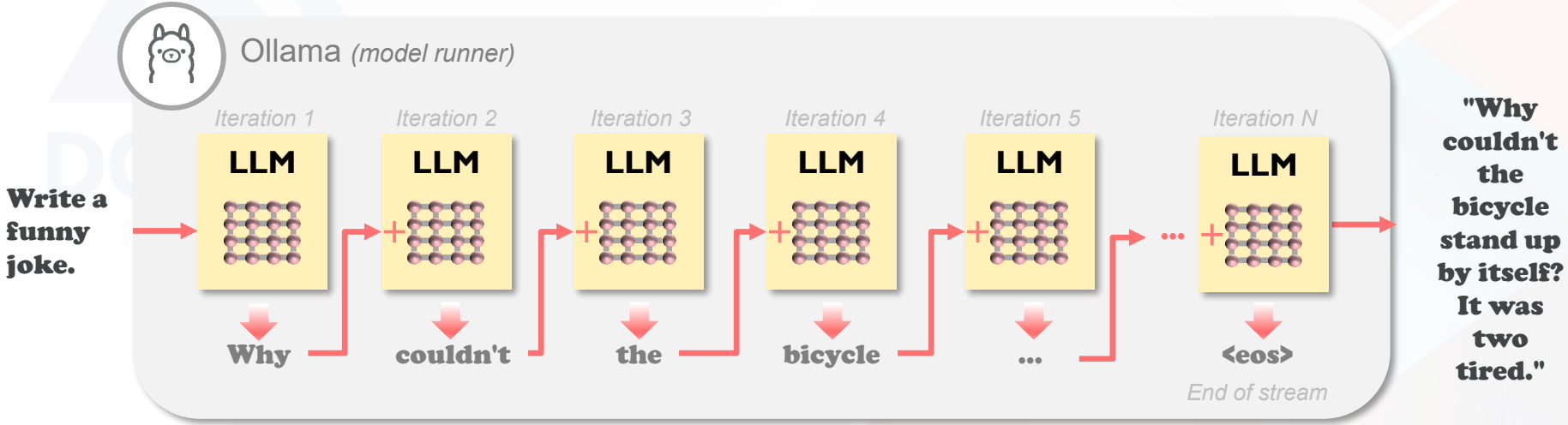
She rode the train

Achieving thought

+ Write a funny joke. ↑

How?

"Why couldn't the bicycle stand up by itself? It was two-tired."



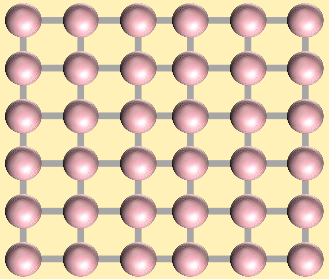
Generative Pre-trained Transformer = GPT

A **"transformer"** transforms our language into a new representation that captures **meaning**, **context**, and **relationships**.

See https://poloclub.github.io/transformer-explainer/

Hallucinations

LLM
(Transformer)



train	0.08003556728363037
bus	0.07108145207166672
bike	0.024848591536283493
elevator	0.019790327176451683
back	0.01585303619503975

She rode the train
(temp = 0.2)

She rode the back
(temp = 0.8)

"Hallucination" is when an output is **valid** but **factually incorrect** based on its training data or input context

temperature	smaller → more accurate
	larger → more creative
top k	picks from top *k* choices
top p	picks from at least *p*

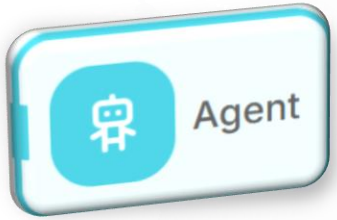
Example

GPT4All Llama 3.1 8B Instruct 128k

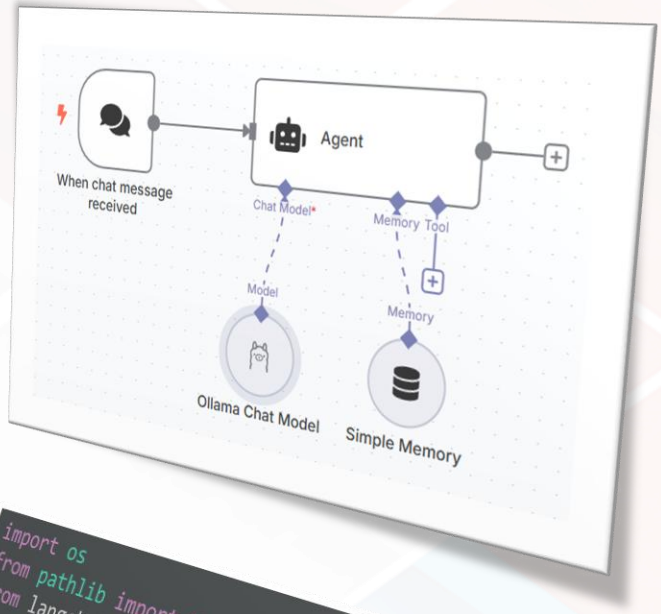
Here are two simple sentences that use the word "cloud" with different meanings:

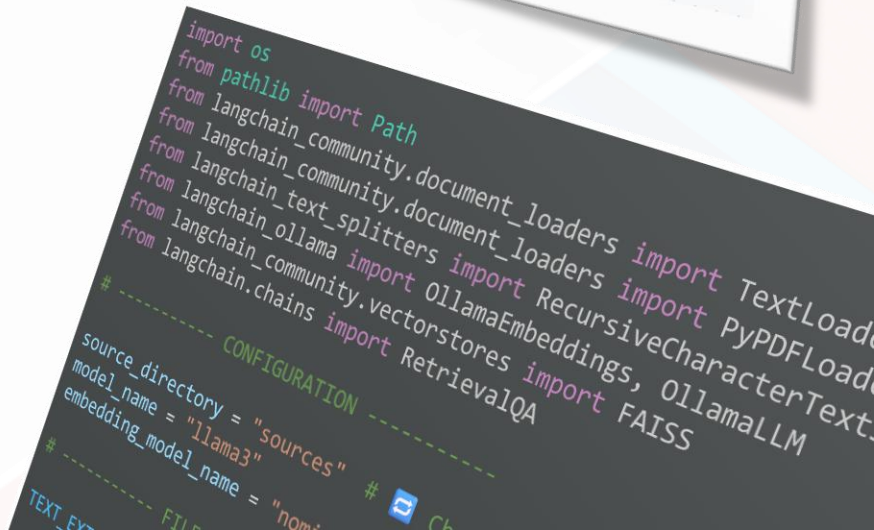
1. The fluffy white cloud floated across the sky.
2. The company's profits were in the red, and they had to borrow money from a bank to cover their expenses.

In this example, "cloud" refers to a collection of water vapor or gas in the atmosphere (sentence 1), while it also means an amount owed that is not yet paid (sentence 2).



Working with AI





LLMs

Key factors of an LLM:

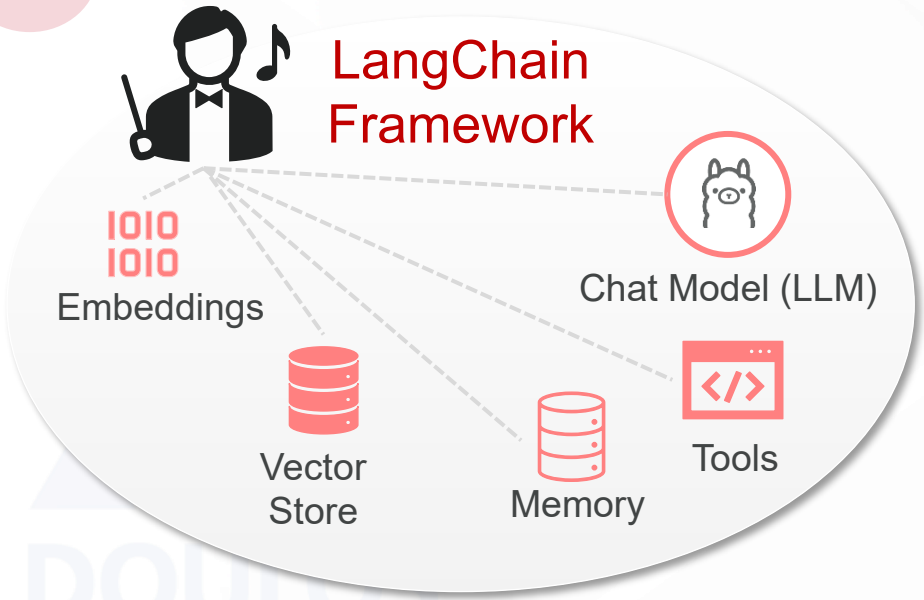
Parameters
Licensing
Tool support
GPUs

Parameters are weights

More weights ...
→ more brain power
→ more memory
→ more GPUs

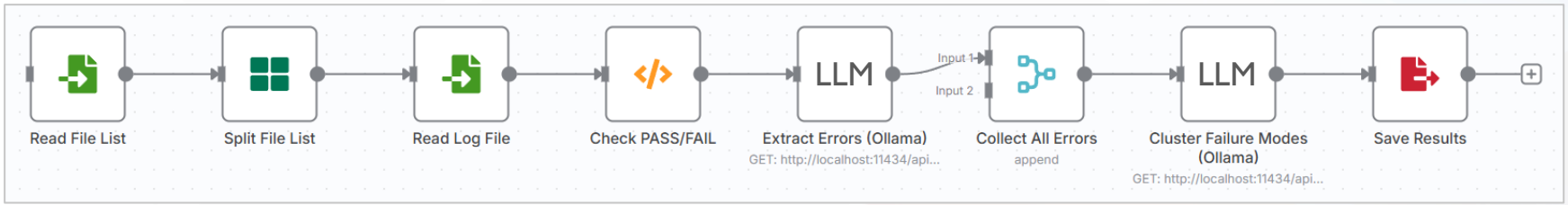
Model	Parameters	License	Originator	Tool Support	Commercial Use	GPU & RAM Requirements
OpenLLaMA 3.1 / 3.2 (Tool-enabled)	8B – 70B	OpenLLaMA Community	Meta AI / OpenLLaMA	✅ Yes	✅ Yes	1× NVIDIA A100 40GB or RTX 3090 24GB; 32–64GB RAM
Mistral / Mixtral (Instruct / Tool)	7B, 8×7B	Apache 2.0	Mistral AI	✅ Yes	✅ Yes	1× NVIDIA A100 40GB or RTX 3090 24GB; 32–64GB RAM
Falcon (Tool / Instruct)	7B, 40B, 180B	Apache 2.0	TII (UAE)	✅ Yes	✅ Yes	2× NVIDIA A100 40GB; 64–128GB RAM
GPT-NeoX / GPT-J (Tool)	6B, 20B (NeoX), 6B (J)	Apache 2.0	EleutherAI	✅ Yes	✅ Yes	2× NVIDIA A100 40GB; 64–128GB RAM
DeepSeek-LLM / V3 / R1	7B, 67B, 236B dense; 671B total (MoE, 37B active)	DeepSeek License	DeepSeek (China)	✅ Yes (native tool reasoning)	✅ Yes	2× NVIDIA A100 40GB or RTX 3090 24GB; 64–128GB RAM
Gemma (1–3 series, Tool)	1B – 27B	Gemma License (open-weights)	Google DeepMind	✅ Yes	✅ Yes	1× NVIDIA A100 80GB or RTX 4090 24GB; 32–64GB RAM
Grok (1, 2.5)	~314B (dense est.), details vary	Apache 2.0	xAI (Elon Musk / X)	✅ Yes	✅ Yes	8× NVIDIA A100 40GB; 512GB RAM
OpenAI GPT-4 (API / Plugins)	175B+	OpenAI	OpenAI	✅ Yes	✅ Yes (via API)	5× NVIDIA A100 80GB; 400GB+ RAM
OpenAI GPT-4-turbo (Function Calling)	175B+	OpenAI	OpenAI	✅ Yes	✅ Yes (via API)	5× NVIDIA A100 80GB; 400GB+ RAM
GPT-OSS (20B / 120B)	20B, 120B	Apache 2.0	OpenAI	✅ Yes	✅ Yes	1× NVIDIA A100 80GB (120B); 1× RTX 4090 24GB (20B); 32–64GB RAM
Qwen (Tool-enabled variants)	3B, 7B, 14B, 72B	Apache 2.0	Alibaba Cloud	✅ Yes	✅ Yes (with conditions)	1× NVIDIA A100 40GB or RTX 3090 24GB; 32–64GB RAM

Chaining



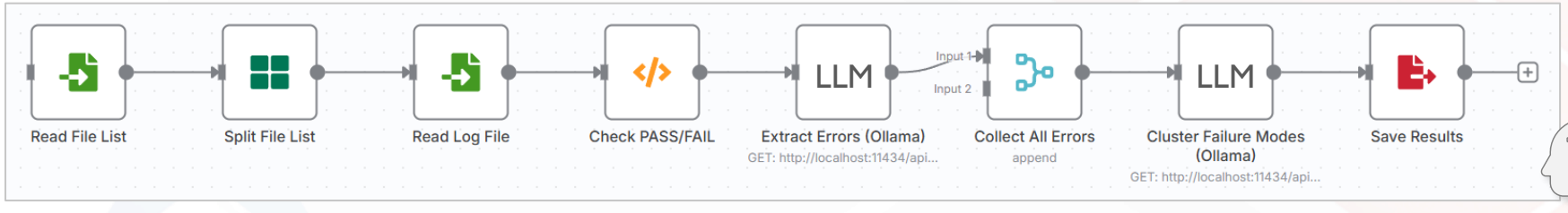
A "chain" is a sequence of steps that process input data through components like prompts, models, and memories.

An example chain of steps



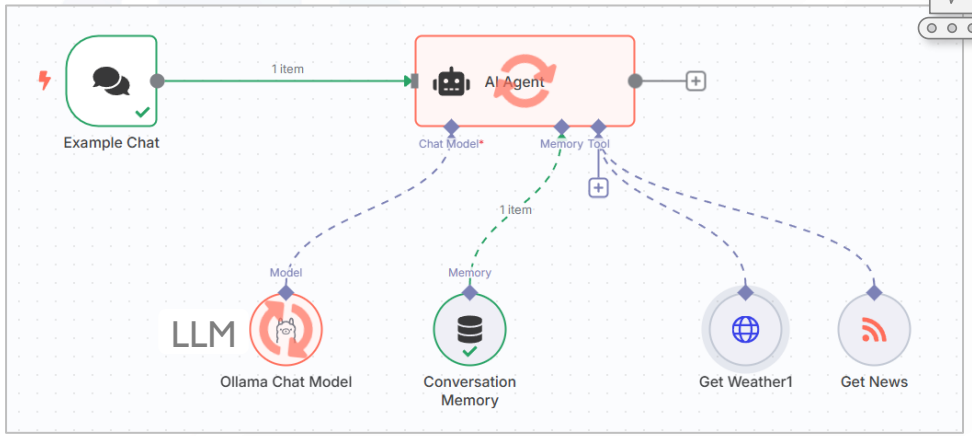
Workflows and agents

Workflow



Human interaction

Agent (Agentic)

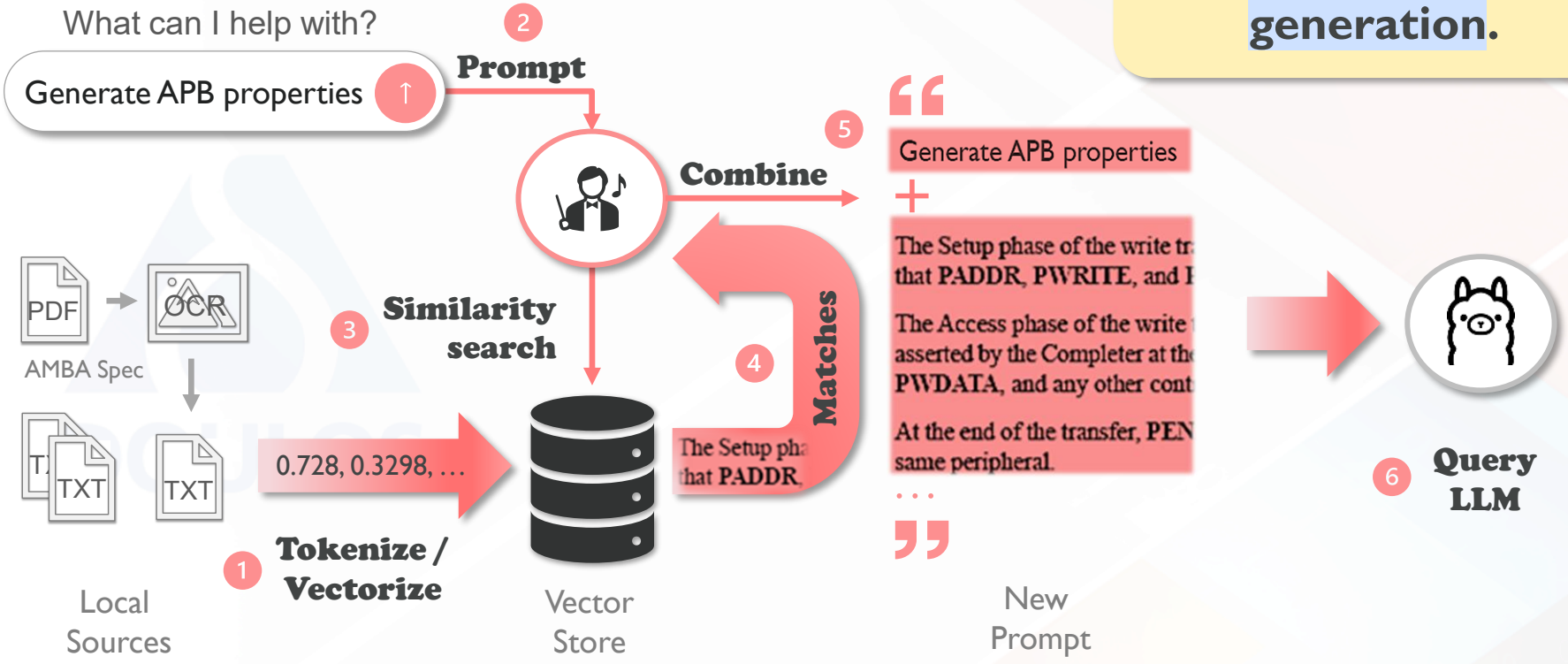


AI decides

An **"agent"** independently decides what it will do next.

Document stores (RAG)

"**RAG**" refers to **retrieval augmented generation.**

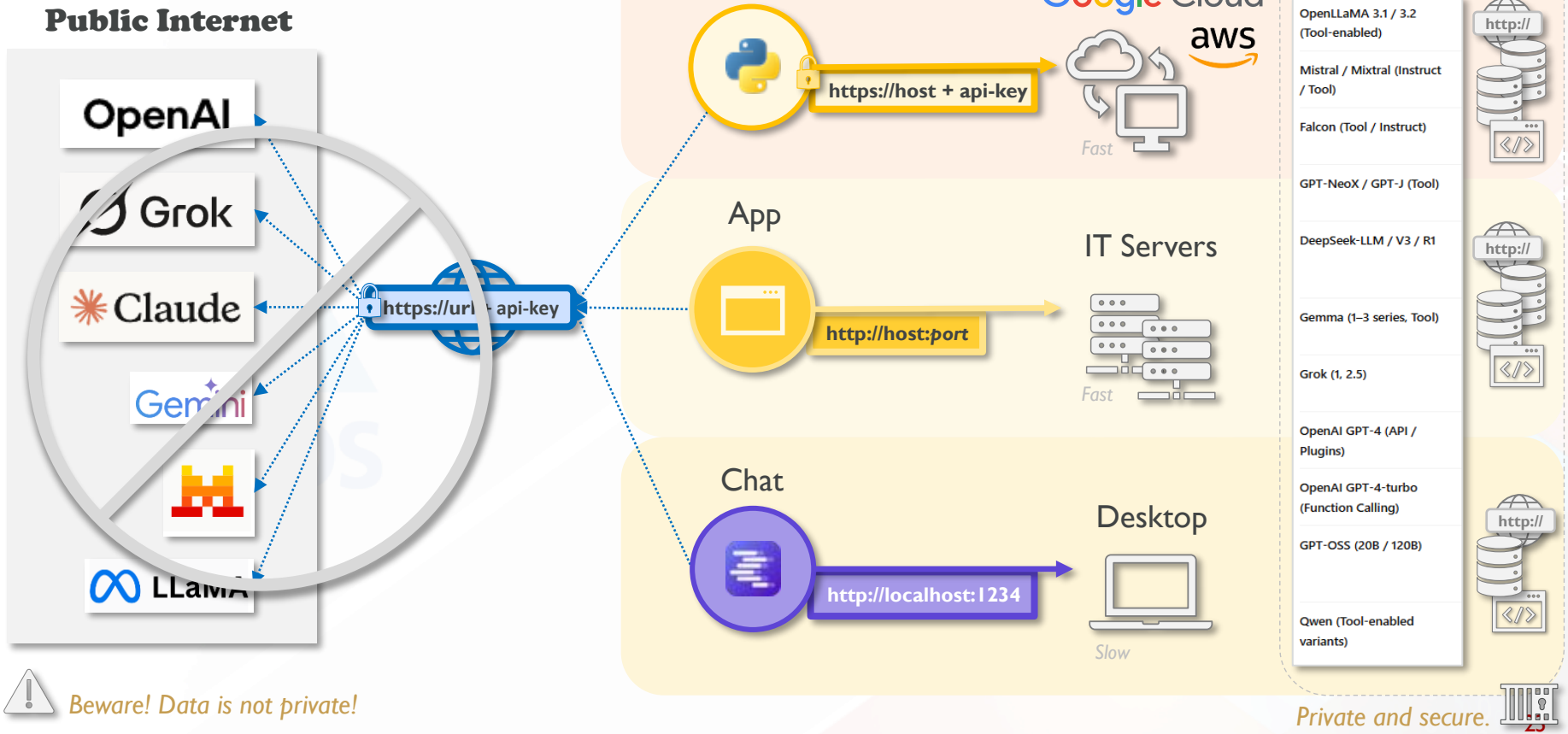


Vectors store context and meaning so better than text

Prompting

- **Task** – always start a task with a generate verb
- **Context** – ask yourself 3 questions :
  - What's the user's background?
  - What does success look like?
  - What environment are you in?
- **Exemplars** – give examples of good and bad templates
- **Persona** – who do you want the AI to be?
- **Format** – how does the result look like?
- **Tone** – indicate the feeling – casual, formal, witty, etc.

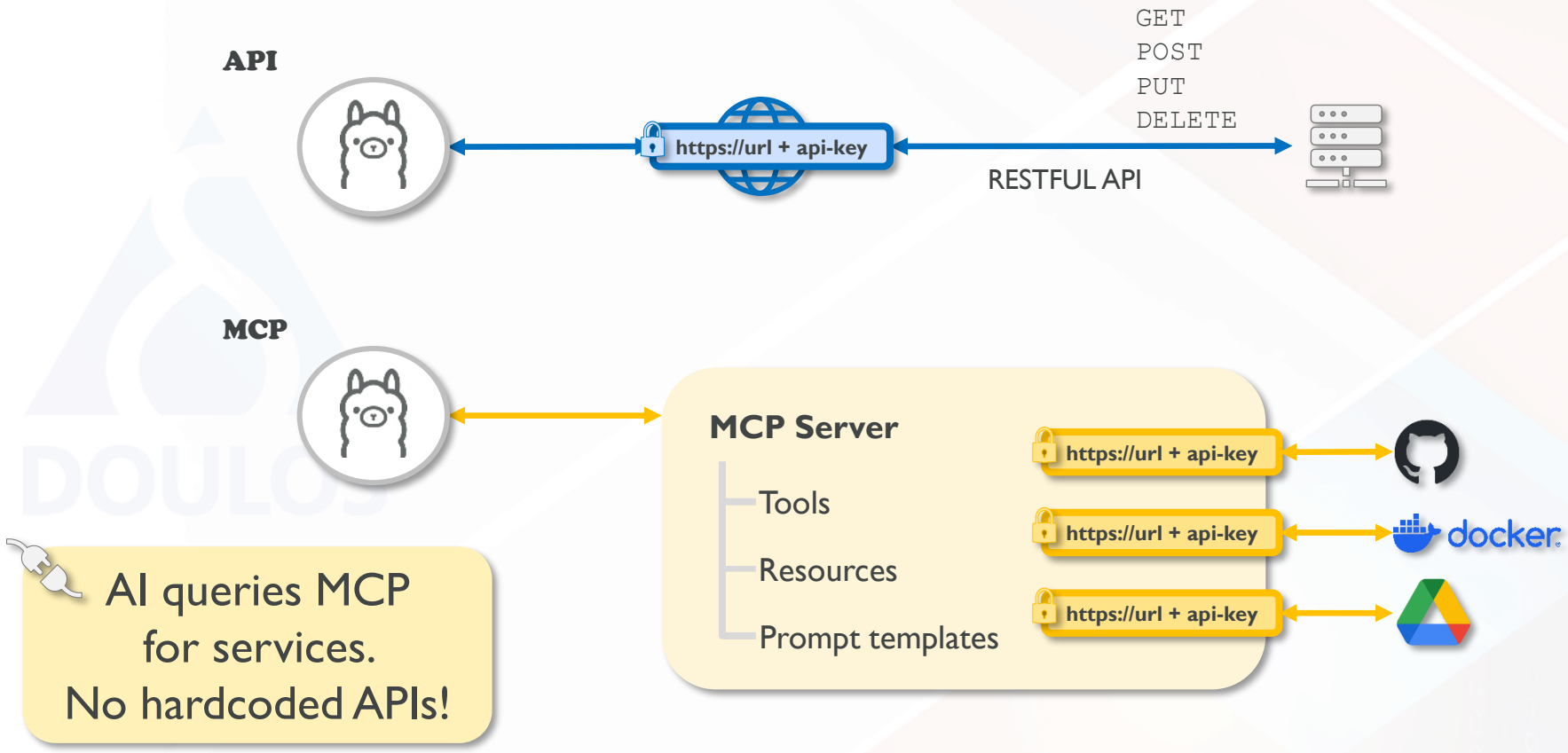
Connecting to AI

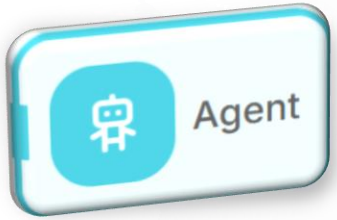


Beware! Data is not private!

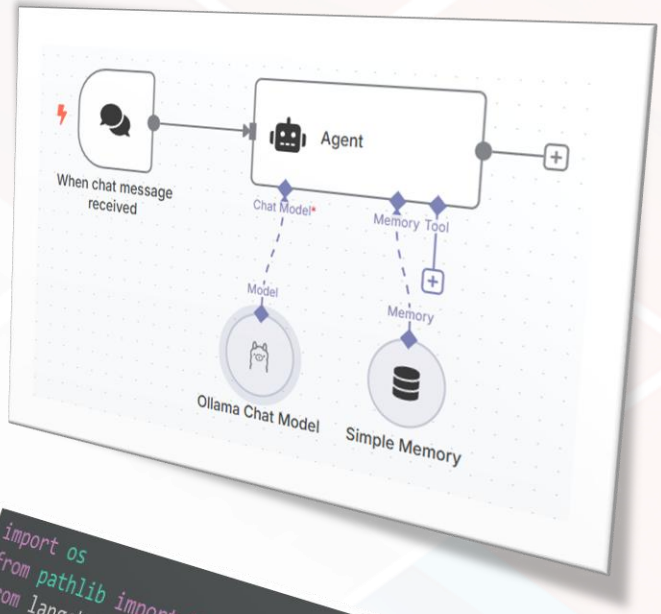
Private and secure.

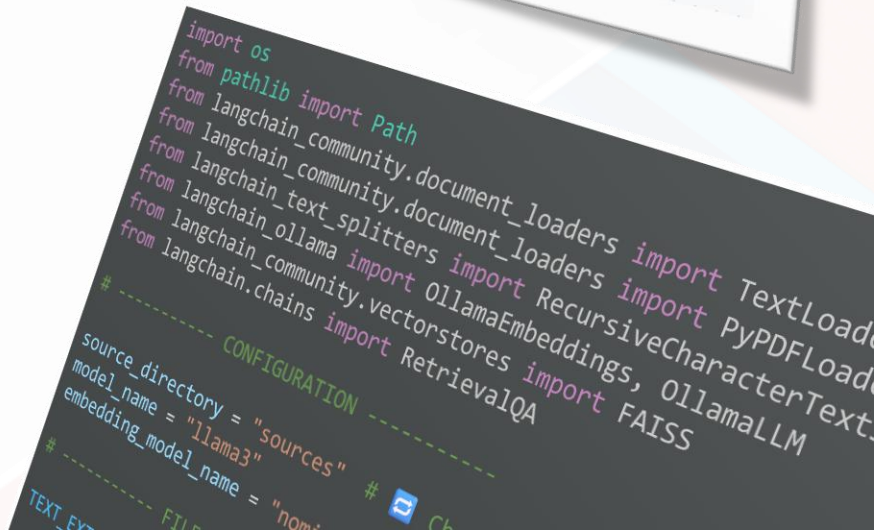
Model Context Protocol (MCP)





AI for Engineering





Use modes

Generative AI

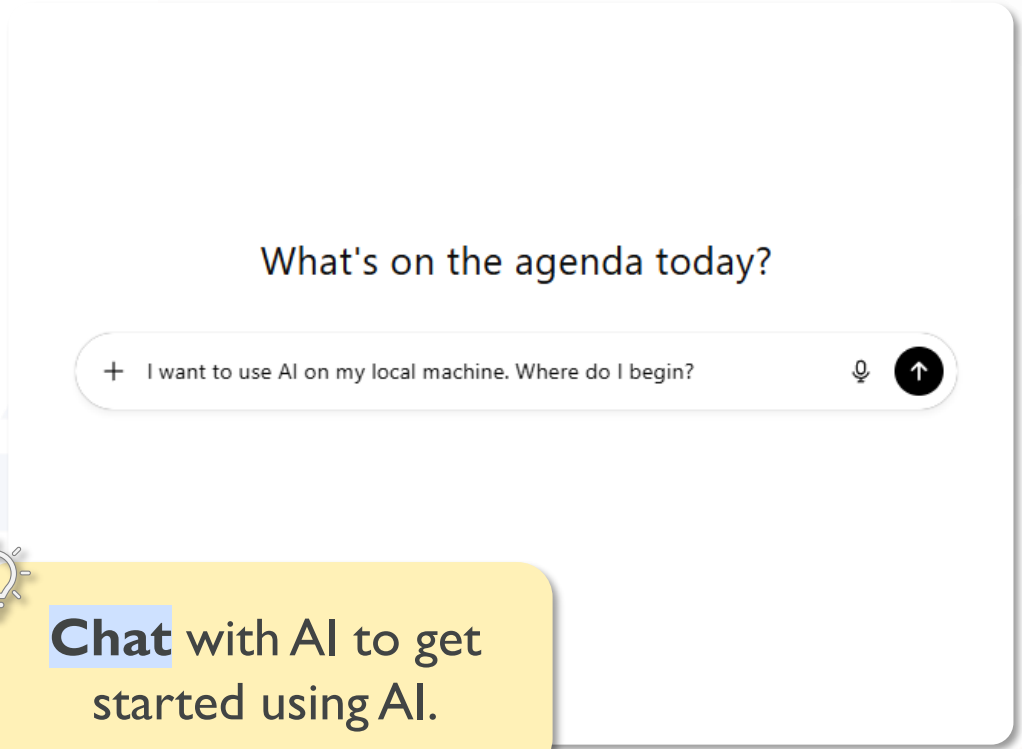
RTL/HDL Generation	Generate synthesizable RTL code from high-level descriptions + specifications
Testbench Generation	Create verification environments, stimulus, and test vectors
SVA/Assertion Generation	Create SystemVerilog Assertions (SVA) to cover corner cases
Specification Summarization	Read long design specs and produce concise summaries or design outlines
Documentation / Commenting	Generate design documentation, comments, or explanations for existing RTL

Analytical / Assistive AI

Bug / Issue Detection	Analyze RTL/testbench code to find bugs or inconsistencies
Performance / Timing Estimation	Predict timing paths or bottlenecks based on RTL patterns
Design Rule Checking	Check RTL against coding standards or design rules
Test Coverage Analysis	Evaluate coverage metrics and suggest missing tests
Debug Assistance	Analyze simulation logs and highlight probable failure causes
Specification Consistency Checking	Compare design spec vs RTL/testbench to find gaps or contradictions

Where do I begin?

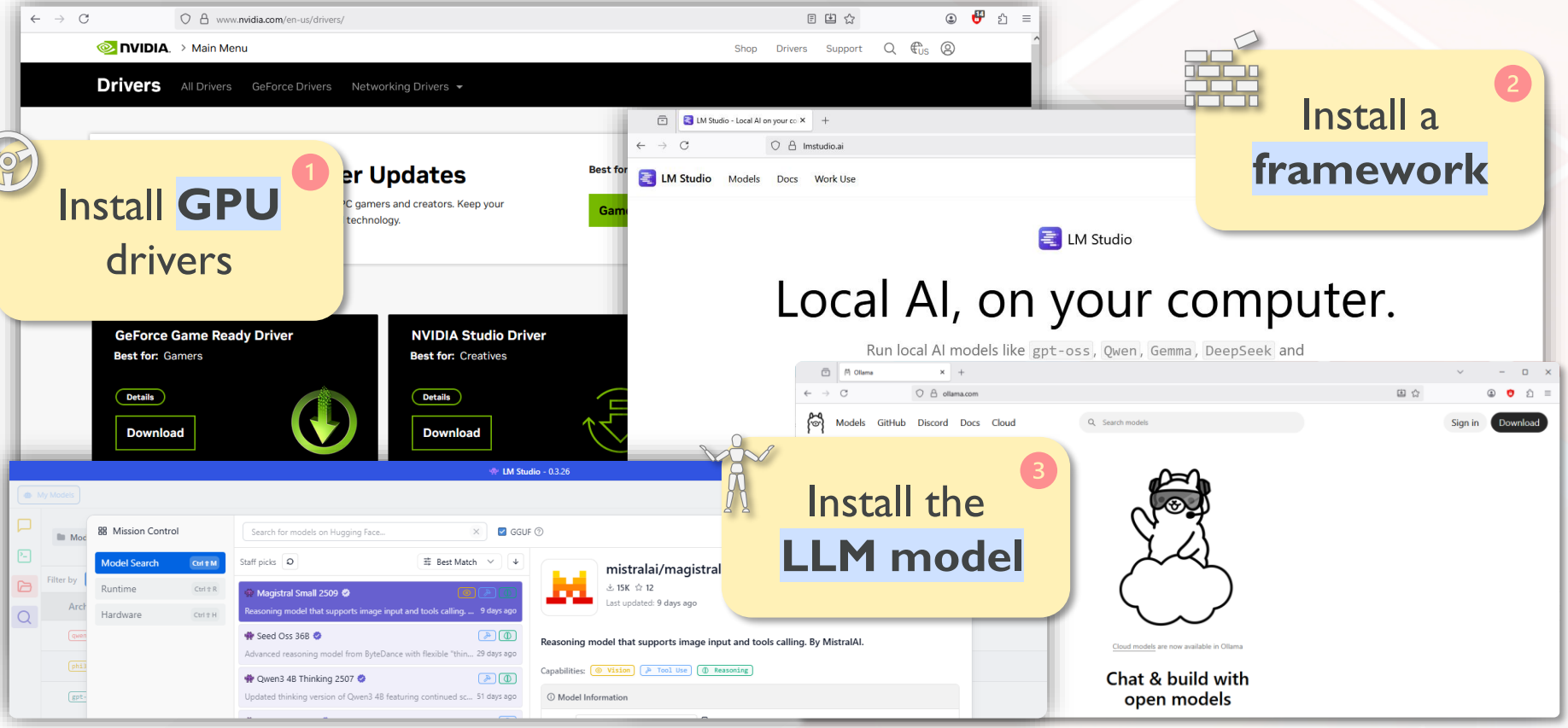
Ask Chat



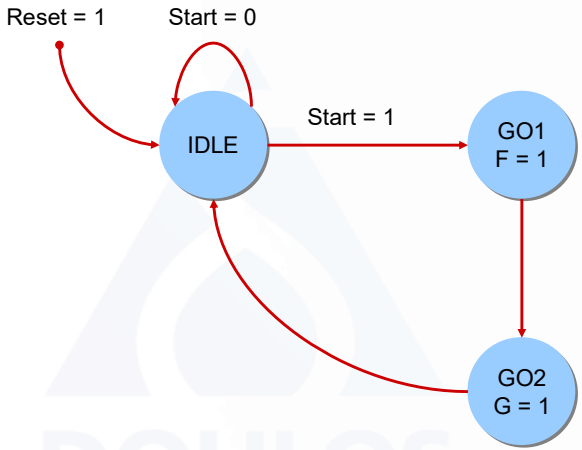
Chat with AI to get started using AI.

+ How do I install a local LLM?
+ How do I use it in docker?
+ Write a script using Ollama
+ How do I enable my GPU?
+ Add RAG to my script
+ How do I install an AI model?

Local LLM install



Experiment with LLMs



I am a hardware engineer and I need to generate a finite state machine written in SystemVerilog. There are 3 states: IDLE, GO1, and GO2. The state machine transitions to IDLE on 'Reset' = 1. When the 'Start' = 1, the state machine transitions to GO1 and the output 'F' = 1. When in the GO1 state, the state machine transitions to GO2 and the output 'G' = 1. The defaults for 'F' and 'G' are 0. When in the GO2 state, the state machine transitions back to IDLE. Generate this state machine as a SystemVerilog module.

Turbo gpt-oss:20b

Thought for 91.4 seconds

Below is a compact, synthesizable SystemVerilog implementation of the requested FSM.
It uses a two-bit enumerated state type, a synchronous state register, and combinational logic for the next state and the two outputs **F** and **G**.

Generative AI

LLMs are already trained in hardware description languages.

Copy

```
/*****************************************************************
 *  Module:  simple_fsm
 *  Purpose: 3-state FSM (IDLE → GO1 → GO2 → IDLE)
 *
 *  Inputs:
 *      clk   : Clock (positive-edge synchronous)
 *      reset : Asynchronous active-high reset
 *      start : Pulse that moves the FSM out of IDLE
 *
 *  Outputs:
 *      F     : 1 when the FSM is in state GO1
 *      G     : 1 when the FSM is in state GO2
 *****************************************************************/
module simple_fsm (
    input  logic clk,     // Clock
    input  logic reset,   // Asynchronous active-high reset
    input  logic start,   // Start pulse
    output logic F,       // Output asserted in GO1
    output logic G        // Output asserted in GO2
);

    /* ---------------------------------------------------------
     *  State encoding
     * --------------------------------------------------------- */
    typedef enum logic [1:0] {
        IDLE = 2'b00,
        GO1  = 2'b01,
        GO2  = 2'b10
    } state_t;

    /* ---------------------------------------------------------
     *  Synchronous state register (reset is asynchronous)
     * --------------------------------------------------------- */
    always_ff @(posedge clk or posedge reset) begin
        if (reset)
            state <= IDLE;          // Return to IDLE on reset
        else
            state <= next_state;    // Update state on clock edge
    end

    /* ---------------------------------------------------------
     *  Next-state logic (combinational)
     * --------------------------------------------------------- */
    always_comb begin
        next_state = state;         // Default: hold current state
        case (state)
            IDLE: begin
                if (start)
                    next_state = GO1;    // Start -> GO1
            end

            GO1: next_state = GO2;    // GO1 -> GO2

            GO2: next_state = IDLE;   // GO2 -> IDLE

            default: next_state = IDLE;  // Safety
        endcase
    end

    /* ---------------------------------------------------------
     *  Output logic (combinational)
     * --------------------------------------------------------- */
    always_comb begin
        F = 1'b0;   // Default
        G = 1'b0;   // Default
        case (state)
            GO1: F = 1'b1;   // F asserted in GO1
            GO2: G = 1'b1;   // G asserted in GO2
            default: ;       // IDLE – both outputs remain 0
        endcase
    end

endmodule
```

How it works

1. **Reset** (`reset=1`) forces the FSM to `IDLE` .
2. While in **IDLE**, a high `start` pulse moves the FSM to **GO1** and sets `F=1` .
3. In **GO1**, the FSM automatically goes to **GO2** on the next clock and sets `G=1` .
4. In **GO2**, it goes back to **IDLE** on the next clock.
5. Both `F` and `G` default to 0 in all other cases.

The module is fully synthesizable, uses only standard constructs, and can be instantiated in any SystemVerilog design.

Look, UVM!

1. Package – common types

systemverilog

```
// file: fsm_pkg.sv
package fsm_pkg;
    import uvm_pkg::*;
    `include "uvm_macros.svh"

    // Sequence item that the driver will push into the DUT
    class fsm_seq_item extends uvm_sequence_item;
        rand bit start;          // stimulus to the DUT
        bit  F_exp;              // expected value of output F
        bit  G_exp;              // expected value of output G

        `uvm_object_utils(fsm_seq_item)

        function new(string name = "fsm_seq_item");
            super.new(name);
        endfunction
    endclass : fsm_seq_item

    // Sequence - just a simple generator
    class fsm_sequence extends uvm_sequence #(fsm_seq_item);
        `uvm_object_utils(fsm_sequence)

        function new(string name = "fsm_sequence");
            super.new(name);
```

2. Environment – agent, driver & monitor

3. Scoreboard

4. Environment – glue all the pieces together

5. Test

6. Top-level testbench (instantiates DUT and UVM)

How it works

1. **Test** (`fsm_test`) creates the environment and starts a sequence that toggles `s`
2. The **driver** applies `start` to the DUT, waits for the next clock edge, records wh
   expected values back to the scoreboard via `add_expected()` .
3. The **monitor** simply raises an event whenever F or G changes – in this simple ca

Summarization

spi_clgen.v spi_defines.v spi_shift.v spi_top.v

You are an expert hardware engineer. Analyze the spi_clgen, spi_defines, spi_shift, and spi_top modules. Create a list of properties describing the functionality of the spi_top module.

Below is a concise "properties" list that captures what the `spi_top` wrapper brings together, based on the public interfaces and typical design intent of an SPI controller implemented with the `spi_defines`, `spi_shift`, and `spi_clgen` sub-modules.

Property	Description
Wishbone-compliant interface	`wb_clk_i`, `wb_rst_i`, `wb_adr_i`, `wb_dat_i`, `wb_sel_i`, `wb_we_i`, `wb_stb_i`, `wb_cyc_i` are the standard Wishbone signals. The module returns `wb_dat_o`, `wb_ack_o`, `wb_err_o`, and `wb_int_o`. This allows the controller to be addressed, read from, and written to by any wishbone master (CPU, DMA, etc.).
SPI I/O pins	External SPI pins are exposed as: `ss_pad_o` (slave-select), `sclk_pad_o` (serial clock), `mosi_pad_o` (master-out-to-slave), and `miso_pad_i` (master-in-from-slave). These correspond to the physical pins on an FPGA or ASIC.
Parameterised clock period	The module has a parameter `Tp = 1;`. While not used directly in the snippet, it is typically leveraged by timing constraints or internal calculations for generating the SPI bit-rate.
SPI clock generation	Internally, `spi_clgen` produces the SCLK (`sclk_pad_o`) from `wb_clk_i` based on a programmable divider (`divider`).

Scripting a chat bot

```
import ollama

# Choose your model
model_name = "llama3"  # change if you have another ollama model

print("=== Simple Ollama Chatbot ===")
print("Type 'quit' to exit.\n")

while True:
    user_input = input("You: ")
    if user_input.lower() == "quit":
        break

    # Generate response from Ollama
    response = ollama.generate(
        model=model_name,
        prompt=user_input,
        options={"temperature": 0.7, "max_tokens": 4000}
    )

    print("Bot:", response["response"])
```

1,5

1. Use framework
2. Specify LLM
3. Get prompt
4. Ask LLM
5. Print response

Equipping your chat bot

RAG

```
# ----------- LOAD DOCUMENTS -----------
def load_documents_from_directory(directory):
    ...

# ----------- SPLIT DOCUMENTS ----------
text_splitter = RecursiveCharacterTextS
0)
docs = text_splitter.split_documents(do
print(f"🛠 Split into {len(docs)} chunks

# ----------- EMBEDDINGS & VECTORSTORE
embeddings = OllamaEmbeddings(model=emb
faiss_path = "faiss_index"
...

# ----------- MEMORY -----------
def load_memory(path):
    ...
def save_memory(path, messages):
    ...

previous_messages = load_memory(memory_
memory = ConversationBufferMemory(memor
e)
memory.chat_memory.messages = previous_
```

Memory

```
# ----------- LLM -----------
llm = OllamaLLM(model=model_name)

# ----------- TOOLS -----------
def search_docs(query: str):
    """Search the local knowledg
    qa_chain = RetrievalQA.from_
        llm=llm,
        retriever=retriever,
        return_source_documents=
        chain_type_kwargs={"outpu
r memory
    )
    result = qa_chain.invoke(quer
    sources = [doc.metadata.get('
ocuments"]]
    return f"{result['result']}\n

tools = [
    Tool(
        name="KnowledgeBaseSearch
        func=search_docs,
        description="Use this to
"
    ),
```

Tools

```
# ----------- AGENT -----------
agent = initialize_agent(
    tools,
    llm,
    agent=AgentType.ZERO_SHOT_REACT_DESCRIPTION,
    memory=memory,
    handle_parsing_errors=True,
    verbose=True
)

# ----------- INTERFACE LOOP -----------
print("🤖 Ready! Ask something (type 'exit' to quit):")

while True:
    query = input("You: ")
    if query.lower() in ["exit", "quit"]:
        print("👋 Exiting. Saving memory...")
        save_memory(memory_path, memory.chat_memory.messages)
        break

    response = agent.invoke(query)
    print("AI:", response)
    save_memory(memory_path, memory.chat_memory.messages)
```

Agentic flow

```python
# --- Agentic Flow ---
def agentic_flow(prompt):
    print(f"\nPrompt: {prompt}\n")

    # Step 1: Generate initial response
    response = llm_generate(prompt)
    print(f"Initial Response: {response}\n"

    # Step 2: Critique the response
    critique = llm_critique(response)
    print(f"Critique: {critique}\n")

    # Step 3: Generate refined response
    refined_response = llm_refine(prompt, c
    print(f"Refined Response: {refined_respo

    return refined_response

# --- Main program ---
if __name__ == "__main__":
    prompt_text = input("Enter your prompt
    final_output = agentic_flow(prompt_tex
```

1
2
3
4

Enter your prompt: Write a funny joke

Prompt: Write a funny joke

Initial Response: I asked the librarian if the library had books on paranoia. She whispered, "They're right behind you."

Critique: **Critique of the response**

> *"I asked the librarian if the library had books on paranoia. She whispered, 'They're right behind you.'"*

1. Clarity

Aspect	Asses
Language	Very c ambig
Narrative flow	Seami natura
Tone	Light,
Potential confusion	None. clear,

Verdict: The response is cr

2. Completeness

Refined Response: **New joke**

> "I asked the librarian for a book on reverse psychology.
> She leaned in and whispered, 'Just stay away from it.'"

Why this one works better

Element	Why it's an improvement
Clear setup	The request for a "book on reverse psychology" immediately signals that the answer will be a subversion.
Tighter punchline	The librarian's reply is a literal echo of reverse-psychology: by telling the person to stay away, she nudges them toward the book.
Visual humor	The image of a librarian whispering "Just stay away from it" in a library is vivid and unexpected.
Universal	Anyone familiar with the phrase "reverse psychology" (or the idea of a paradoxical

A **generative adversarial network (GAN)** is where a *generator* is critiqued by a *discriminator* to make a better response.

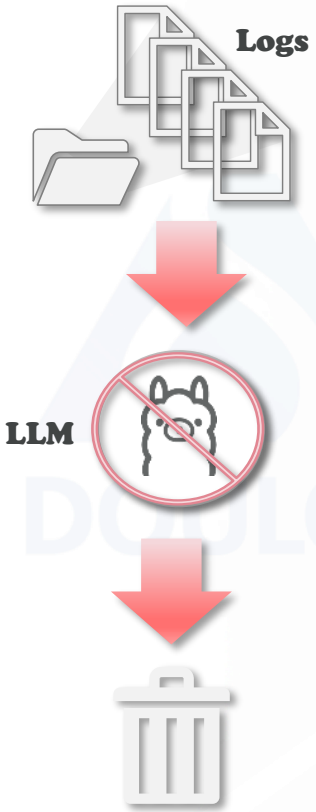
An engineering example

Log file triage

```
# UVM_ERROR verilog_src/uvm-1.1c/src/seq/uvm_sequencer_base.svh(1236) @ 7096: uvm_test_top.m_env.m_wb_agent.m_wb_seqr [SEQFINERR] Parent sequence
'uvm_test_top.m_env.m_v_sequencer.vseq.dut_config' should not finish before locks from itself and descedent sequences are removed.  The lock held by the child sequence
'uvm_test_top.m_env.m_v_sequencer.vseq.dut_config' is being removed.
# UVM_INFO @ 7096: uvm_test_top.m_env.m_wb_agent.m_wb_drv [WB_DRIVER] Performing a RX ...
# UVM_INFO @ 7115: uvm_test_top.m_env.m_wb_agent.m_wb_mon [wb_monitor] Detected a WB transaction ...
# UVM_INFO @ 7126: uvm_test_top.m_env.m_wb_agent.m_wb_drv [WB_DRIVER] wb_read task: addr = 00000004, data = 00000000
# UVM_INFO @ 7127: uvm_test_top.m_env.m_wb_agent.m_wb_seqr@@vseq.wb_read_rx_regs.wb_read [wb_read] Generated sequence:
# UVM_INFO @ 7127: uvm_test_top.m_env.m_wb_agent.m_wb_drv [WB_DRIVER] Performing a RX ...
# UVM_INFO @ 7135: uvm_test_top.m_env.m_wb_agent.m_wb_mon [wb_monitor] Sending transaction to the scoreboard
# ----------------------------------------------------------------------------------------------
# Name                          Type       Size  Value
# ----------------------------------------------------------------------------------------------
# uvm_sequence_item             wb_trans   -     @1285
#   addr                        integral   5     'h4
#   data                        integral   32    'h0
#   kind                        trans_t    32    RX
# ----------------------------------------------------------------------------------------------
# UVM_ERROR @ 7135: uvm_test_top.m_env.m_wb_spi_sb [wb_spi_sb] SPI error callback. Error selecting the bit to twiddle!"
# UVM_INFO @ 7145: uvm_test_top.m_env.m_wb_agent.m_wb_mon [wb_monitor] Detected a WB transaction ...
# UVM_INFO @ 7156: uvm_test_top.m_env.m_wb_agent.m_wb_drv [WB_DRIVER] wb_read task: addr = 00000000, data = 0000050e
# UVM_INFO @ 7157: uvm_test_top.m_env.m_wb_agent.m_wb_seqr@@vseq.wb_read_rx_regs.wb_read [wb_read] Generated sequence:
# UVM_INFO @ 7157: uvm_test_top.m_env.m_wb_agent.m_wb_drv [WB_DRIVER] Performing a RX ...
# UVM_INFO @ 7165: uvm_test_top.m_env.m_wb_agent.m_wb_mon [wb_monitor] Sending transaction to the scoreboard
# ----------------------------------------------------------------------------------------------
# Name                          Type       Size  Value
# ----------------------------------------------------------------------------------------------
# uvm_sequence_item             wb_trans   -     @1315
#   addr                        integral   5     'h0
#   data                        integral   32    'h50e
#   kind                        trans_t    32    RX
# ----------------------------------------------------------------------------------------------
# UVM_ERROR @ 7165: uvm_test_top.m_env.m_wb_spi_sb [wb_spi_sb] WB write to an invalid address!  Address = 16647
# UVM_INFO @ 7175: uvm_test_top.m_env.m_wb_agent.m_wb_mon [wb_monitor] Detected a WB transaction ...
# UVM_INFO @ 7186: uvm_test_top.m_env.m_wb_agent.m_wb_drv [WB_DRIVER] wb_read task: addr = 0000000c, data = 00000000
```

Can we use AI to group common failure modes?

AI limitations



LLM limitations:

- Bad at scale (too many log files)
- Bad at long noisy text
- Distracted by random, test-specific values
- RAG helps retrieval, but ... messy raw data is messy retrieval
- Better at *semantic* than *symbolic* analysis

Solution

1. Pre-process with a script
   - *Filter random, test-specific values*
   - ***Extract error signatures***
   - *Normalize into a compact form:*

```
[FAIL] parity_mismatch in module A
[FAIL] addr_decode_error in module B
[FAIL] timeout waiting for handshake in module C
```

2. Cluster and categorize messages
   - *Use **embeddings** to group similar errors*
   - *Use **sentence level** instead of **word level***
   - *Return signature groupings*
3. Summarize with an LLM
   - ***Summarize** the signature groupings in the desired format*
4. Generate a report of results
   - *Simple **reporting script code***

Results

```python
def export_report(clustered, summaries, out_csv="failure_report.csv"
):
    rows = []
    for label, sigs in clustered.items():
        summary = summaries.get(label, "Uncategorized")
        rows.append({
            "Cluster": label,
            "Summary": summary,
            "Num_Failures": len(sigs),
            "Examples": "; ".join(set(sigs[:3]))
        })
    df = pd.DataFrame(rows)
    df.to_csv(out_csv, index=False)
    return df

if __name__ == "__main__":
    logs_dir = "./workflow/log_parsing"
    sigs = collect_signatures_from_logs(logs_dir)
    clustered = cluster_signatures(sigs)
    summaries = summarize_clusters(clustered)
    report = export_report(clustered, summaries)
    print(report)
                                                172,1         Bot
```

Results OK, but …
easier (and faster)
with a regular script

Output

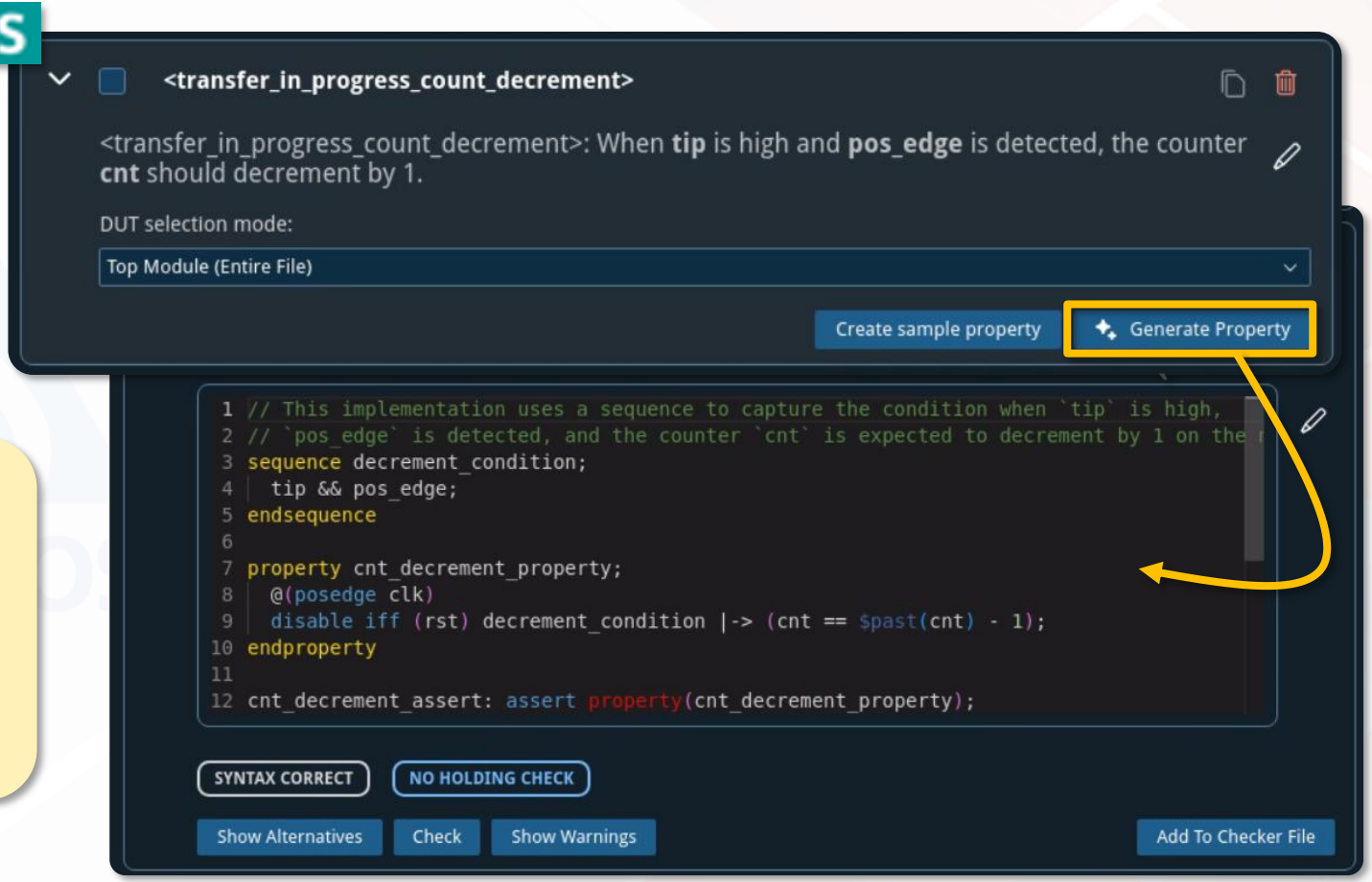
```
[cluster_signatures] device = cuda
  tried eps=0.200 -> clusters=4, labels_set={np.int64(0), np.int64(1), np.int64(2), np.int64(3)}
  tried eps=0.300 -> clusters=2, labels_set={np.int64(0), np.int64(1)}
  tried eps=0.400 -> clusters=2, labels_set={np.int64(0), np.int64(1)}
  tried eps=0.500 -> clusters=1, labels_set={np.int64(0)}
  tried eps=0.600 -> clusters=1, labels_set={np.int64(0)}
✅ Chosen clusters: 4 (eps=0.2)
   Cluster  ...                                  Examples
0        0  ...  UVM_ERROR @ <NUM>: uvm_test_top.m_env.m_wb_spi...
1        1  ...  UVM_ERROR verilog_src/uvm-<NUM>.<NUM>c/src/seq...
2        2  ...  UVM_ERROR :  <NUM>; UVM_FATAL :  <NUM>; UV...
3        3  ...  UVM_ERROR @ <NUM>: uvm_test_top.m_env.m_wb_spi...

[4 rows x 4 columns]
```

CSV

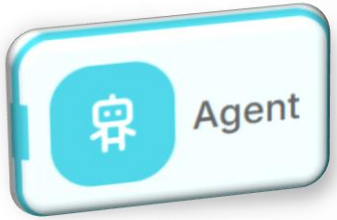
Cluster	Num_Failures	Examples
0	70	UVM_ERROR @ <NUM>: uvm_test_top.m_env.m_wb_spi_sb [wb_spi_sb] WB write to an invalid address! Address = <NUM>
1	20	UVM_ERROR verilog_src/uvm-<NUM>.<NUM>c/src/seq/uvm_sequencer_base.svh(<NUM>) @ <NUM>: uvm_test_top.m_env.m_w
2	60	UVM_ERROR : <NUM>; UVM_FATAL : <NUM>; UVM_WARNING : <NUM>
3	10	UVM_ERROR @ <NUM>: uvm_test_top.m_env.m_wb_spi_sb [wb_spi_sb] SPI error callback. Error selecting the bit to twiddle!"

EDA tools

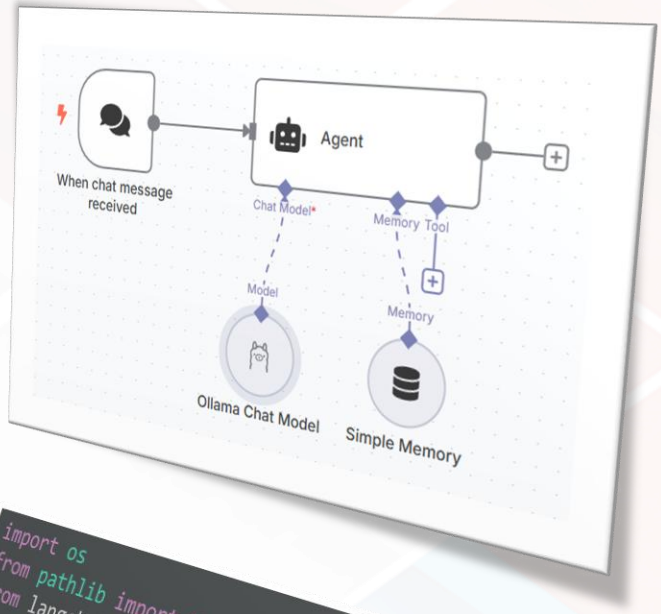


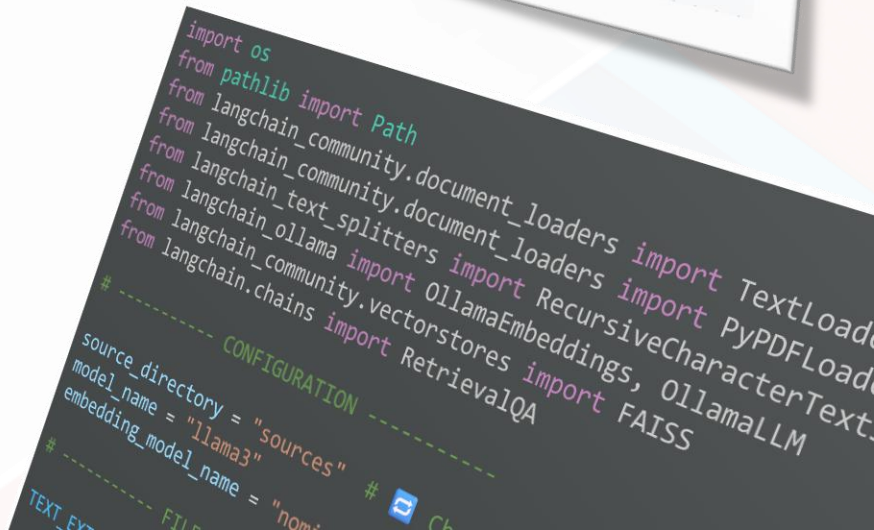
EDA tools call LLMs to help with tasks

Siemens EDA Questa One SFV. Screenshots used by permission.



AI or not AI?





"Chat, summarize AI"

✅ What it's good at …	⚠️ What it's bad at …
✅ Understanding language	⚠️ Understanding context deeply
✅ Summarization	⚠️ Common sense reasoning
✅ Generating content quickly from its training	⚠️ Creativity and originality
✅ Automating repetitive tasks	

Recommendations with AI

❌ Avoid …	ℹ️ FYI …
❌ Blind trust in its outputs	⚡ More resources → better performance
❌ AI on non-GPUs & small amounts of GPU memory	🧮 Reasoning models mainly help with math & coding
❌ Lack of human oversight	🚀 Quantized models may give faster results
❌ Non-private or insecure AI	🖼️ Use Vision Large Language Model (VLM) for text + images
❌ Over-reliance on AI	📄 LLMs are not great for parsing

Practical uses of AI

🔧 Task	💡 How AI Can Help
📝 Automate code generation	Generate RTL, testbenches, and assertions from specs.
💬 Chat-driven design assistant	Convert design intents into RTL and testbenches via chat.
🕵️ Verification automation	Summarize simulation results and flag anomalies.
🔍 Design review and refactoring	Detect issues and suggest improvements in HDL code.
🔄 Continuous integration support	Auto-comment or review on check-in.

References

1. https://poloclub.github.io/transformer-explainer/
2. Vaswani, Ashish; Noam M. Shazeer; et ali. "Attention is All you Need." *Neural Information Processing Systems*, 2017.
3. Kumar, Aman and Deepak Narayan Gadde. Generative AI Augmented Induction-based Formal Verification. *37th IEEE International System-on-Chip Conference*, Sep. 16-19, Dresden Germany, 2024.
4. Truong, Andy; Daniel Helström; Harry Duque; and Lars Viklund. "Clustering and Classification of UVM Test Failures Using Machine Learning Techniques." *DVCon Europe*, 2018.
5. Ismail, K.A.; Ghany, M.A.A.E. Survey on Machine Learning Algorithms Enhancing the Functional Verification Process. *Electronics* 2021, *10*, 2688. https://doi.org/10.3390/electronics10212688
6. Thakur, Shailja; Baleegh Ahmad; et ali. "Benchmarking Large Language Models for Automated Verilog RTL Code Generation." https://arxiv.org/abs/2212.11140. 2022.

DOULOS

Delivering KnowHow www.doulos.com

SoC Design & Verification	» SystemVerilog » UVM » Formal » SystemC » TLM-2.0
FPGA & Hardware Design	» VHDL » Verilog » SystemVerilog » Tcl » AMD
Embedded Software & Arm	» Emb C/C++ » Emb Linux » Yocto » RTOS » Security » Android » Arm » Rust » Zephyr
Python, AI & Machine Learning	» Python » Edge AI » Deep Learning

SystemVerilog OUT OF THE BOX · UVM · SYSTEMC · yocto PROJECT PARTICIPANT · Zephyr · SECURE EMBEDDED · arm APPROVED TRAINING PARTNER · AI & MACHINE LEARNING

New Design and Verification courses

Advanced Formal Verification

- equips you to tackle complex verification challenges by taking full advantage of formal verification in your engineering projects.
- forms a complete learning path with **Essential Formal Verification** course, which gives you a solid, practical grounding in formal verification.

SystemVerilog for New Designers: Self-Paced course

- get project-ready for FPGA or ASIC design, including RTL synthesis, block-level test benches, and FPGA design flows.
- high-quality content developed by expert instructors - now in self-paced format.

IC Verification with Python and cocotb

Coming Soon!

- learn cocotb principles, constructing different aspects of a verification environment using cocotb and Verification strategies and tactics.

Questions?