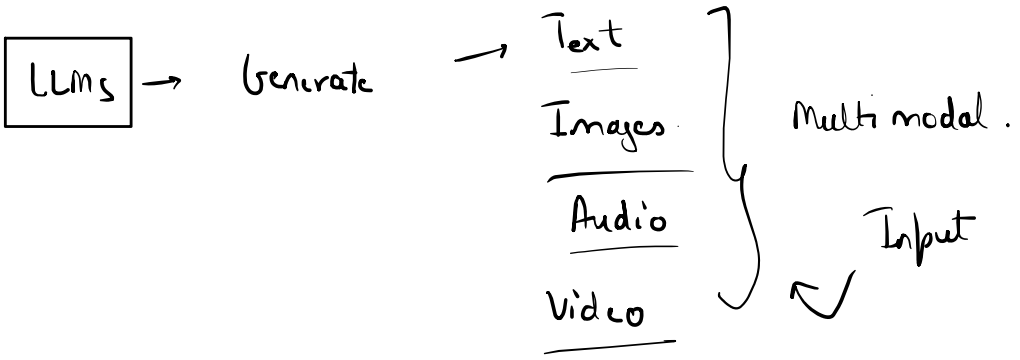


Image captioning , Video → Transcript

Audio → Video

2) Memory → Context aware

↓

RAG → Documents → QA

3) Reasoning → Cognitive ability → Reactive

↓

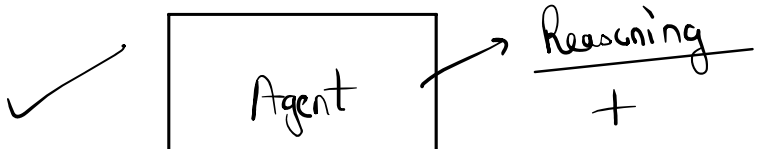
Prompt → Response

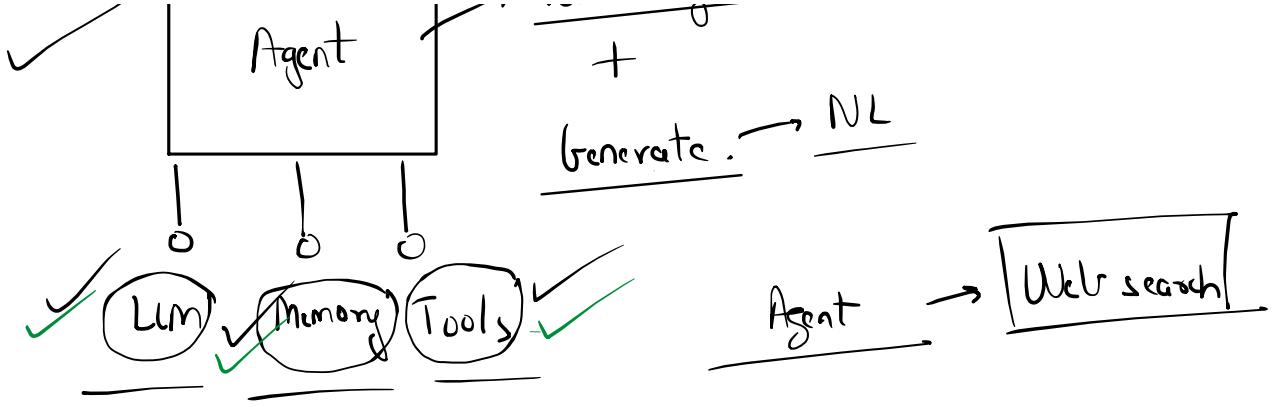
↓

Proactive

Complex task → multiple simple tasks

4) Tools → Outer world → Do task / take action.





Agno → Python Framework

↓

Declarative

1) Brain → LLM

2) Memory → Context Aware

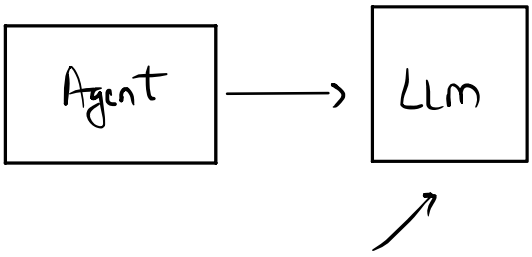
3) Tools.

Agent :

Brain → LLM

1) Cognitive ability

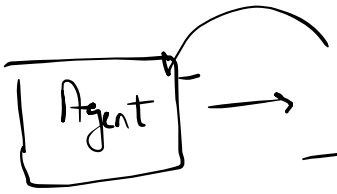
2) Generate new data / text



Respond

500 word report → Gen AI

env → Agno install + dependencies → UV



Prompt → Response

Memory → chat history / conversational history

Agno → Session ← session_id

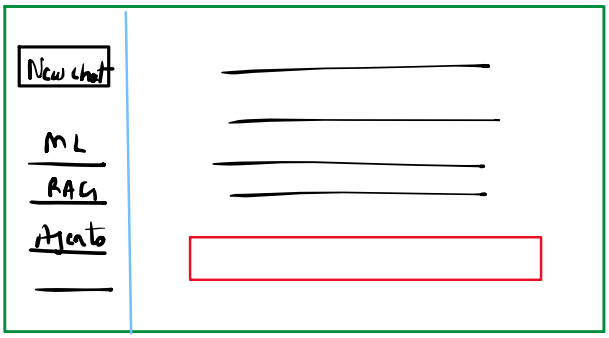
100 word summary → topic ? ← ∝

Agents → Per execution basis

input → Agent → output

LR

Agent ↑↓ LLM



Separate conversions → session

unique session id

A single user can hold multiple conversations.

session → Chats → continue, pause, read. → Memory

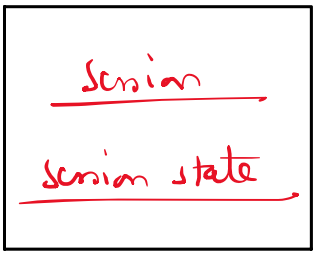
ML

Memory

Short term Memory

long term Memory

Session



Memory → Session wise memory

Memory → User wise memory

Memory

Linear Regression →

Logistic Regression →

sessions

I like to discuss about DS

Memory

1) Conversation / chat history → session.

2) Session related extra info → session state

3) User related memory → Memory / User memory

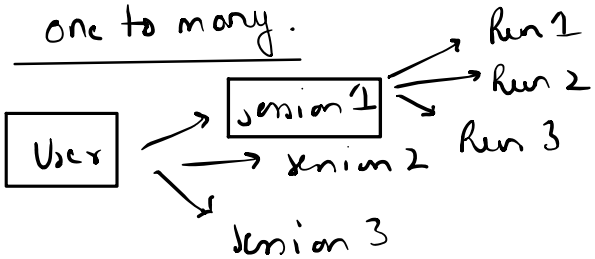
User → User

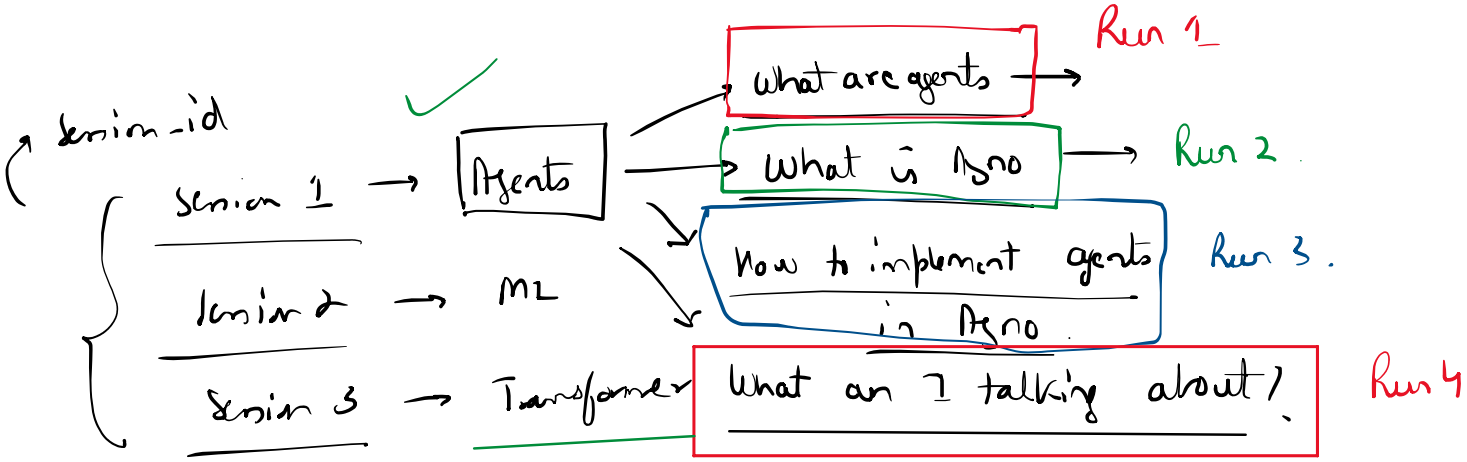
Session → One single conversation.

Runs → Execution of the agent

Input → Agent → Output

One to many.





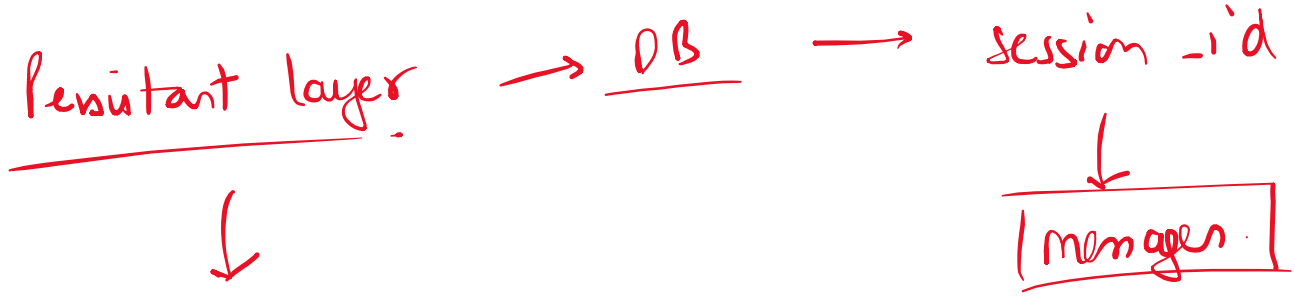
users → unique_id.

sessions → session_id.

runs → run_id.

session_id.

Storage



Storage → Backend.

Memory → frontend.

Agent state → Extra info

↓

Structured format → Accessed by all your runs.

Session → Chat history

→ dict Key : value

→ Working memory

Session state = { username : Nimanshu,
age : 34, }

Information manage across runs → to do list, shopping list, wishlist

CRUD → Update, delete

Add items

Remove items

Agno

What is my name?

Session state

↓

name

↓

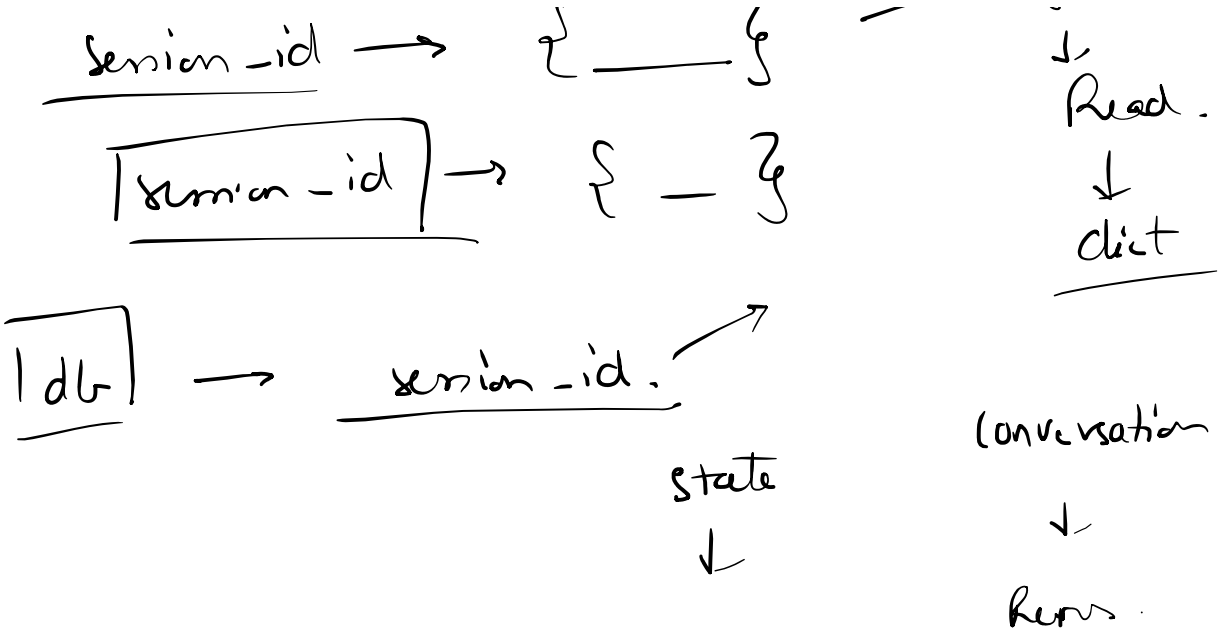
Nimanshu

Add task → Record Video

Recording completed. All done.

Session_id → { ____ } → agent

↓


session_id → { — }
session_id → { — }
db → session_id.
state
Read.
click
(conversation
Rerr

To do actions / perform tasks.

python functions → Agent gets the capability to call these functions.

→ Tool calls.

def func(a)(b): ← Agent

Agent → 2 functions.

1) Agent decides on which tool to call based on the user's query

2) What input values / arguments to pass to the tool.

Argument type.

def add_numbers (a, b): → "3", "4"

return a+b

add_numbers (3,4)

Agent → LLM → Brain → Generative.

→ Cognitive → Planning

→ Reasoning

Planning → Complex goal → Sub tasks

Reasoning → Sub task → achieve

for a given query -

✓ 1) which tool to call ?

✓ 2) If tool is selected, what should be its inputs ?

tools → docstring LLM → tools access. tools → task.

type hinting def add_item (session_state : dict, item: str) → str

@tool → langchain "This tool adds items to our list"

Tools = [tool 1, tool 2, tool 3] → custom tools → python functions

↓ Pre defined tools classes.

instructions = You have access to 3 tools.

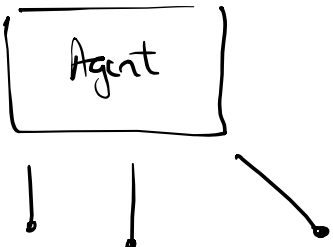
↓ System instructions

1) Tool 1 →
2) Tool 2 →
3) Tool 3 →

Knowledge → Agentic RAG

Components of Agent

Memory → Conversational history
Maintain the context of
the conversation.



a) Generation
b) Reasoning

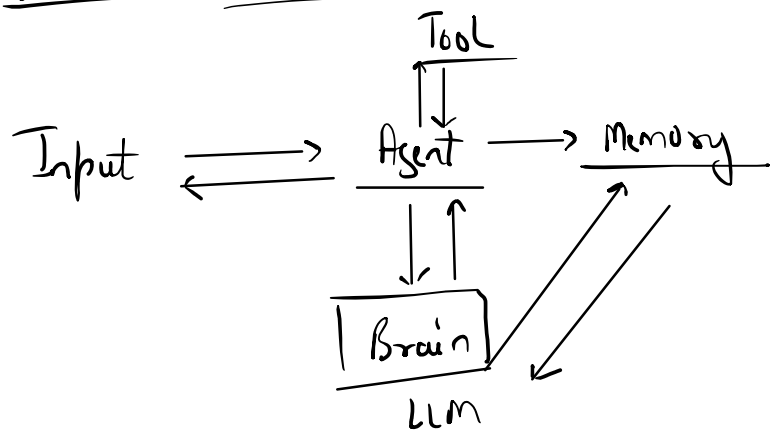
Brain
LLMs
↓
Reason
Cognitive

Memory
↓
i) Chat history (session)
ii) State (Agent state)

Tool → Perform task / Take actions

Tool → Connect to the outer world → Web search
↓
Context

↓
functions / API calls.



Knowledge → Information.

Agent → Response.

Knowledge → Information

Asia Cup ✗

LLM
(knowledge cut off date)
Nov 2024

I don't have info about this event.

Hallucinate → Asia Cup 2025 was won by Australia ✗

Agent → superpower Informational superpower

LLM → Recent events

↓ Knowledge Knowledge

↓
up to date info

Customer support Agent → Info

Status

Knowledge → RAG → Retrieval Augmented Generation

Agentic RAG

Knowledge → Agentic RAG.

RAG → Retrieval augmented generation.

Agent

LLMs

enhance the ability.

Retrieval

To fetch

100 items.

to fetch → similarity.

similar to the user input

RAG.

Chatbot → Company → responses.

Access to my company documentation.

What is the leave policy?

LLM → generic information

LLM → similarity search

What is the leave policy?

Retrieve

LLM.

3 paragraphs.

2 inputs?

Output

input query : What is the leave policy?

additional context : 3 paragraphs

response → The company policy regarding leaves is -

1)

specific for my organisation.

Retrival augmented generation → RAG.

more specific.

RAG → Generation → LLM → input + context → Response.

Input query → Retrival → Output

similar items / additional context

Query + context

RAG components :- (Work in sequential order)

Pipeline.

1) Readers / Loaders.

4) Storage.

2) Chunking

3) Embedding

5) Retrieval.

1. Readers/Loaders and Sources

Sources → Sources of documents

- Documents/files
  1) Markdown
  2) PDF
  3) doc
  4) PPT
  5) Txt
- Web
- CSV files
  - Tables.
  - 1 Row = 1 data point
- Topic
  - Wikipedia
  - Arxiv
- Databases
  - SQL, N.SQL

Readers / Loaders → load the document and parse it.

Format: Text, images, graphs, tables

Hard drive → .pdf

PDF Reader → show the document in proper format

2. Chunking

Chunking → To break into chunks.

500 pages.

Page wise.

Chapter wise

Agno →

1) fixed Chunking → 2000 words.

2) Semantic chunking → Similar text.

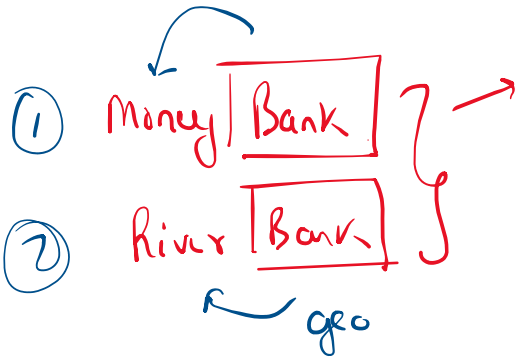
3) Document chunking → Document format

4) Recursive text chunking →

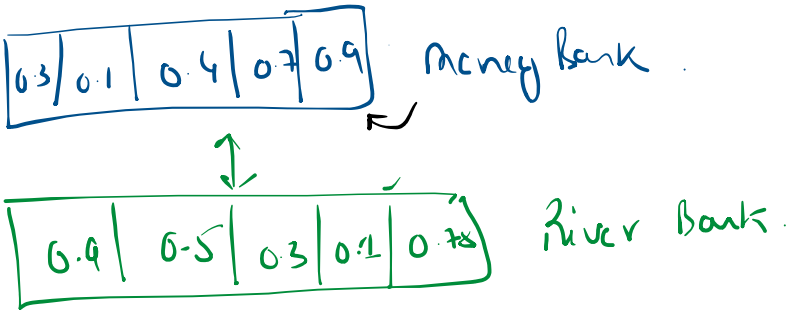
Embedding → Conversion of your text into vector.

Encoding → Categories → Integers

Sets of numbers



Context of the text



① Document load.

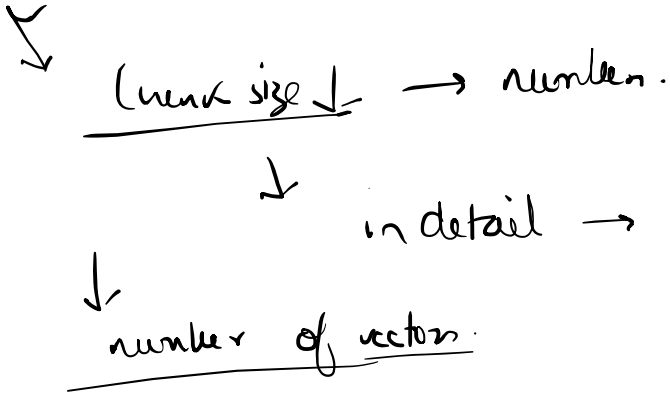
② Broke the document into multiple sub documents

③ Number convert → LLM → embedding

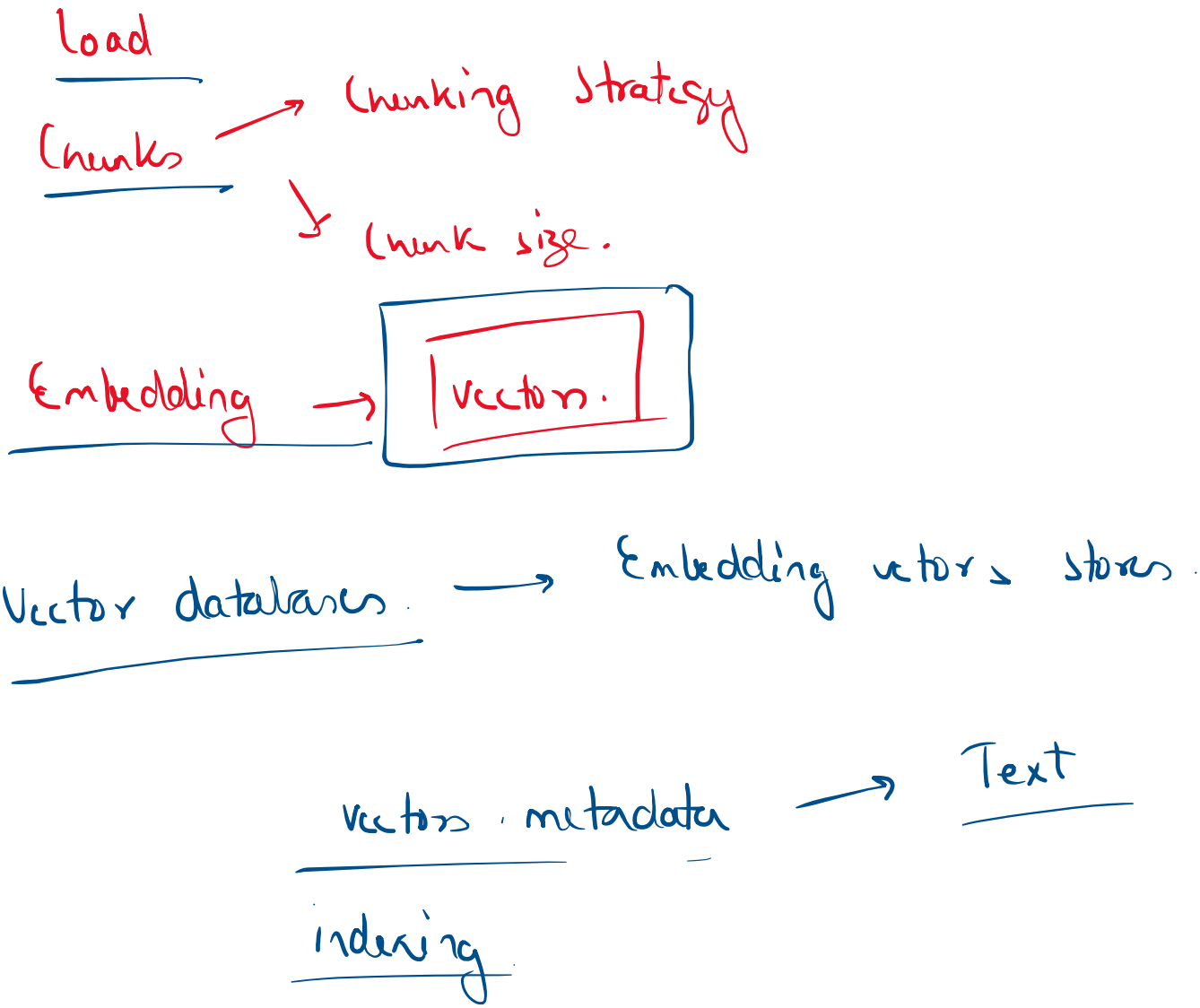
→ cohere

Chunk-size → Chunk size ↑ → context → number.

large context → ↑ number.


Chunk size ↓ → number.
in detail →
number of vectors.

4. Storage


load
Chunks
Chunking Strategy
Chunk size.
Embedding
Vector.
Vector databases
Embedding vectors stores
Vectors + metadata
Text
Indexing

Retrieval → Vector.

↓

Vector database

RAG.

leave policy → [General] → wrong ∝ → Super accurate.

↓

knowledge up to date. relevant

company specific

Documentation.

↓

[3 paragraphs]

— Input.

— Output

— Processing

Retrieval pipeline:-

Input → What is the leave policy of my company?

↓

embedding component → numbers.

5 numbers.

input vector [0.1 | 0.3 | 0.2 | 0.9 | 0.6]

Semantic context

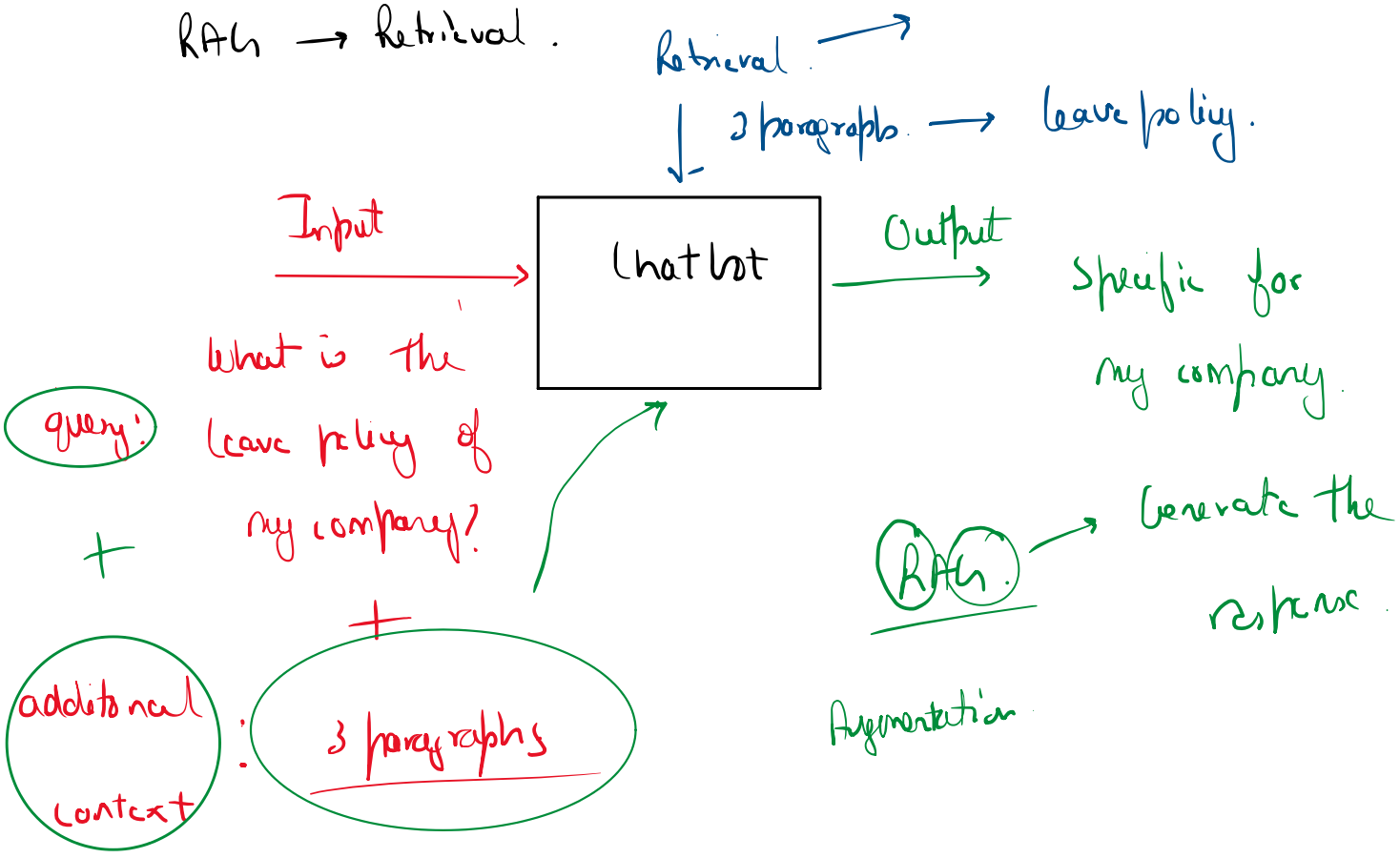
↓

similarity search → Cosine similarity

Top 3 matches → return → 3 paragraphs.

similar to my input.

leave policy

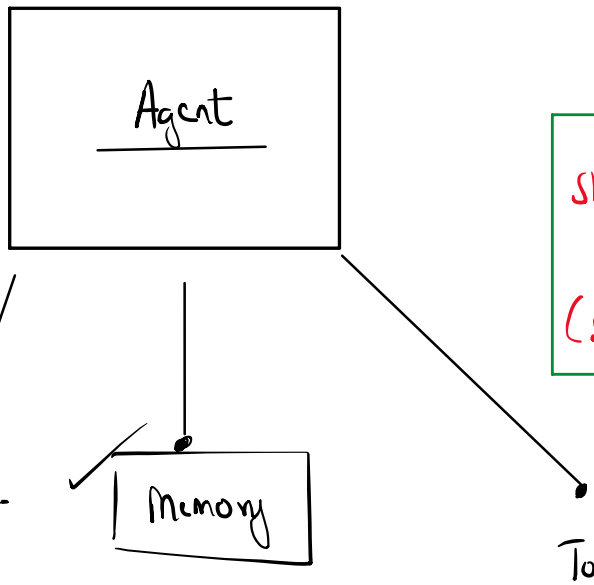


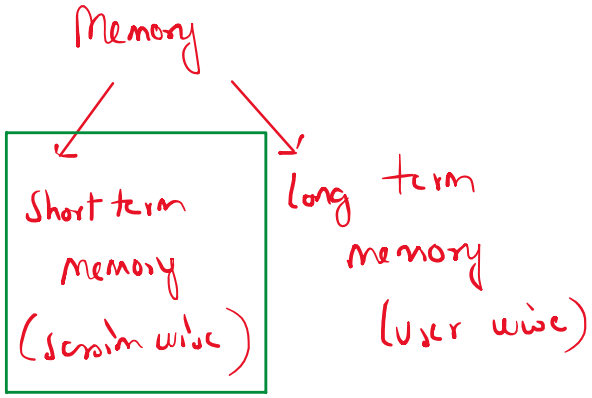
What is Agentic RAG

Knowledge Pipeline in Agno

Code

✓ Knowledge → Superpower for your agents





✓ Brain / LLMs

1) Generate
2) Think/Reason

Cognitive abilities

✓ Memory

1) Context
2) Retain context

Tools

Perform tasks / Do actions.

Functions.

Which tool to call?

Which inputs to give to a tool?

Memory → short term memory

1) Conversational history / Chat history → Sessions
2) Agent state / Session state → Working memory that can be accessed across your runs.

{ shopping list : [ ] } → CRUD

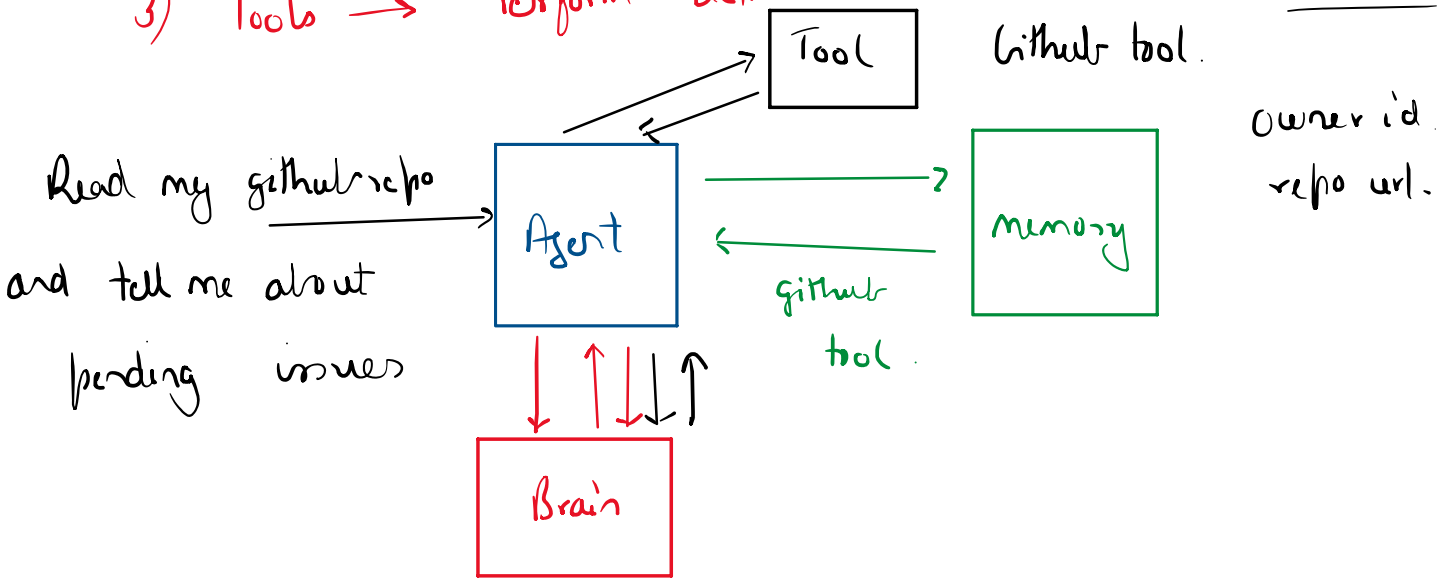
Agent → 3 components

Agent

1) Brain → Cognitive / thinking ability

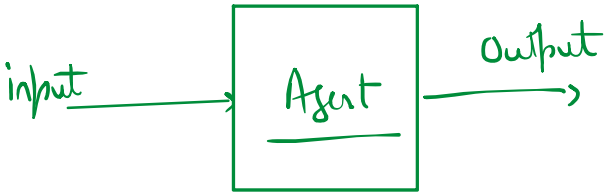
2) Memory → Retains context / meaning across the entire conversation

3) Tools → Perform actions.



These are some of the pending issues for the repo.

1) Issue name Issue Id → Description



4th component :-

Knowledge → Information / Data. → Relevant info

Agent → LLM Knowledge cut off date.

Dec 2024

Who won Asia Cup 2025 ? → I don't have info about this event

↓

Hallucinate → Was won by Bangladesh.

Proper information

Relevant Info and Up to date information.

Knowledge → RAG

Retrieval augmented generation.

Agentic RAG

Flow :-

RAG

Diff components in RAG

Agentic RAG

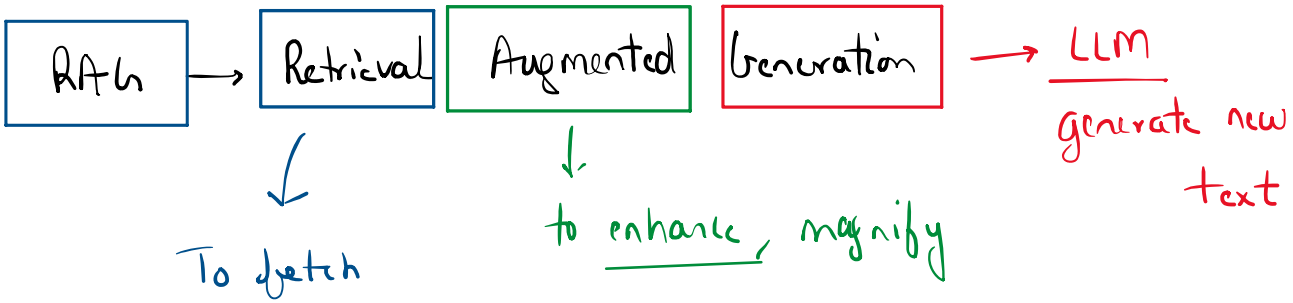
Diff b/w Agentic RAG vs Traditional RAG

Knowledge → Agentic RAG

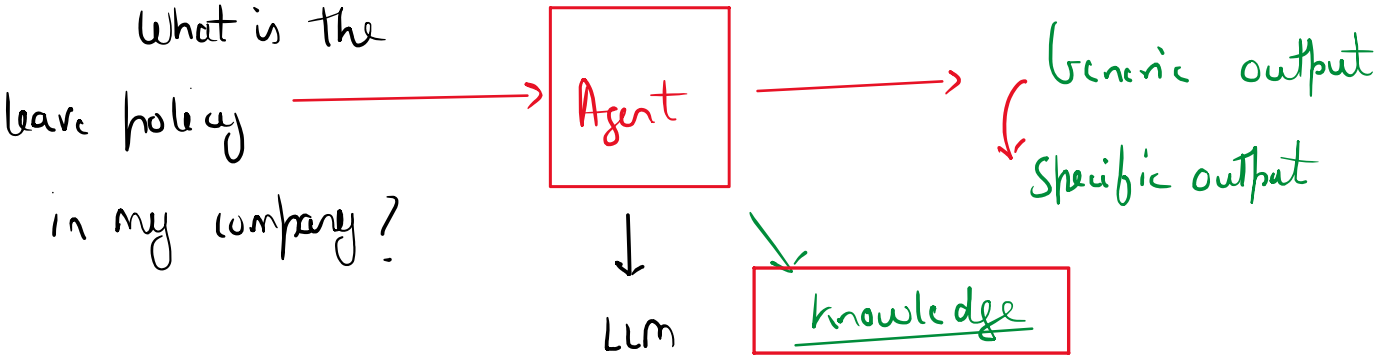
↓

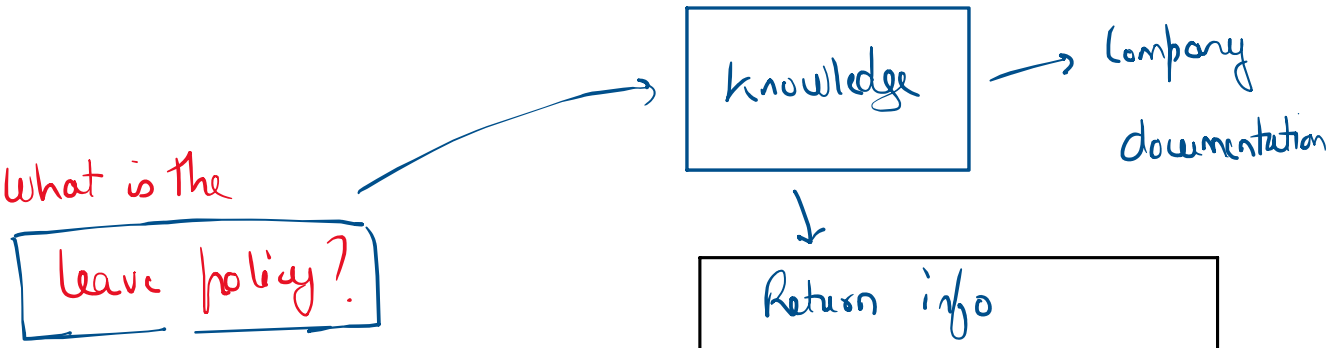
Code

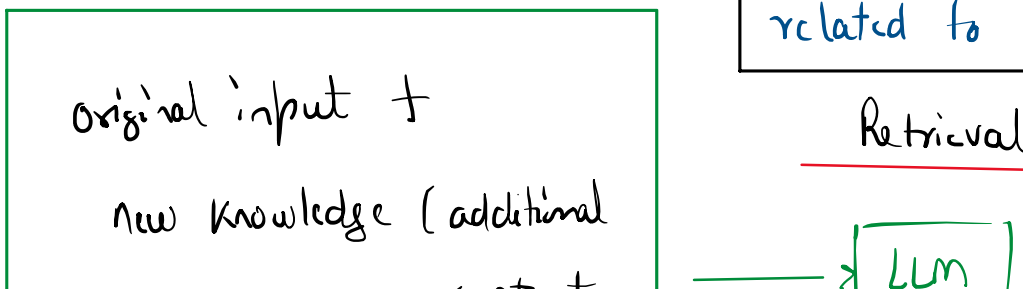
What is RAG?

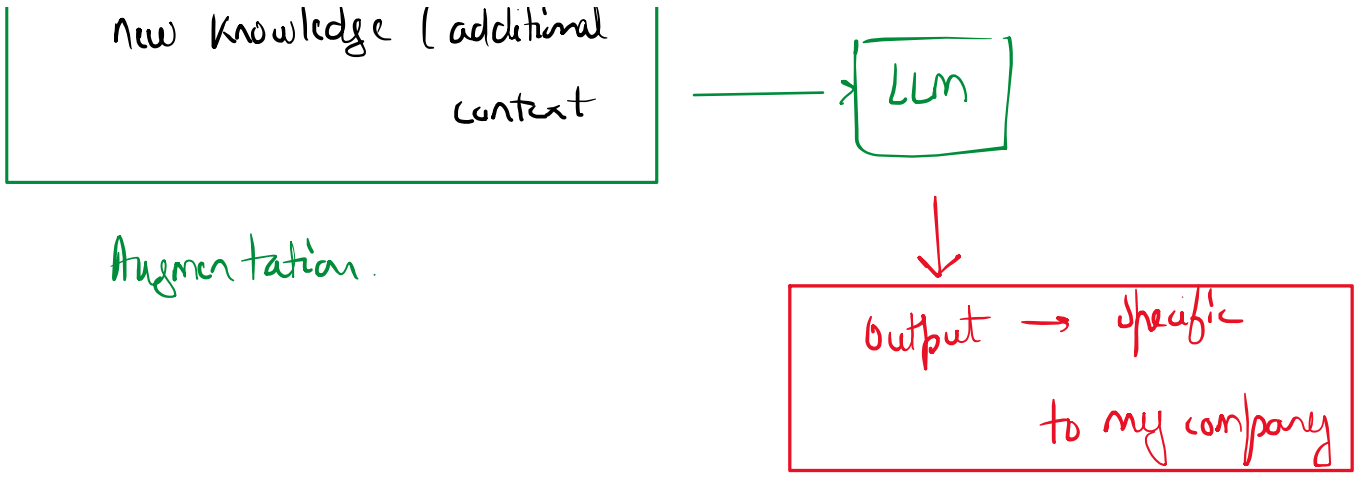


fetch relevant information









Retrain → Fine tune

Context → generic → specific

↓

Upto date

5 main components of RAG :-

1) Loaders / Readers.
2) Chunking strategy.
3) Embeddings
4) knowledge base / storage.
5) Retrieval.

Components → Which help in loading the document.

Knowledge / content sources :-

1) File → Markdown, doc, txt, pdf, ppt

2) Web → Web page

3) Database → SQL, NoSQL

4) topic → Wikipedia, Arxiv

langchain.

Loaders / Readers → Agno

.pdf

→ S3

Word drive.

1) Load the document into memory

2) Parse the document → text, images, tables, links

3) Convert it into document type.

PDF.

type | content | metadata { no. of pages = —, creation = —,

saved on = —/

Modified — 4

We have the loaded document.

→ PDF read. extract

Document → Very large. → Into chunks.

logical.

Chapter wise → Topic wise.

Paragraph wise, Sentence wise.

Chapter 2

↓

LR

↓

20 pages

20 times

Some of the chunking strategies:-

1) fixed type → 100 words.

2) Semantic Chunking → Semantic meaning similar.

max chunk size.

3) Document Chunking → Well formatted.

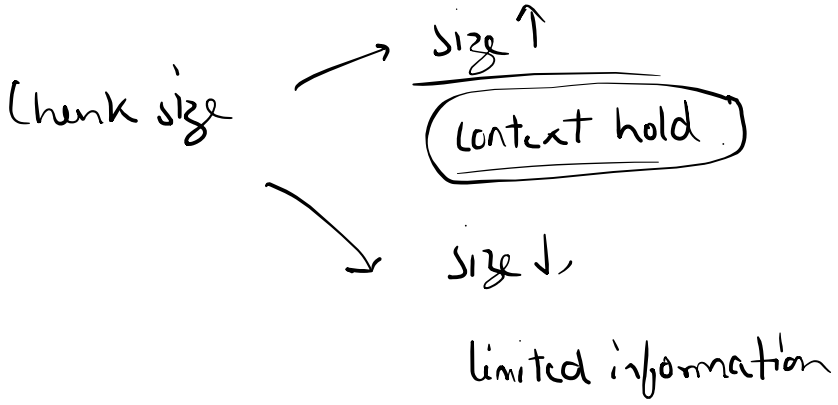
4) Recursive chunking →

2 selections-

1) Chunking strategy → Experimentation.

1) Chunking strategy → Experimentation.

2) Chunk size.



Load, chunks.

Into numbers. → Text ∝

Categorical → OHE

↓

encode → 0 1 2 3

Embedding → Semantic meaning of the text

dynamic → context

→ institution.

Money Bank

↓

0.3	0.1	0.6	0.9	0.5

→ location.

River Bank

↓

0.4	0.2	0.9	0.7	0.5

Semantic meaning

dynamic contextual embeddings

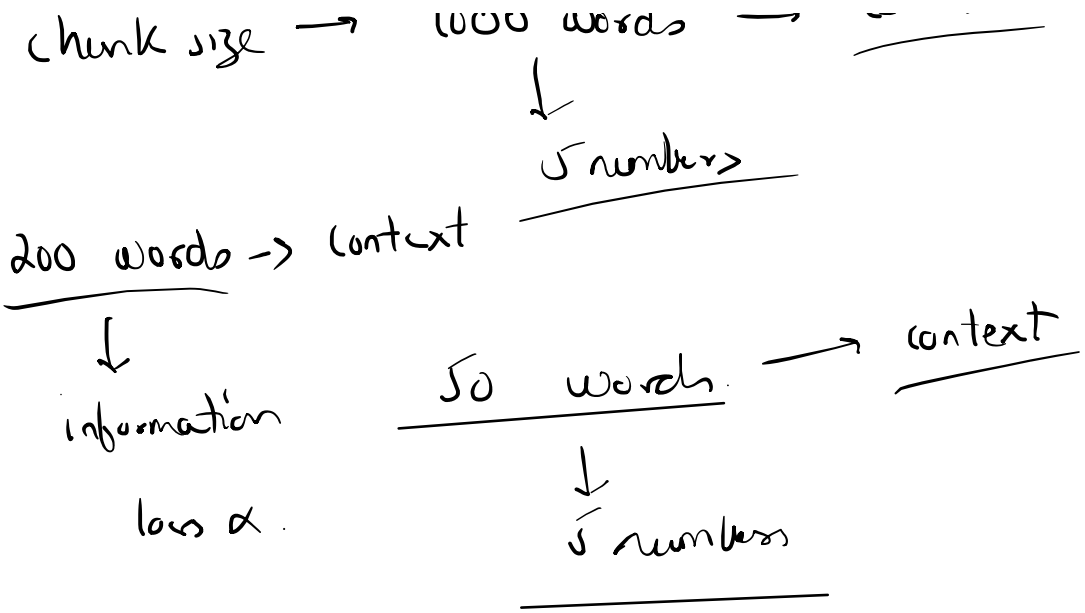
Load 500 page → Page wise chunk

↓ 500 documents

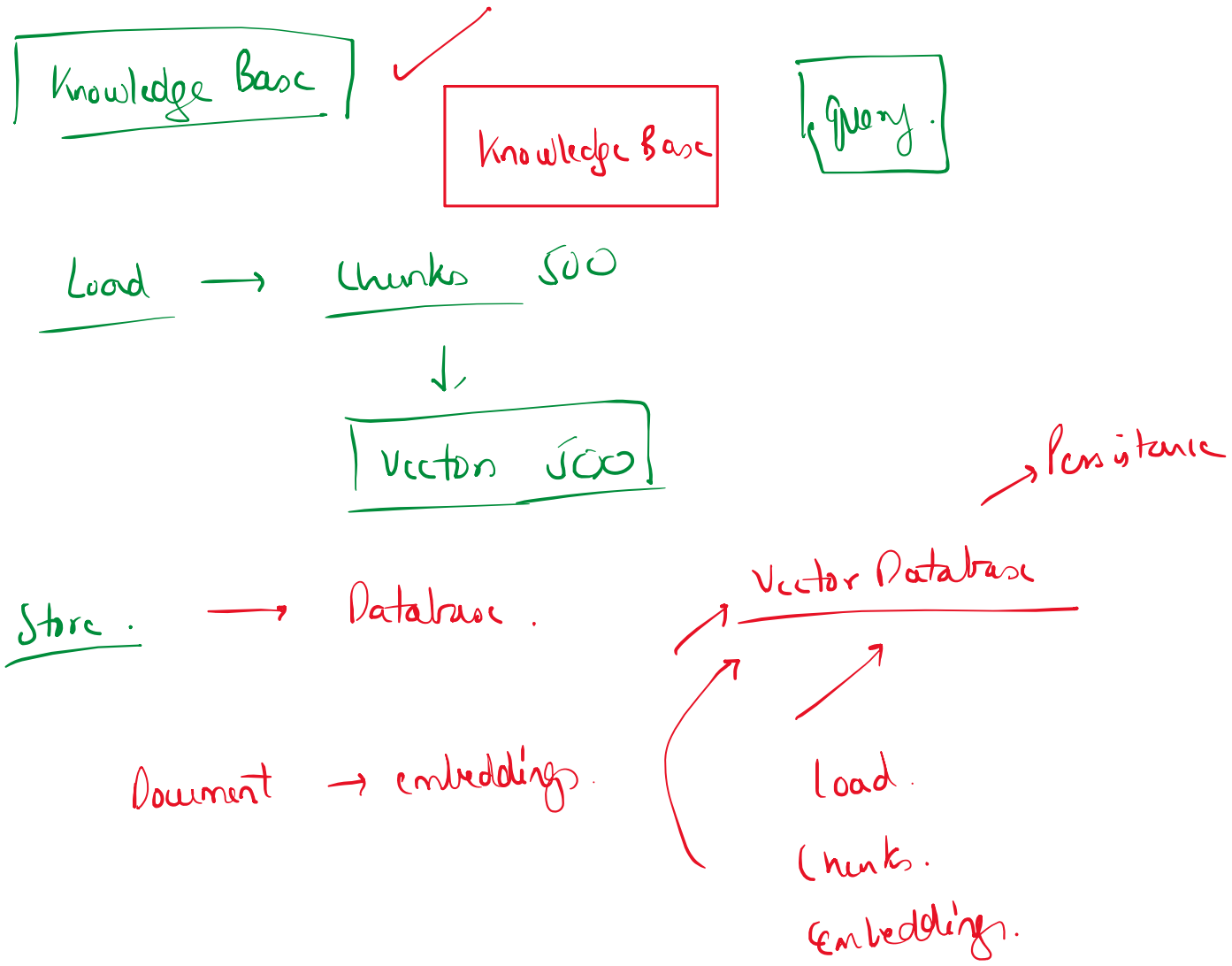
Embeddings 500

chunk size → 1000 words → context

↓


chunk size → 1000 words →
↓
5 numbers
200 words → context
↓
information
loss ∝
50 words → context
↓
5 numbers

4. Storage


Knowledge Base
Knowledge Base
Query.
Load → Chunks 500
Vector 500
Persistance
Store. → Database.
Vector Database
Document → embeddings.
Load.
Chunks.
Embeddings.

Documentational → Info

Very specific → based on the user input

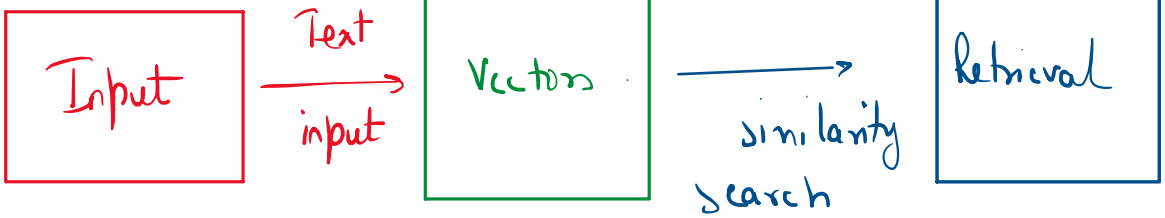
Leave policy → leave — similar to the input query.

Salary

Knowledge Base — Similarity search.

Input → Vectors.

✓ Embedding model. — consistent



Similarity search → cosine similarity — −1 to 1

vector distance

What is the leave policy of my company → Embedding model

Input vector

Context of my input

0.3	0.2	0.46	0.78	0.93

Context of my input

0.3	0.2	0.46	0.78	0.93

Knowledge base → 500 vectors → compare

500 → similarity score → sort descending order (using similarity.)

similarity search.

Top 3, top 5 documents retrieve.

top 5 vectors.

metadata

↓

text → Page.

Page 50, 63, 128, 250, 367

Retrieval phase.

↓

RAG : Retrieval Augmented Generation

enhance

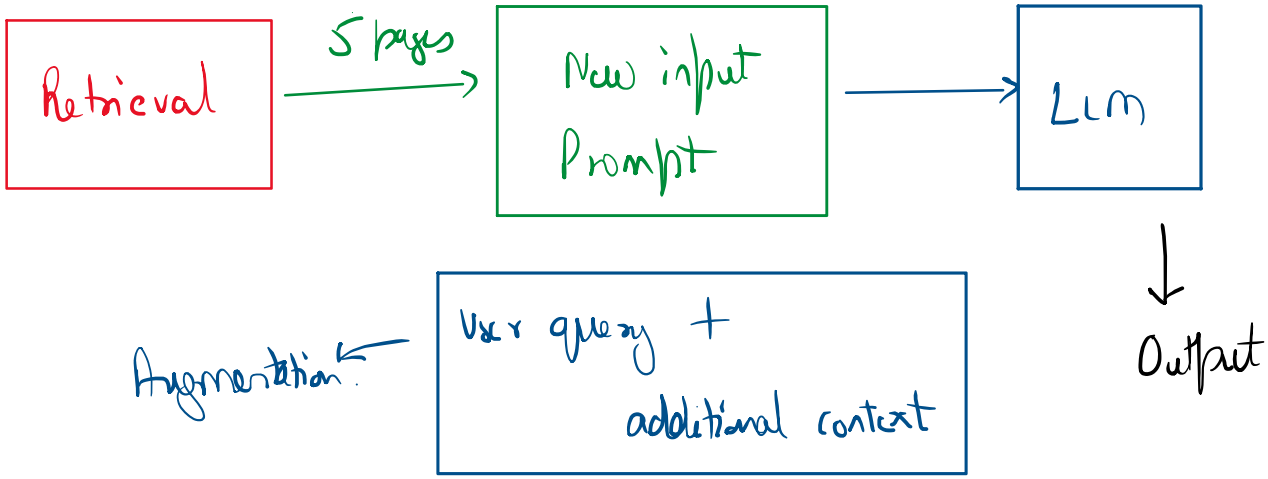
Prompt → Input

Based on the input query and context, answer the question. Make sure

query: ,

context:

to answer from the context.



Agentic RAG → Retrieval

↓

Smart Retrieval.

Agent has the capability:-

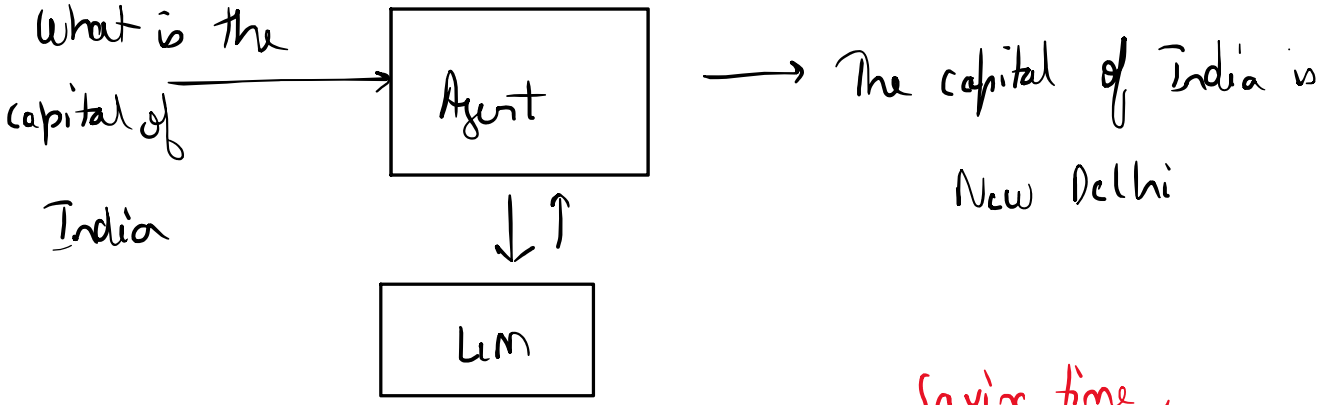
1) Agent autonomously decides when to search knowledge base.
2) Determines what to search.
3) Controls how to use the retrieved information.

Agent → Knowledge Base

Reactive.

Proactively.

↓ Documents add.



Tool → Knowledge

Saving time, input tokens.

Knowledge → Tool

Knowledge → tool

Knowledge Base ∝ Web

Websearch → Tool.

Leave policy → 5 pages → Additional documents

Better response.

Such leave, holidays.

Characteristics of Agentic RAG:-

1) Active decision making
2) Dynamic
3) Full control
4) Query Reformulation
5) Selective usage
6) Iterative search

Advantages:-

1) Efficiency →
2) Better search Quality → Knowledge Base ✓

Compare b/w (X) and (Y) → knowledge Base 2

3) Iterative Refinement

4) Context aware decisions.

5) Selective information use

Diff between Traditional vs Agentic RAG

	Traditional RAG	Agentic RAG
Decision making	Reactive	Proactive
Searching	Always	Agent decides
Query formulation	Direct	Reformulate for better retrieval
Number of searches	Usually 1	More than 1
Result Processing	Use all of the retrieved info	Agent selects the relevant info
Efficiency	less efficient	More efficient
Adaptability	No. fixed workflow	Dynamic workflow
Intelligent	Basic retrieval	Smart retrieval

Code

Teams → Multi-agent systems

Knowledge → Agentic RAG

agents → generic
↓
specific

↳ autonomy for retrieval

smart retrieval.

1) when to retrieve
2) how to retrieve
3) from where to retrieve

→ what to do with the retrieved documents

Reasoning → Brain → code.

Knowledge Base

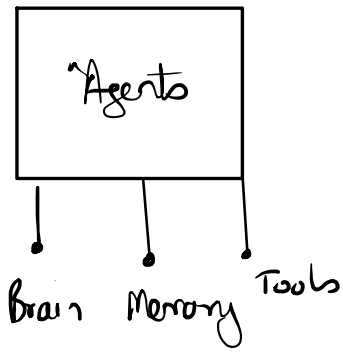
Retrieval

→ Tool

1) Reader
2) Chunking strategy
3) Embeddings

4) Vector DB.
5) Retrieval
↓
Agentic process.

Search knowledge → Tool access.



Complex task

Brain → Planning

↓

Into multiple sub tasks.

Achieve → Complete

Tool → Tool call ← Reasoning

Reasoning → from the list of tools

↓

Selection of appropriate Tool

Reasoning → Forms the input for the tool.

example_tool (name : str, age : int)

↓

Response.

query → Natural language query.

→ Suitable inputs to the tool.

Convert to proper dtypes.

str

It understands ← Tracks the execution ← Passes on the inputs

of tool. to the tool.

It understands ← of tool → to the tool

the tool result

↓

output for the user in natural language.

Planning phase → 25 subtasks.

2 tools → 50 tool calls.

A lot to handle for a single agent.

Separation of concern.

5 Agents

↓

Selected tools.

Complex task ✓

↓

25 subtasks.

↓

50 tool calls.

✓ Github Repo, ✓ Slack messages, ✓ email send

Jack of all trades.

Separation of concern

Agent

3 Agents → github agent → Github related

→ Slack agent → Slack related

→ Gmail agent → Gmail related

3 Agents which are experts in what they

do

Manager → Orchestrator model → query.

↓

Planning

↓

Subtasks.

tools → Action.

members attached.

Captain of a Team

Team leader → knows about the capabilities of all of its members.

3 members. a) Github agent c) Email agent

b) Slack Agent

Delegates tasks to members. → delegate task to members tool.

↓

Power.

1) Member → github.

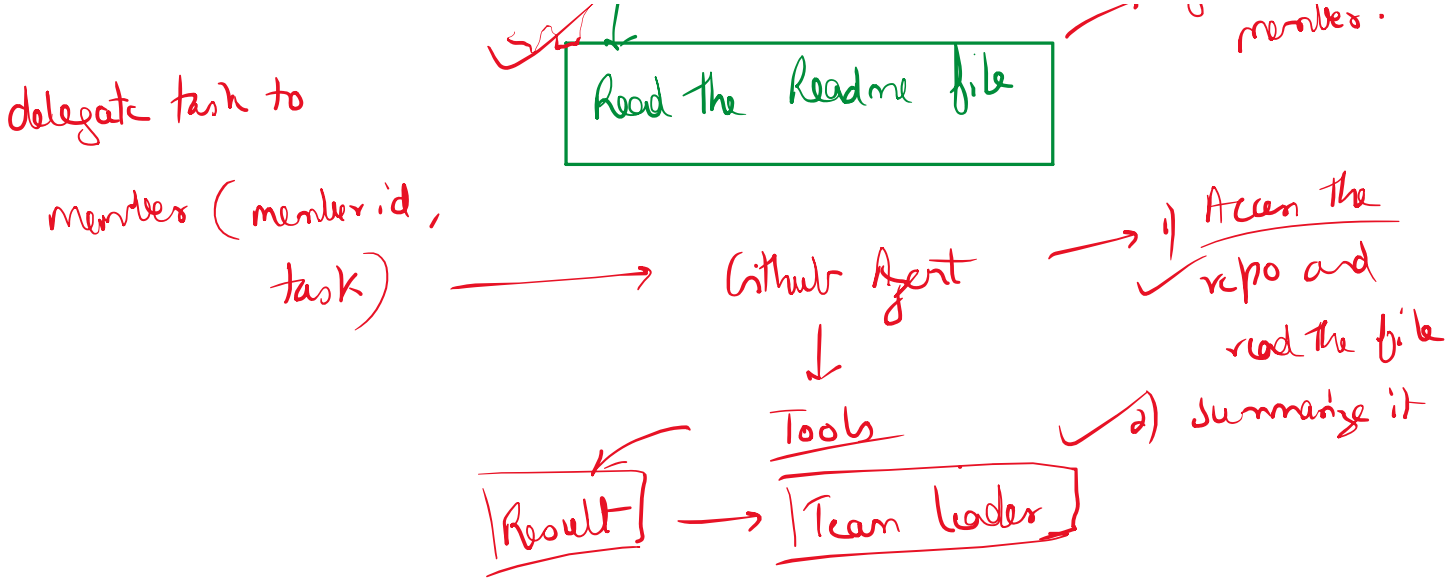
2) task → Read the Readme file of this Repo

Github related tools.

Team leader

Complex goal → 10 subtasks

Read the Readme file → github members.


Read the Readme file
delegate task to
member (member_id, task)
Github Agent
1) Access the repo and read the file
Tools
2) summarize it
Result
Team Leader

Team → Team-leader (orchestrator)

↓

delegate tasks to its members.

Flow :-

1) Team input → user query

2) Team leader → reads the input → decides on how to break it into subtasks.

3) Team leader delegates specific tasks to appropriate members

4) Team members complete their assigned tasks and return their results. Agents

5) Decide → whether to delegate further to members, or else it synthesizes all outputs into a fine cohesive response

↓

to the user

Complex goal.

to the user

Task form → Team leader

complex goal.
↓
for each member
↓
generate specific task.

Team leader has 4 imp roles:-

1) Team leader understands the context of input query and creates subtasks.

2) Team leader knows about its members in the team, their capabilities and also which tools each member has.

3) Team leader has to delegate task to the appropriate member. The task itself is generated by the team leader and is generally diff from the user query.
This allows the member to perform the task efficiently.

4) Team leader compiles all the outputs from its members.

and send the response back to the user.

Teams → Multi-Agent system

Collection of Agents → Collaboration.

Orchestrator Agent → Team leader → 5 members → Capabilities + Roles

Team leader → task delegation → Member 1, Member 2, Member 3

Team leader → frame the input

Team member. → 3 steps

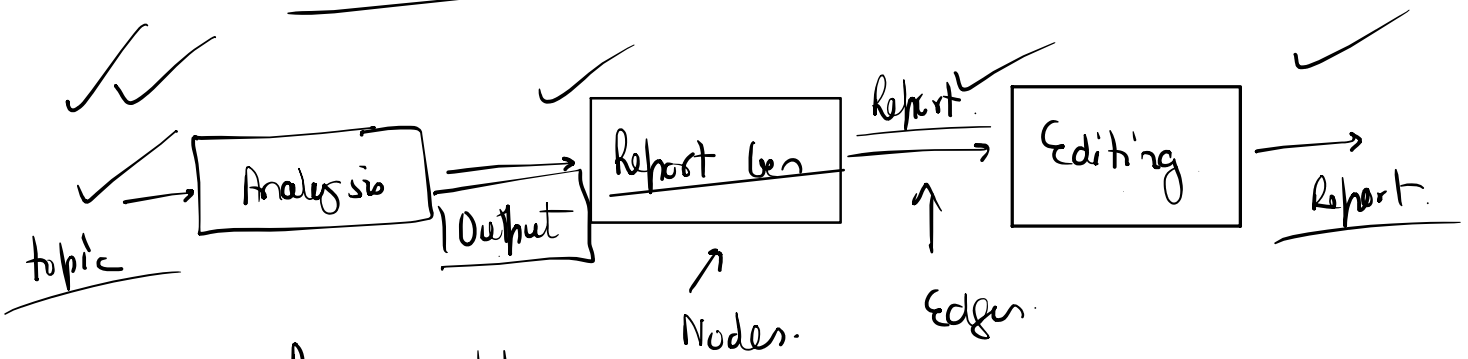
Output

Dynamic

Deterministic Multi agent system.

Path of execution

topic



As a graph

Node → steps.

Edges. → Connections

Agents → steps. Report generation step. → Agent

Boxes. → Nodes

Workflow → steps → order

Predetermined Path of execution

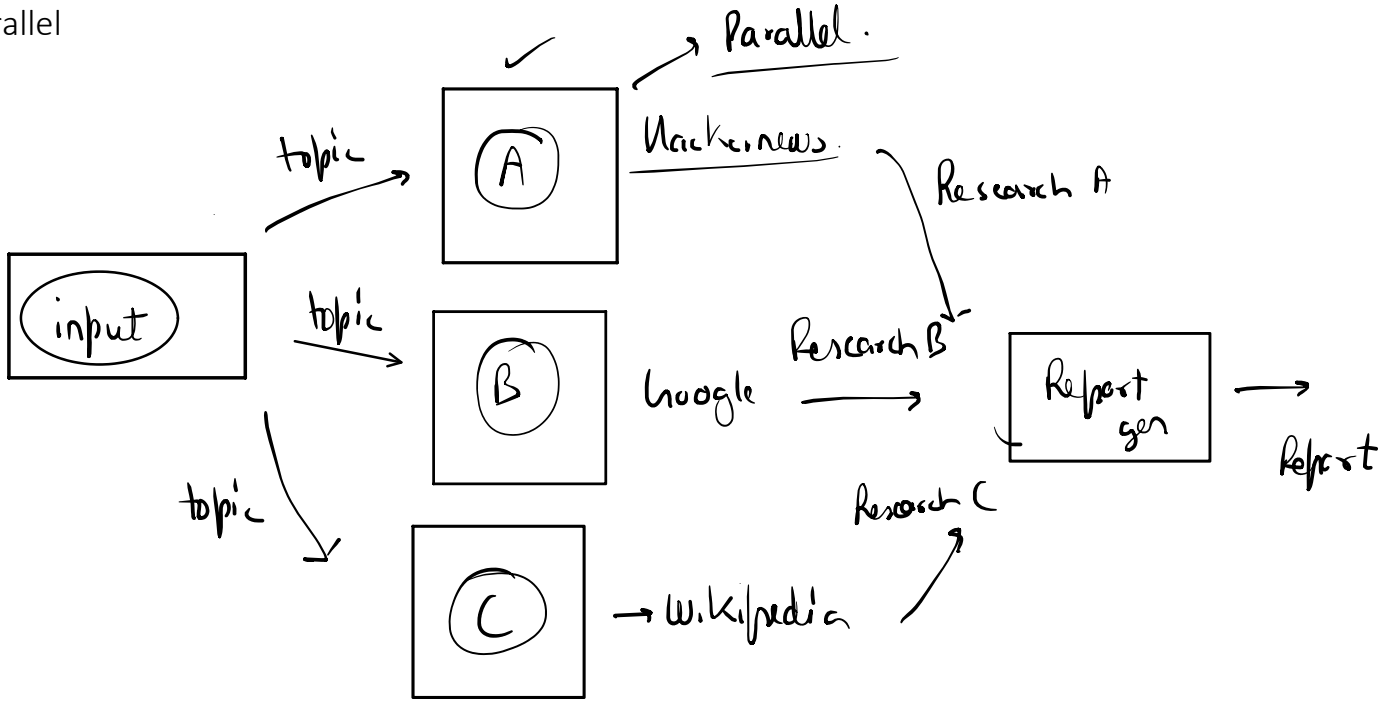
1) Sequential.

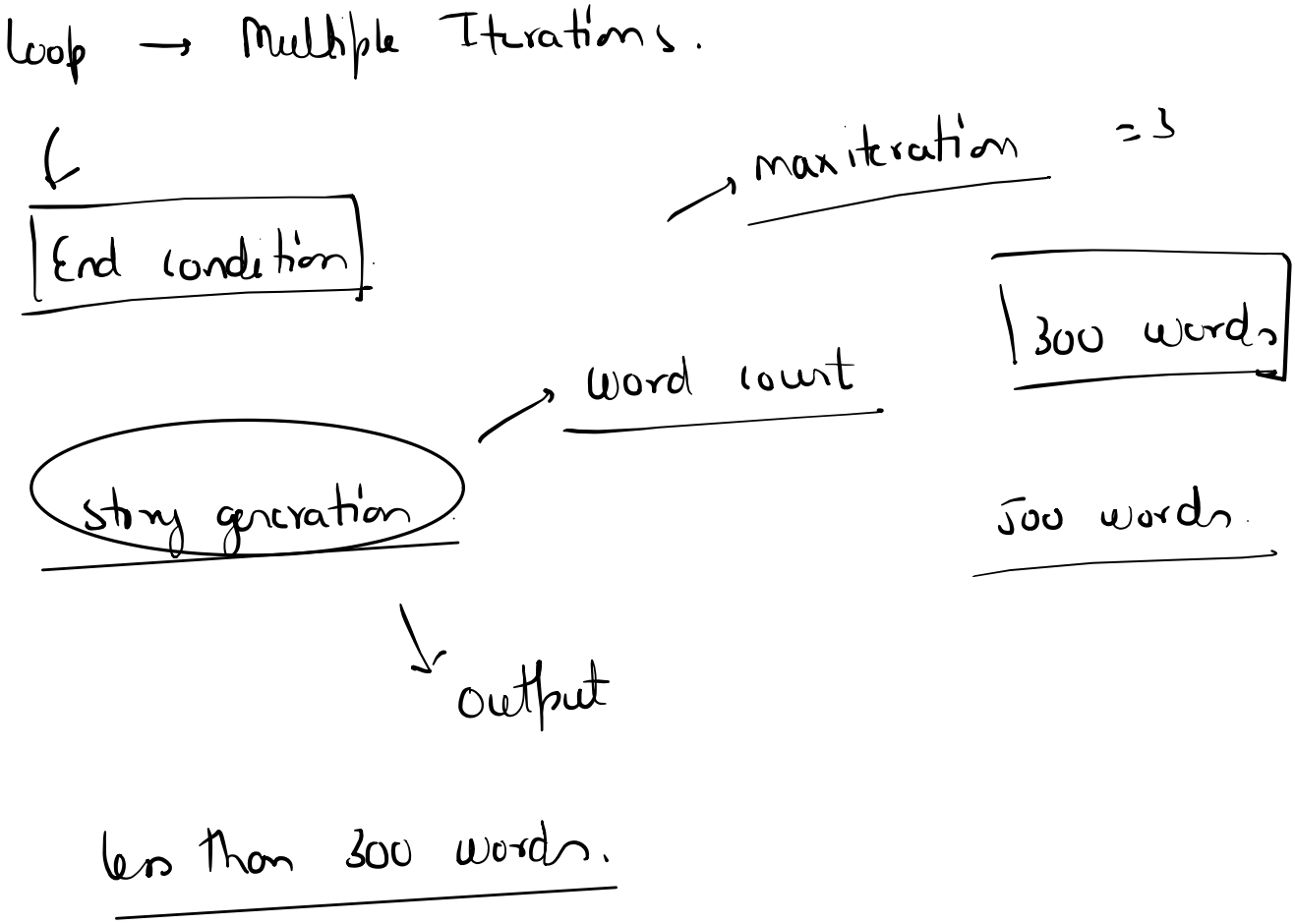
2) Parallel

3) Conditional

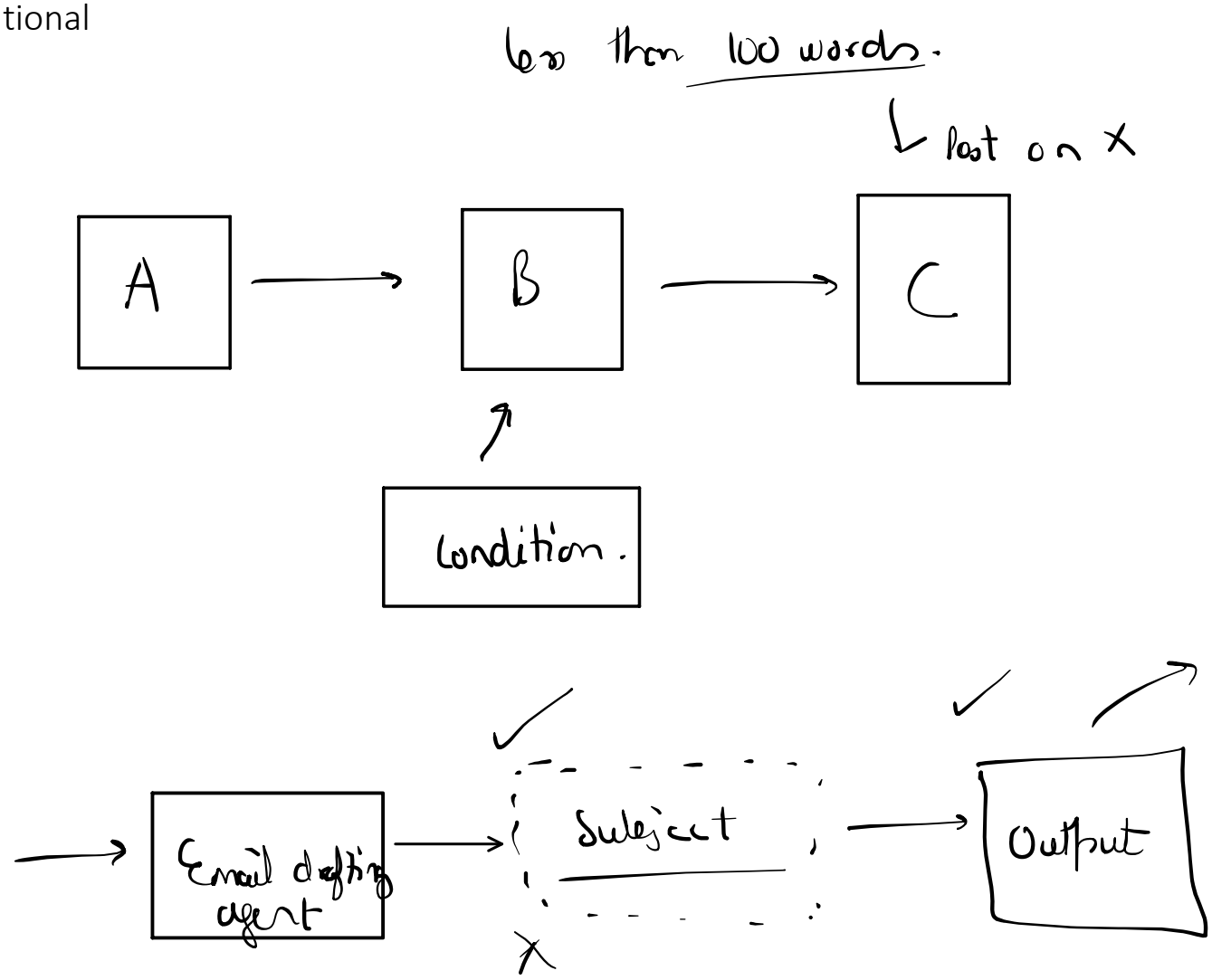
4) looping.

Parallel

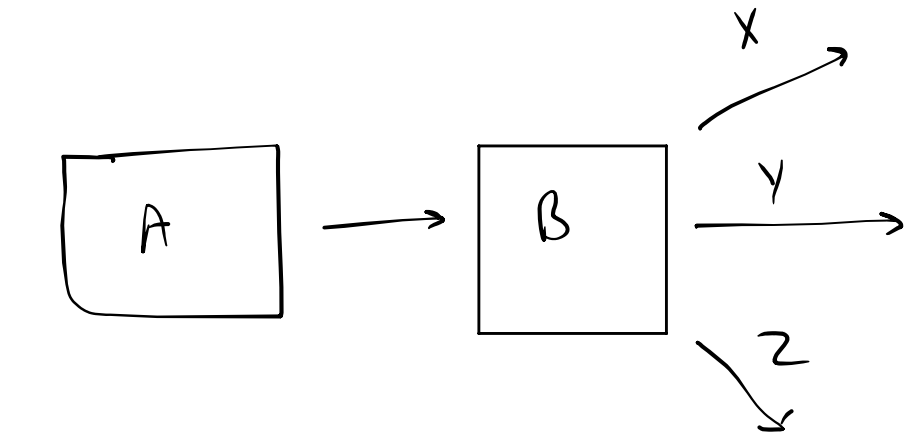

Parallel.
topic
A
Hackernews
Research A
input
topic
B
Research B
Google
Report gen
Report
topic
Research C
C
Wikipedia

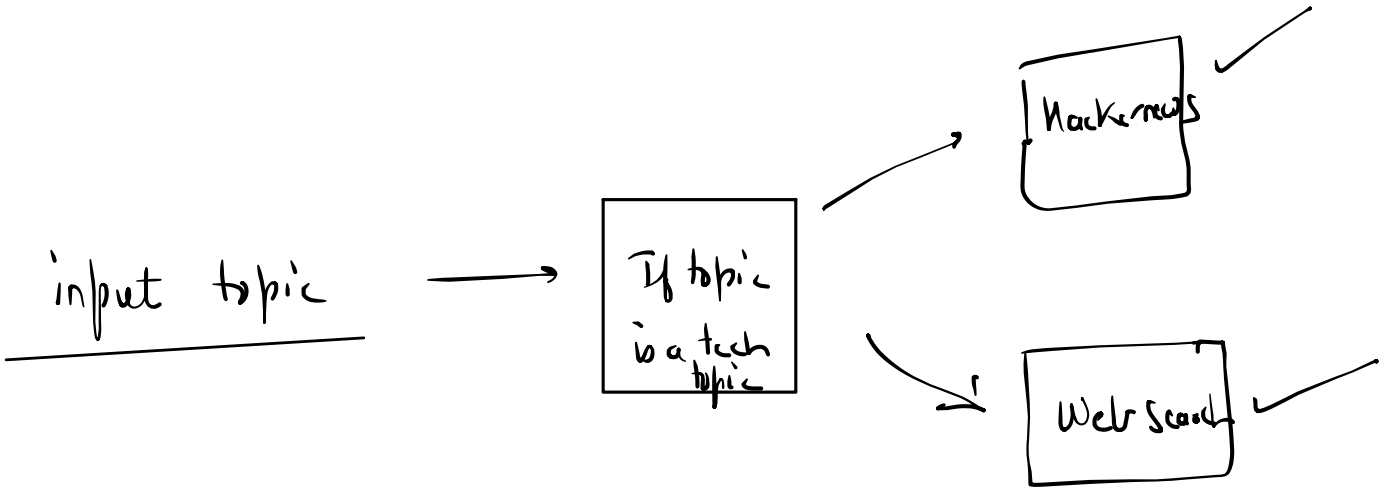

Loop → Multiple Iterations.
End condition
max iteration = 3
300 words
Word count
Story generation
300 words
output
less than 300 words.

Conditional



Router/Branching





Workflows in Agno. → Predetermined series of steps.

Steps. → Agent → task.

Connect → Sequential, Parallel, Conditional (dynamic workflows), looping.

Graphs → Nodes → Step

Edges → Connections, Combinations.

Medium Article Generation Project

From where to read about the topic → Resources.

Topic → Resources → Validate

Validate ← Read

Validate → Summarize.

Compile → Remove duplicate info

Agno.

Teams

Capability.

↓

Compile → Report

↓

Article → Publish on Medium.

Orchestrator Agent → Medium Article.

↓

Save → markdown

↓

Team members → Specialised Agents → Research material.

↓

Resource → Tools Resource.

↓

Agent → Wikipedia

Arxiv

Reddit

Websearch

Twitter

Compile.

↓

Article create. → Write as a markdown file.

Agno OS → Chat with your Team

Data Science Team

Data Ingestion

Data Understanding

EDA

Data cleaning

Feature Engg

Model training

Model Evaluation → Metrics

HP tuning stage.

Orchestrator → Manager

Data ingestion.

✓ EDA

Plots

FE

Model training

Model Evaluation.

Tools

Pandas.

matplotlib

Chat interface → Dataset → dir.

EDA → Summarized.

Plots → save → artifacts

Web capability

FE → Execute → train.csv, test.csv

Model training → Code.

| Model training | → Code

Execute → Trained model

Artifacts