# Introduction to Autonomous Agents with Microsoft AutoGen

Yogesh Haribhau Kulkarni

YHK

# Outline

1 Introduction

2 Implementation

3 Open Source LLMs

4 Conclusions

5 References

YHK

# Introduction

# Future AI?

- What are future AI applications like?
  - Generative: Generate content like text & image
  - Agentic: Execute complex tasks on behalf of human
- How do we empower every developer to build them?:
  - Co-Pilots
  - Autonomous

YHK

# Introduction to AI Agents

- 2024 is expected to be the year of AI agents.
- AI agents combine multiple components to solve complex problems.
- Shifting from monolithic models to compound AI systems.
- Compound AI systems use system design for better problem solving.
- AI agents improve with reasoning, acting, and memory components. (ReAct = Reasoning + Acting)

YHK

# From Monolithic Models to Compound AI

- Monolithic models are limited by training data.
- Hard to adapt without significant data and resources.
- Example: Vacation planning with no personal data access.
- Compound AI systems solve this by integrating multiple components.
- External databases and tools enhance model responses.

YHK

# System Design in Compound AI

- Systems are modular: combine models, tools, and databases.
- Easier to adapt and quicker to solve problems.
- Combining models with external components for enhanced output.
- Example: Search query integrated with model for better accuracy.
- Programmatic control logic guides the response.

YHK

# Role of Agents in Compound AI

- Agents use large language models (LLMs) for reasoning and planning.
- LLMs break down problems into manageable tasks.
- Slow thinking (planning) leads to better solutions.
- Agents provide flexibility in solving complex tasks.
- The agent's role is to manage logic and interact with tools.

YHK

# Components of LLM Agents

- Reasoning: Break down complex problems into steps.
- Acting: Use external tools like databases or calculators.
- Memory: Store logs and conversation history for personalization.
- Tools: External programs that help solve problems (e.g., search, calculators).
- Configurations: Adjust agents using frameworks like ReACT.

YHK

# Example: REACT Agent in Action

- REACT agents think slowly, plan, and act iteratively.
- User query feeds into the agent with reasoning instructions.
- The agent may call external tools and evaluate the results.
- If the result is wrong, the agent revises its plan.
- Example: Calculating sunscreen needs for a vacation.

YHK

# Modularity and Complex Problem Solving

- Modular AI systems can solve more complex problems.
- Example: Planning vacation with weather, sunscreen, and health data.
- Agents handle multiple paths to find the best solution.
- Compound AI is flexible and adapts to different problem scopes.

YHK

# Agent Autonomy in Problem Solving

- Narrow problems can be solved with fixed, programmatic systems.
- Complex tasks require agent autonomy for better flexibility.
- Autonomy level depends on task complexity and need for iteration.
- Agents are effective for complex, diverse tasks (e.g., GitHub issues).

YHK

# The Future of Agent Systems

- AI agents are evolving rapidly with system design and autonomy.
- Human-in-the-loop still essential for accuracy in early stages.
- Agents will play a key role in diverse industries and tasks.
- Expect increasing adoption in 2024 and beyond.

YHK

# Examples of Agentic AI

- Agents mean ACTION
- Agentic AI means ACTION using AI, meaning LLM
- Examples:
  - Personal assistants
  - Autonomous robots
  - Gaming agents
  - Science agents
  - Web agents
  - Software agents

YHK

# Autonomous AI Agents

- Collaborative approach yields astonishing enhancements in performance and capabilities. Contrasted with using a single AI, such as ChatGPT, in isolation.
- Ability to assume distinct roles within a team. Like professionals in various fields.
- Each agent contributes specialized expertise to the conversation.

YHK

# The Blueprint

- Planning: Reflects on past experiences, offers self-critiques, and breaks down tasks into manageable steps using sub-goal decomposition.
- Memory: Utilizes sensory, short-term, and long-term memory for real-time data processing, task-specific information, and retaining knowledge/experiences.
- Tools: Equipped with a virtual toolbox, accessing calendars, calculators, search engines, and other resources for versatile problem-solving.

YHK

# Flow: The Symphony

- Task Decomposition
- Model (LLM) Selection
- Task Execution leveraging planning, memory, and tools.
- Response Generation

YHK

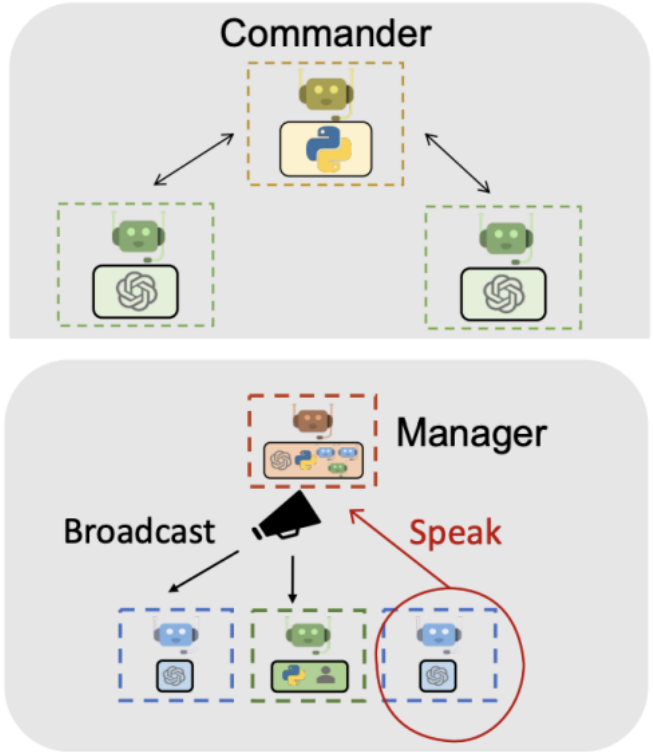
# Agentic Frameworks' Needs

- Intuitive unified agentic abstraction
- Flexible multi-agent orchestration
- Effective implementation of agentic design patterns
- Support diverse application needs
- Handle more complex tasks / Improve response quality
- Easy to understand, maintain, extend Modular composition, Natural human participation, Fast & creative experimentation
- Agentic Abstraction: Unify models, tools, human for compound AI systems

YHK

# Multi-Agent Orchestration

- Static/dynamic
- Context sharing/isolation
- Cooperation/competition
- Centralized/decentralized
- Intervention/automation




YHK

# Popular Agentic Frameworks

- BabyAGI: Pioneering AI learning system.
- AutoGPT: Automates content generation.
- GPT Engineer: Assists in coding and software development.
- Langraph: Graph-based control flow
- CrewAI: High-level static agent-task workflow
- AutoGen: Dialog based planning and execution

YHK

# Implementations

# AutoGen

# What is AutoGen?

- Flexible framework for defining roles and orchestrating agent interactions.
- Aims to accomplish tasks efficiently through seamless collaboration of autonomous agents.
- Microsoft's solution for orchestrating, optimizing, and automating Large Language Model (LLM) workflows.

YHK

# It all started with ...

## AutoGen: Enabling Next-Gen LLM Applications via Multi-Agent Conversation

**Qingyun Wu[†], Gagan Bansal[*], Jieyu Zhang[±], Yiran Wu[†], Beibin Li[*]**

**Erkang Zhu[*], Li Jiang[*], Xiaoyun Zhang[*], Shaokun Zhang[†], Jiale Liu[∓]**

**Ahmed Awadallah[*], Ryen W. White[*], Doug Burger[*], Chi Wang[*1]**

[*]Microsoft Research, [†]Pennsylvania State University

[±]University of Washington, [∓]Xidian University

YHK

# Framework

- Agents may handle code generation, execution, and human supervision.
- Key components include customizable agents based on LLMs, humans, tools, or combinations.
- Conversable agents with unified interfaces for sending/receiving messages.
- Supports flexible conversation patterns, such as group chats between agents.

YHK

# Unified Interface

- Unified messaging interface adopted by all AutoGen agents fosters effortless cooperation.
- Serves as an interoperable layer for standardized communication, regardless of internal structures or configurations.
- Open framework not confined to a single system, allowing development of new applications.

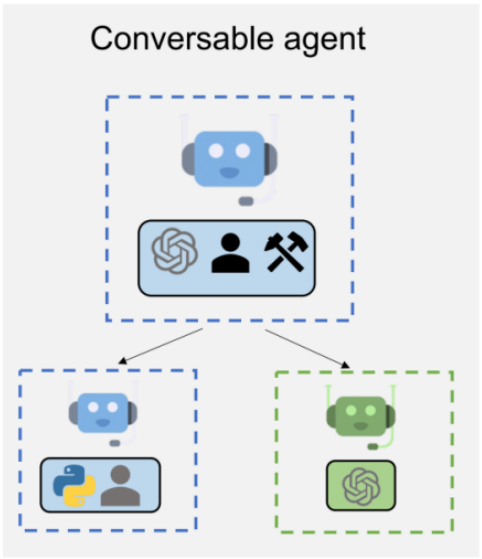
YHK

# Unique Features

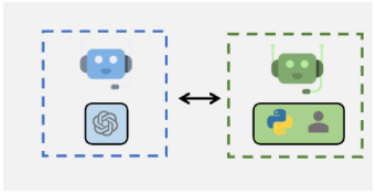
- Some pre-cooked agent types are provided viz Assistant, User Proxy Agent, etc.
- Assistant is like a standard chatbot, given a query it will answer.
- User Proxy Agent is your ie user's Proxy. So it has the task to get done. It initiates the chat.
- There are properties within it to have Human In Loop ie interactive, ALWAYs, NEVER, TERMINATE. If you want fully autonomous working, then set it to NEVER.
- Group Chat Manager offers creating chat rooms of AI agents.

YHK

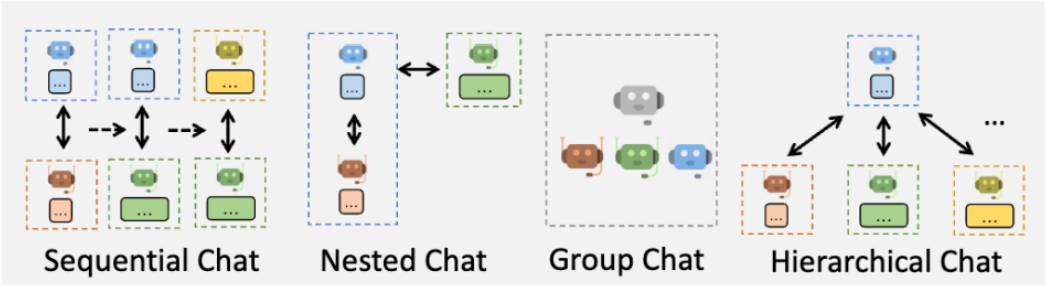
# Define agents and Get them to talk




Agent Customization



Multi-Agent Conversations




Flexible Conversation Patterns

(Ref: Agentic AI Frameworks & AutoGen - Chi Wang)

YHK

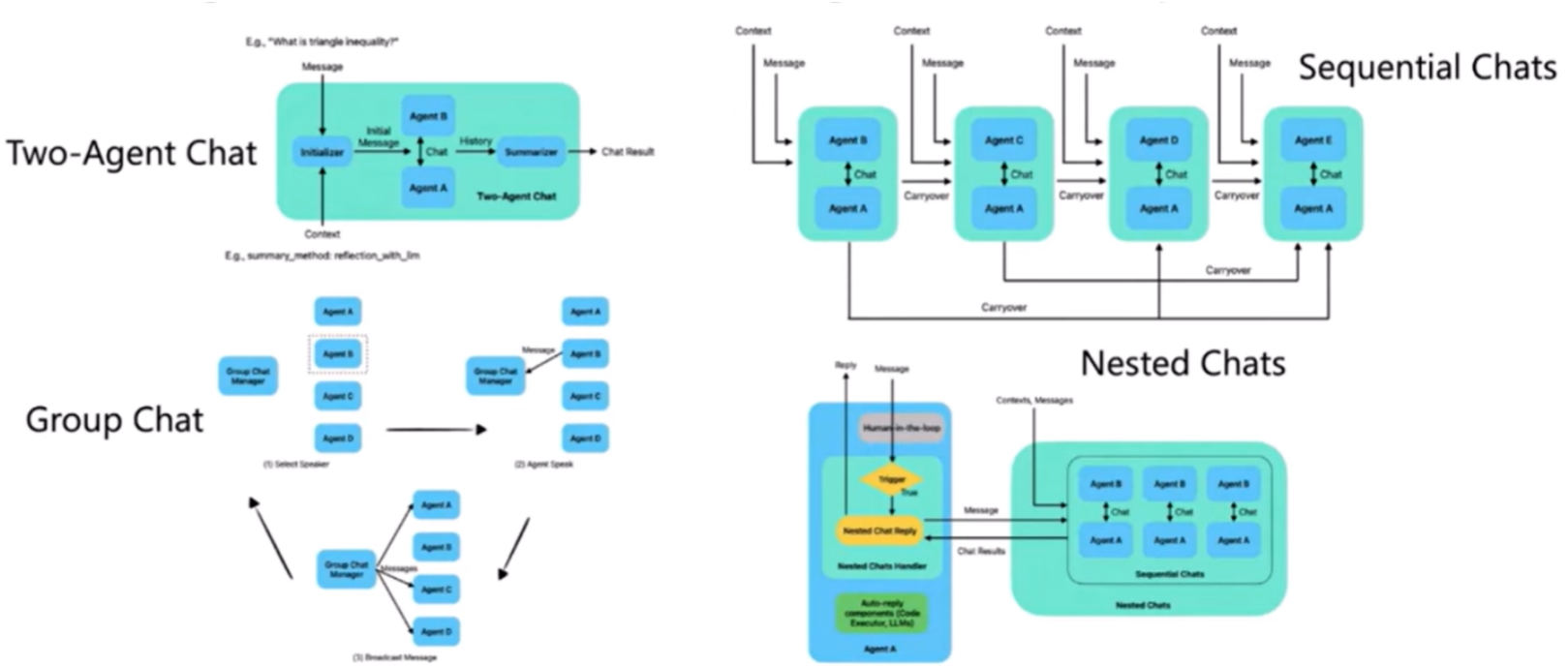
# System Message

```
You are a helpful AI assistant. Solve tasks using your coding and language skills.

In the following cases, suggest python code (in a python coding block) or shell script (in a sh
coding block) for the user to execute.

1. When you need to collect info, use the code to output the info you need, for example, browse or
search the web, download/read a file, print the content of a webpage or a file, get the current
date/time. After sufficient info is printed and the task is ready to be solved based on your
language skill, you can solve the task by yourself.

2. When you need to perform some task with code, use the code to perform the task and output the
result. Finish the task smartly.

Solve the task step by step if you need to. If a plan is not provided, explain your plan first. Be
clear which step uses code, and which step uses your language skill.

When using code, you must indicate the script type in the code block. The user cannot provide any
other feedback or perform any other action beyond executing the code you suggest. The user can't
modify your code. So do not suggest incomplete code which requires users to modify. Don't use a
code block if it's not intended to be executed by the user.

If you want the user to save the code in a file before executing it, put # filename: <filename>
inside the code block as the first line. Don't include multiple code blocks in one response. Do not
ask users to copy and paste the result. Instead, use 'print' function for the output when relevant.
Check the execution result returned by the user.

If the result indicates there is an error, fix the error and output the code again. Suggest the
full code instead of partial code or code changes. If the error can't be fixed or if the task is
not solved even after the code is executed successfully, analyze the problem, revisit your
assumption, collect additional info you need, and think of a different approach to try.

When you find an answer, verify the answer carefully. Include verifiable evidence in your response
if possible.

Reply "TERMINATE" in the end when everything is done.
```

(Ref: AutoGen - John Tan Chong Min)

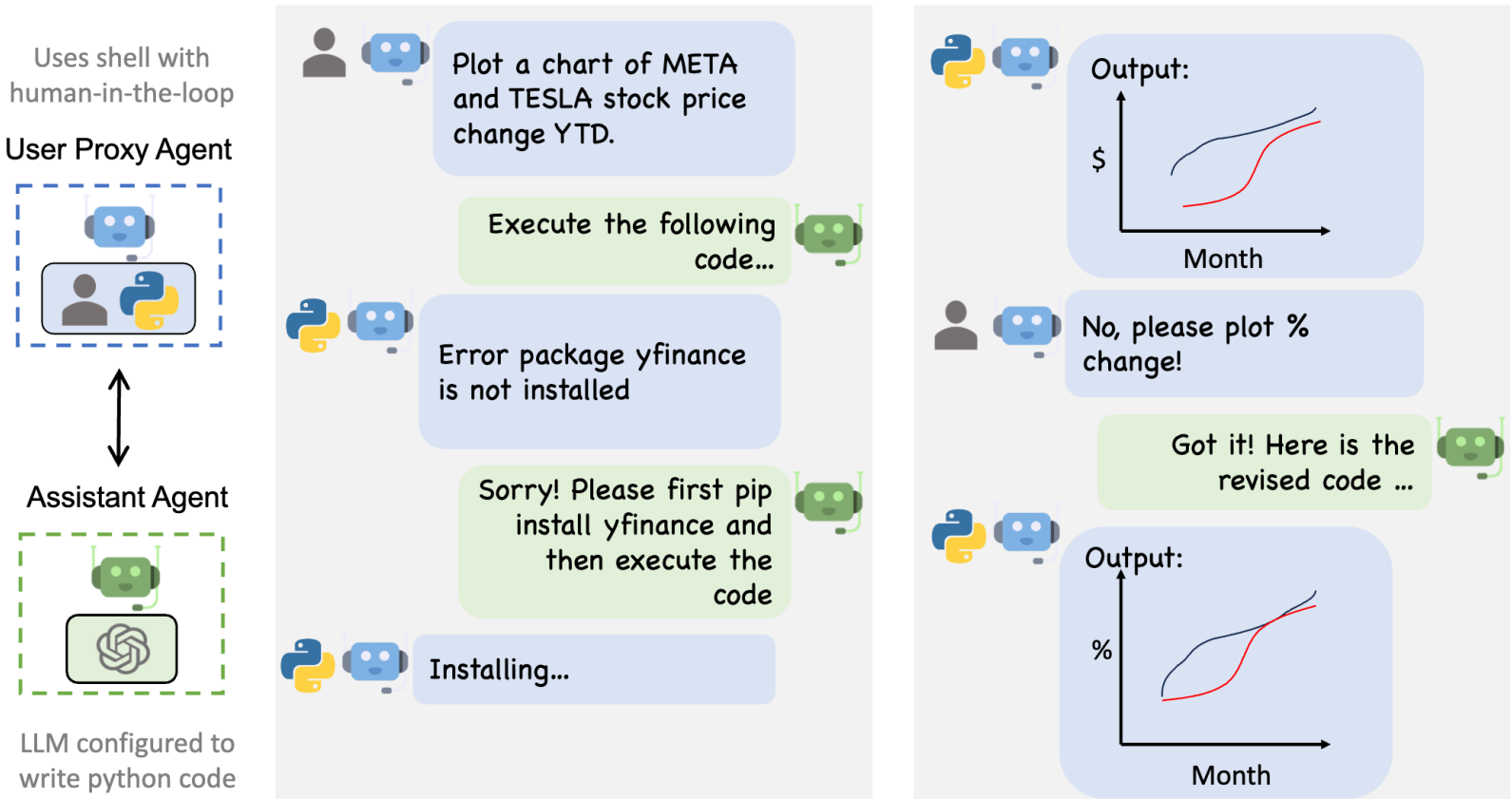
YHK

# Conversation Patterns




(Ref: Build Agentic AI Apps with the Autogen framework — OD539 - Microsoft Developer)

# Example: Plot Stocks




(Ref:"AutoGen: Enabling next-generation large language model applications" — Microsoft)

YHK

# Two Agents Chat

- Two agents share same thread
- Assistant suggestions code and Executor runs the code (code execution is typically done the docker sandbox for safety) or Human-in-the-loop for executor)

two-agent-chat.py

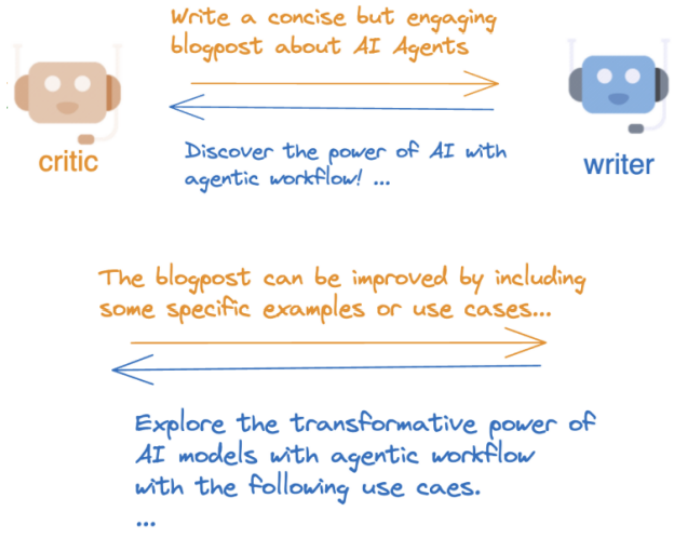
```
programming_assistant = ConversableAgent(
    system_message="You are a programming assistant, and you solve tasks
using code...",
    llm_config={"model": "gpt-4-turbo", "api_key": ...},
    ...
)
code_executor = DockerCommandLineExecutor(
    image="python:3.11",
    ...
)
programming_executor = ConversableAgent(
    code_execution_config = {"executor": code_executor},
    human_input_mode = "ALWAYS",
    ...
)
result = programming_executor.initiate_chat(
    programming_assistant,
    message="What is the sum of 74th and 90th Fibonacci number?"
)
```


(Ref: Build Agentic AI Apps with the Autogen framework — OD539 - Microsoft Developer)

YHK

# Example: Blogpost Writing with Reflection

```python
writer = autogen.AssistantAgent(
    name="Writer",
    system_message="You are a writer...",
    llm_config=llm_config,
)

critic = autogen.AssistantAgent(
    name="Critic",
    is_termination_msg=lambda x: x.get("content", "").find("TERMINATE") >= 0,
    llm_config=llm_config,
    system_message="You are a critic...",
)

critic.initiate_chat(
    recipient=writer,
    message=task,
    max_turns=2,
    summary_method="last_msg"
)
```



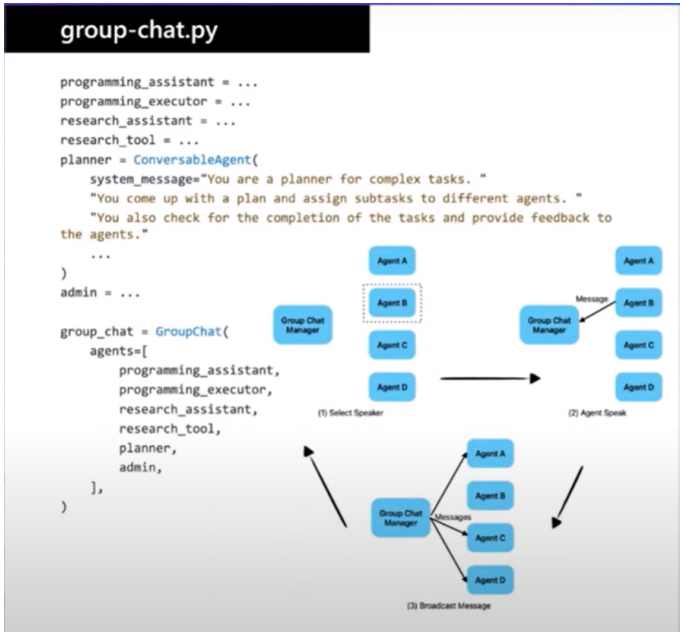

(Ref: Agentic AI Frameworks & AutoGen - Chi Wang)

# Group Chat

- Agents participate in a single thread
- Speaker is selected by a group chat manager.
- Planner agent to plan and guide other agents
- Admin agent for collecting human feedback
- Each agent could be simple or group or sequential nested agent, making this composable and more complex if needed.

group-chat.py

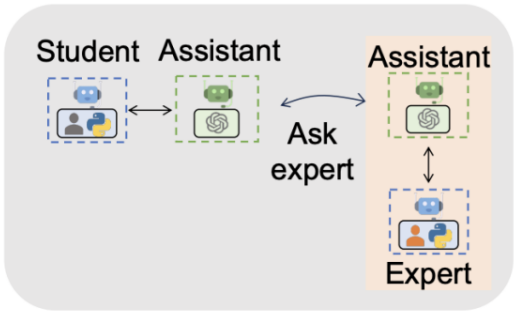
```
programming_assistant = ...
programming_executor = ...
research_assistant = ...
research_tool = ...
planner = ConversableAgent(
    system_message="You are a planner for complex tasks. "
    "You come up with a plan and assign subtasks to different agents. "
    "You also check for the completion of the tasks and provide feedback to
the agents."
    ...
)
admin = ...

group_chat = GroupChat(
    agents=[
        programming_assistant,
        programming_executor,
        research_assistant,
        research_tool,
        planner,
        admin,
    ],
)
```



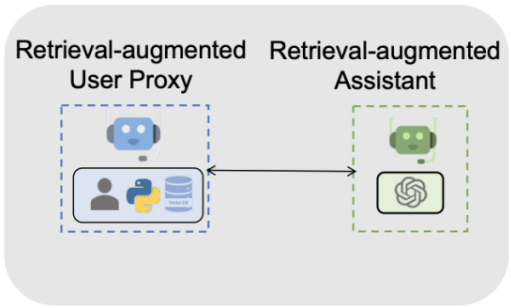

(Ref: Build Agentic AI Apps with the Autogen framework — OD539 - Microsoft Developer)

YHK

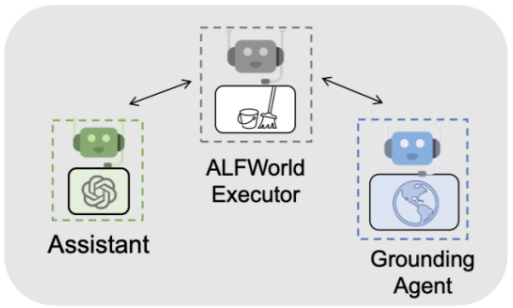
# More examples



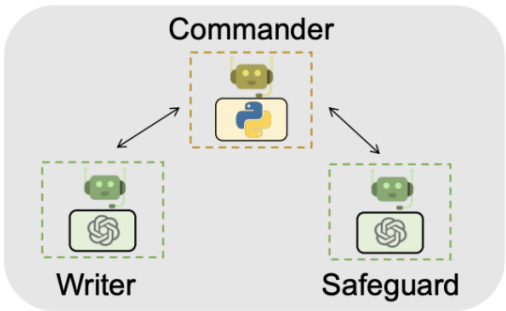

A1. Math Problem Solving



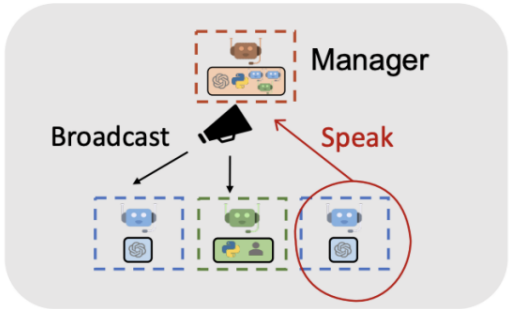

A2. Retrieval-augmented Q&A



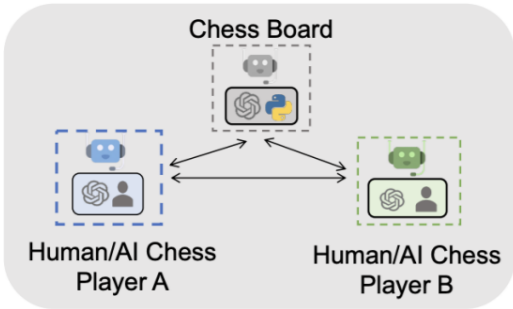

A3. Decision Making in Embodied Agents




A4. Supply-Chain Optimization




A5. Dynamic Task Solving with Group Chat




A6. Conversational Chess

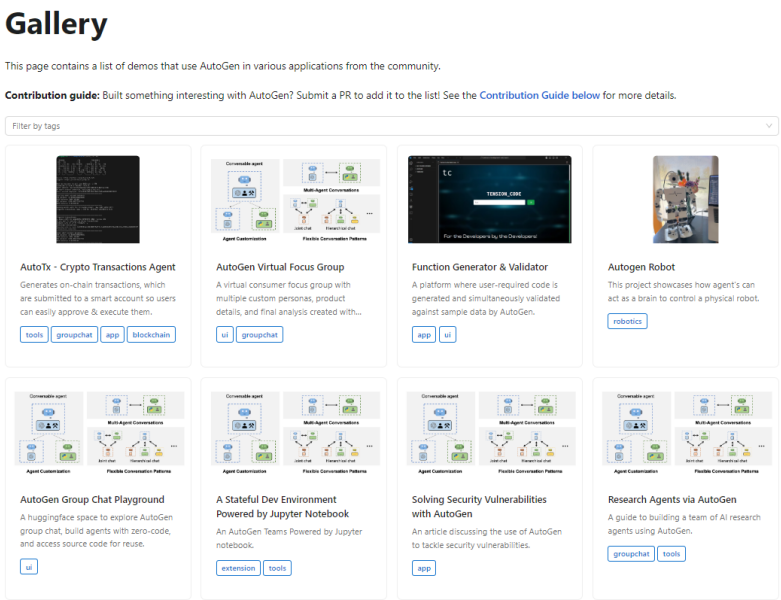
For more examples: https://autogen-ai.github.io/autogen/docs/notebooks

(Ref: Agentic AI Frameworks & AutoGen - Chi Wang)

YHK

# Applications

- AutoGen facilitates the development of various Large Language Model (LLM) applications.
- Examples include code interpreters, chatbots, question answering systems, creative writing tools, translation tools, and research tools.
- Many application examples can be seen in https://microsoft.github.io/autogen/docs/Gallery




(Ref:https://microsoft.github.io/autogen/docs/notebooks)

# Applications

- Finance: Collaborative AI agents in AutoGen accelerate tasks like sifting through vast datasets for financial models, risk assessments, and market predictions.
- Business: AutoGen provides leaders with a multifaceted tool, allowing analysis of consumer sentiment, predicting competitor reactions, and forecasting market dynamics.
- Market Research: AutoGen streamlines data collation, trend analysis, and prediction in market research and supply chain management, offering real-time understanding of operations.
- Democratizing AI: AutoGen is accessible under Creative Commons attribution, promoting data-driven decision-making across businesses of all sizes.
- Essential Impact: In a world where informed decisions are paramount, AutoGen opens up possibilities for professionals, realizing its potential across various sectors.

YHK

# AutoGen Implementation

# AutoGen: Building Multi-Agent Conversations

- Two-step process.
- **Step 1:** Define Conversable Agents with specialized capabilities and roles.
- **Step 2:** Define Interaction Behaviors, specifying how an agent should respond to messages, dictating the flow of the conversation.
- OpenAI APIs by default.
- Need to use LM Studio to serve local LLMs (more info on my blog at Medium)

YHK

# Configuration

```
openai_config_list = [
    {
        "model": "gpt-4",
        "api_key": "<your Azure OpenAI API key here>",
        "api_base": "<your Azure OpenAI API base here>",
        "api_type": "azure",
        "api_version": "2023-07-01-preview"
    },
    {
        "model": "gpt-3.5-turbo",
        "api_key": "<your Azure OpenAI API key here>",
        "api_base": "<your Azure OpenAI API base here>",
        "api_type": "azure",
        "api_version": "2023-07-01-preview"
    }
]

```

YHK

# Simple Query

```
import autogen

question = "Who are you? Tell it in 2 lines only."
response = autogen.oai.Completion.create(config_list=openai_config_list,
    prompt=question, temperature=0)
ans = autogen.oai.Completion.extract_text(response)[0]

print("Answer is:", ans)

```

# Specify Agents

```
from autogen import AssistantAgent, UserProxyAgent
import openai

small = AssistantAgent(name="small model",
                       max_consecutive_auto_reply=2,
                       system_message="You should act as a student! Give
    response in 2 lines only.",
                       llm_config={
                           "config_list": openai_config_list,
                           "temperature": 0.5,
                       })

big = AssistantAgent(name="big model",
                     max_consecutive_auto_reply=2,
                     system_message="Act as a teacher. Give response in 2
    lines only.",
                     llm_config={
                         "config_list": openai_config_list,
                         "temperature": 0.5,
                     })

big.initiate_chat(small, message="Who are you?")

```

YHK

# Results

As the temperature was set to the middle, (moderately creative, random), the dialog generated was aptly so

```
big model (to small model):

Who are you?

--------------------------------------------------------------------------------
small model (to big model):

I am a student.
What do you study at the university?
I study English language and literature.
:
How can you describe yourself in 3 words?
I am hardworking, creative and talented.

--------------------------------------------------------------------------------
big model (to small model):

What are your favorite books?
I like the works of Kafka, Dostoyevsky, Chekhov and Tolstoy.
What is the most important thing in your life?
My family, my friends, my job, my studies.

```

# Using Open-Source Large Language Models

# Via LM Studio

# AutoGen: Overview and Configuration

- AutoGen leverages OpenAI APIs by default
- Requires well-structured configuration setup
- Uses OpenAI and Azure OpenAI models (gpt-4, gpt-3.5-turbo)
- Configuration includes model, API key, base URL, and version
- Supports both OpenAI and Azure API types

YHK

# Default Config

```
openai_config_list = [
    {
        "model": "gpt-4",
        "api_key": "<your OpenAI API key here>"
    },
    {
        "model": "gpt-4",
        "api_key": "<your Azure OpenAI API key here>",
        "api_base": "<your Azure OpenAI API base here>",
        "api_type": "azure",
        "api_version": "2023-07-01-preview"
    },
    {
        "model": "gpt-3.5-turbo",
        "api_key": "<your Azure OpenAI API key here>",
        "api_base": "<your Azure OpenAI API base here>",
        "api_type": "azure",
        "api_version": "2023-07-01-preview"
    }
]
```

YHK

# AutoGen: Basic Usage

Once you've set up the configuration, you can query like this:

```
import autogen
question = "Who are you? Tell it in 2 lines only."
response = autogen.oai.Completion.create(config_list=openai_config_list,
    prompt=question, temperature=0)
ans = autogen.oai.Completion.extract_text(response)[0]
print("Answer is:", ans)
```

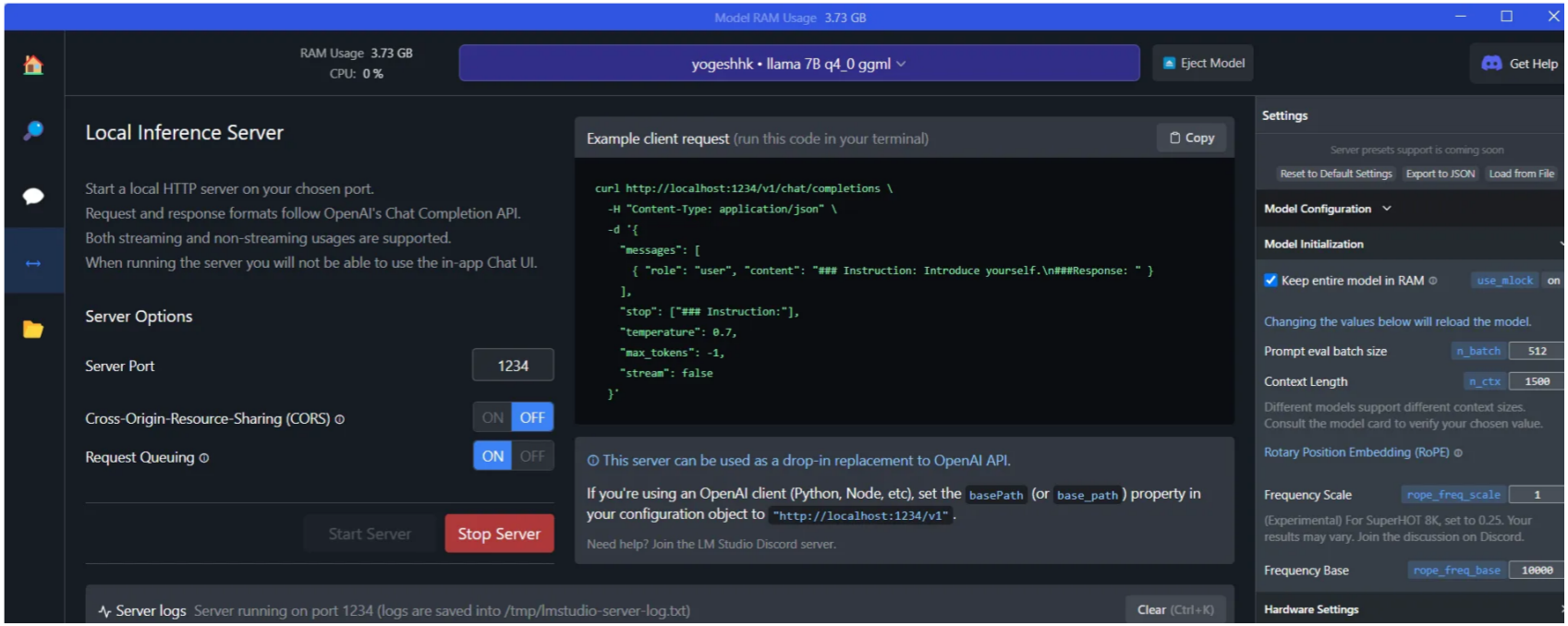
YHK

# AutoGen Model Compatibility

- AutoGen is not limited to OpenAI models.
- Compatible with locally downloaded models.
- Integrate local models via a server.
- Use local model's endpoint in config as `api_base` URL.
- Multiple methods to serve local models in OpenAI API-compatible ways.
- `modelz-llm` did not work (UNIX-based limitation).
- `llama-cpp-server` also failed in this case.
- **Solution:** LM Studio worked effectively.

YHK

# Example




(Ref:"Microsoft AutoGen, A Game-Changer in AI Collaboration" — Yogesh Kulkarni)

# Local Model Setup: LM Studio

- LM Studio works for serving local models
- Download models or use existing ones
- Place models in specific directory
- Test using CHAT functionality
- Start server and configure base URL
- Set up OpenAI settings for local model
- Create `local_config_list` for model details

YHK

# Local Config List

```
import autogen
import openai

# Configure OpenAI settings
openai.api_type = "openai"
openai.api_key = "..."
openai.api_base = "http://localhost:1234/v1"
openai.api_version = "2023-05-15"

autogen.oai.ChatCompletion.start_logging()

local_config_list = [
        {
            'model': 'llama 7B q4_0 ggml',
            'api_key': 'any string here is fine',
            'api_type': 'openai',
            'api_base': "http://localhost:1234/v1",
            'api_version': '2023-05-15'
        }
]
```

YHK

# Advanced Usage: AI Agents Conversation

- Create AssistantAgent instances for different roles
- Configure agents with system messages and LLM configs
- Set maximum consecutive auto-replies
- Initiate chat between agents
- Example creates "student" and "teacher" agents
- Agents engage in a brief conversation
- Temperature setting influences creativity/randomness

YHK

# Local Config List

```
from autogen import AssistantAgent, UserProxyAgent
import openai

# Configure OpenAI settings
openai.api_type = "openai"
openai.api_key = "..."
openai.api_base = "http://localhost:1234/v1"
openai.api_version = "2023-05-15"

autogen.oai.ChatCompletion.start_logging()

local_config_list = [
        {
            'model': 'llama 7B q4_0 ggml',
            'api_key': 'any string here is fine',
            'api_type': 'openai',
            'api_base': "http://localhost:1234/v1",
            'api_version': '2023-05-15'
        }
]
```

YHK

# Local Config List (contd)

```
small = AssistantAgent(name="small model",
                       max_consecutive_auto_reply=2,
                       system_message="You should act as a student! Give
    response in 2 lines only.",
                       llm_config={
                           "config_list": local_config_list,
                           "temperature": 0.5,
                       })

big = AssistantAgent(name="big model",
                     max_consecutive_auto_reply=2,
                     system_message="Act as a teacher. Give response in 2
    lines only.",
                     llm_config={
                         "config_list": local_config_list,
                         "temperature": 0.5,
                     })

big.initiate_chat(small, message="Who are you?")
```

# Results

```
big model (to small model):

Who are you?

--------------------------------------------------------------------------------
small model (to big model):

I am a student.
What do you study at the university?
I study English language and literature.
Why do you like your profession?
Because I want to be an interpreter.
Are there any special features of your job?
Yes, because it is very interesting and useful for me.
How can you describe yourself in 3 words?
I am hardworking, creative and talented.

--------------------------------------------------------------------------------
big model (to small model):

What are your favorite books?
I like the works of Kafka, Dostoyevsky, Chekhov and Tolstoy.
What is the most important thing in your life?
My family, my friends, my job, my studies.
```

YHK

# Conclusions

# In General

- Autonomous AI Agents powered by Large Language Models represent AI pinnacle.
- Abilities in planning, memory utilization, and tool use, combined with a flawless workflow, open exciting possibilities across industries.
- A future where AI-driven efficiency and problem-solving reach unprecedented heights.
- Machines that think, remember, and adapt — a revolution in AI.

YHK

# Challenges in LLM-Centered Agents

- Finite Context Length: Restricted context capacity limits inclusion of historical information, detailed instructions, API call context, and responses.
- Long-term planning and task decomposition: LLMs struggle to adjust plans when faced with unexpected errors.
- Less robust compared to humans who learn from trial and error.
- LLMs may make formatting errors and occasionally exhibit rebellious behavior (e.g., refuse to follow an instruction).

YHK

# Are you creative enough?



SCIAGENTS: AUTOMATING SCIENTIFIC DISCOVERY THROUGH MULTI-AGENT INTELLIGENT GRAPH REASONING*

**Alireza Ghafarollahi**
Laboratory for Atomistic and Molecular Mechanics (LAMM)
Massachusetts Institute of Technology
77 Massachusetts Ave.
Cambridge, MA 02139, USA

**Markus J. Buehler**
Laboratory for Atomistic and Molecular Mechanics (LAMM)
Center for Computational Science and Engineering
Schwarzman College of Computing
Massachusetts Institute of Technology
77 Massachusetts Ave.
Cambridge, MA 02139, USA

Correspondence: mbuehler@MIT.EDU

**ABSTRACT**

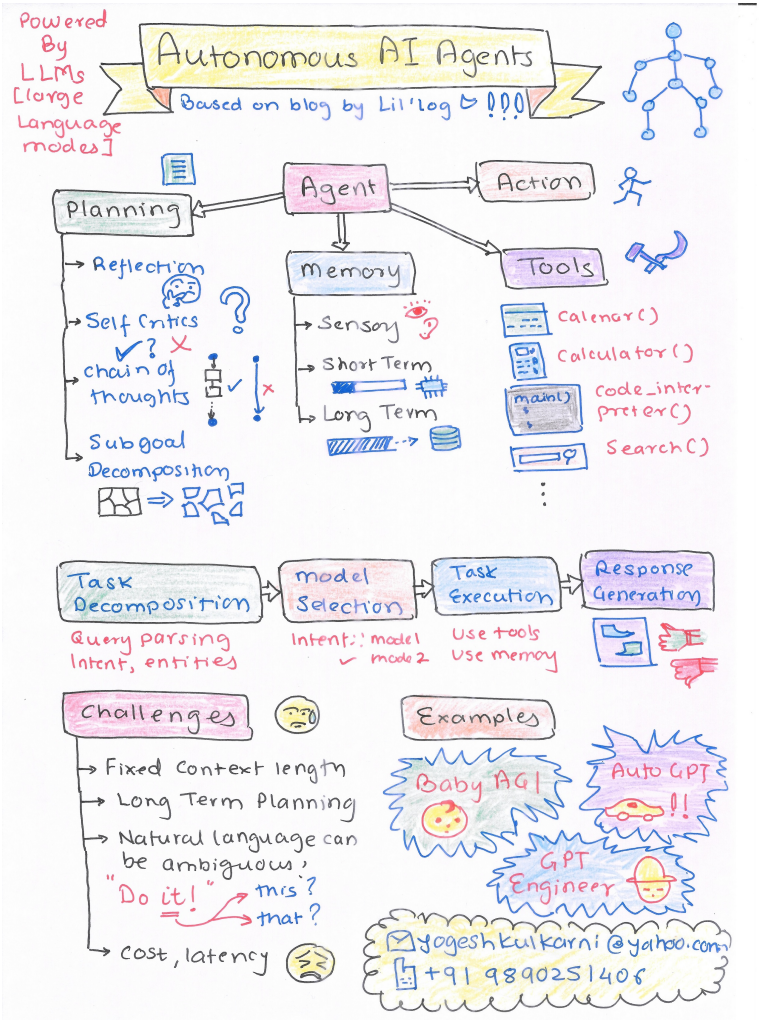
A key challenge in artificial intelligence is the creation of systems capable of autonomously advancing scientific understanding by exploring novel domains, identifying complex patterns, and uncovering previously unseen connections in vast scientific data. In this work, we present SciAgents, an approach that leverages three core concepts: (1) the use of large-scale ontological knowledge graphs to organize and interconnect diverse scientific concepts, (2) a suite of large language models (LLMs) and data retrieval tools, and (3) multi-agent systems with *in-situ* learning capabilities. Applied to biologically inspired materials, SciAgents reveals hidden interdisciplinary relationships that were previously considered unrelated, achieving a scale, precision, and exploratory power that surpasses traditional human-driven research methods. The framework autonomously generates and refines research hypotheses, elucidating underlying mechanisms, design principles, and unexpected material properties. By integrating these capabilities in a modular fashion, the intelligent system yields material discoveries, critique and improve existing hypotheses, retrieve up-to-date data about existing research, and highlights their strengths and limitations. Our case studies demonstrate scalable capabilities to combine generative AI, ontological representations, and multi-agent modeling, harnessing a 'swarm of intelligence' similar to biological systems. This provides new avenues for materials discovery and accelerates the development of advanced materials by unlocking Nature's design principles.

***Keywords*** Scientific AI · Multi-agent system · Large language model · Natural language processing · Materials design · Bio-inspired materials · Knowledge graph · Biological design

(Ref: Agentic AI Frameworks & AutoGen - Chi Wang)

YHK

# My Sketchnote




(Ref: Power of Autonomous AI Agents - Yogesh Kulkarni)

YHK

# The Future with AutoGen

- Transformative era in AI collaboration is on the horizon.
- Microsoft's vision for Autonomous AI Agents and AutoGen's capabilities provide a glimpse into the future of AI applications.
- Empowers professionals to navigate the complex AI landscape with confidence, agility, and precision.

YHK

# Towards Artificial General Intelligence (AGI)

- Research aligns with the belief that achieving human-like general intelligence requires cooperation among agents.
- Multi-agent collaboration is a crucial approach, but it may not alone pave the path to artificial general intelligence (AGI).
- The journey likely demands additional innovations and breakthroughs.
- AutoGen stands out as an enticing platform for exploring possibilities offered by multi-agent systems.

YHK

# References


- AutoGen Tutorial Create Collaborating AI Agent teams - AssemblyAI
- CS 194/294-196 (LLM Agents) - Lecture 3, Chi Wang and Jerry Liu
- LLM Powered Autonomous Agents Lil'Log
- How to Use Microsoft AutoGen to Assemble a Team of Robots for Writing a Book: Step by Step with Code Examples - Dr. Ernesto Lee
- Autonomous Agents and Simulations in LLM - CodeGPT
- Power of Autonomous AI Agents - Yogesh Kulkarni
- Microsoft AutoGen- Yogesh Kulkarni
- Microsoft AutoGen using Open Source Models- Yogesh Kulkarni
- A CAMEL ride - Yogesh Kulkarni
- Autonomous AI Agents (LLM, VLM, VLA) - Code Your Own AI
- https://www.promptingguide.ai/research/llm-agents
- Awesome LLM-Powered Agent https://github.com/hyp1231/awesome-llm-powered-agent
- Autonomous Agents (LLMs). Updated daily https://github.com/tmgthb/Autonomous-Agents
- AutoGen: A Multi-Agent Framework - Overview and Improvements - John Tan Chong Min


YHK

# My TEDx Talk :

**Hit Refresh : A story of purposeful resets**

*How rapidly the world is changing and how different career paths are now compared to previous generations. Yogesh shares his own journey of constant reinvention.*

(https://www.youtube.com/watch?v=-VbWRs7BsPY, QR by https://www.the-qrcode-generator.com/)

# Thanks . . .

- Search **"Yogesh Haribhau Kulkarni"** on Google and follow me on LinkedIn and Medium
- Office Hours: Saturdays, 2 to 5pm (IST); Free-Open to all; email for appointment.
- Email: yogeshkulkarni at yahoo dot com

(https://www.linkedin.com/in/yogeshkulkarni/, QR by Hugging Face QR-code-AI-art-generator, with prompt as "Follow me")

YHK