ChatGPT

In-Depth Technical Report: Codebases, AI Agents, and Assistant Systems

This report explores advanced software and AI agent concepts relevant to modern startups. Each section covers historical background, definitions and distinctions, real-world examples, and a Python code sketch illustrating the core idea.

1. Codebase

Background: A *codebase* is the complete collection of source code files for a software project. Early on, programmers managed code simply as files on disk. With the rise of version control in the 1970s–80s (e.g. RCS, CVS) and later distributed systems (Git, Mercurial), organized code repositories became standard. Today, large projects may use a single **monolithic codebase** or many **distributed repositories**. Monorepos (single large repository) simplify atomic changes and dependency management, while multi-repo setups isolate components but complicate cross-project refactoring [1] [2]. For example, Google maintains an immense monolithic repository (reported ~1 billion files), whereas the Linux kernel uses many distributed repos, reflecting different architecture and team needs [1] [3].

Definition & Distinctions: Formally, a codebase is "the complete body of source code for a software program, component or system" [4]. It includes all human-written files needed to build the software (source files, configuration, etc.) [5] [4]. Generated files or binaries are generally excluded, since they can be rebuilt. A codebase is typically managed in a version-control system (e.g. Git, Subversion), which tracks changes and enables collaboration [6]. Codebases differ from mere file sets by this management: for instance, a "monolithic codebase" has all code in one repo, whereas a "distributed codebase" uses many repos [1] [2]. These choices affect build systems, dependencies, and team workflows, and are independent of whether the final system is monolithic or microservice-based.

Examples: Real-world codebases include the Linux kernel and major open-source projects. For instance, Python's standard library is a codebase spanning hundreds of modules (all under one VCS repo). By contrast, Google's Android OS is split across many GitHub and Git repositories. Companies often keep private codebases: Microsoft's Windows and Office each are enormous codebases. On GitHub, projects like the Pytest testing framework have around 600+ source files [3]. Codebases also include DevOps scripts and documentation (e.g. `README.md`, CI config files) [4].

Sample Implementation: In Python we might interact with a codebase by scanning its files or using VCS commands. For example, the snippet below walks a directory tree (a codebase) and reports how many Python files and lines it contains. This illustrates programmatic inspection of a codebase.

```
import os

def summarize_codebase(path):
```

```
    total_files = 0
    total_lines = 0
    for root, dirs, files in os.walk(path):
        for file in files:
            if file.endswith('.py'):
                total_files += 1
                with open(os.path.join(root, file), 'r', encoding='utf-8',
errors='ignore') as f:
                    total_lines += sum(1 for _ in f)
    print(f"Codebase summary: {total_files} Python files, {total_lines} total
lines.")

# Example usage (uncomment and set a real path):
# summarize_codebase('/path/to/my_project')
```

This script reports the size of a Python codebase. In practice, teams often integrate similar scans into CI pipelines or IDE tools to analyze code size, complexity, or test coverage.

2. Artificial Consciousness

Background: *Artificial consciousness* (machine or synthetic consciousness) is a field at the intersection of AI, cognitive science, neuroscience and philosophy of mind [7]. It asks whether a computer could ever have subjective experiences or a sense of "self." The idea traces back to mid-20th-century debates on machine intelligence (e.g. Turing's work and discussions of feedback systems) [8]. Early AI pioneers like Minsky discussed "consciousness" in terms of replicating brain functions. In the 1980s–90s cognitive scientists (e.g. Baars) developed the Global Workspace Theory of consciousness, inspiring architectures like LIDA. Today, AC remains mostly theoretical and experimental; researchers design cognitive architectures or simulations to explore conditions for conscious-like processing [7].

Definition & Distinctions: Artificial consciousness is loosely defined as "consciousness hypothesized to be possible in artificial intelligence" [7]. In practice, it is the study of how to give a machine aspects of sentience or self-awareness. This differs from general AI in that AI can be intelligent without being conscious (e.g. a chess engine). AC specifically involves subjective qualities, awareness, or qualia. Some differentiate *access consciousness* (information available for reasoning) versus *phenomenal consciousness* (raw subjective experience) [9]. Unlike narrow AI agents, an "artificial conscious" system would, in theory, have an internal model or self-representation. Philosophers debate whether consciousness is substrate-dependent; functionalists argue a machine with the right structure could be conscious, while others say true conscious experience may be unattainable in silicon [10] [11].

Examples & Architectures: No widely-recognized real-world system is truly conscious. However, several research prototypes aim at conscious-like processing. For example, the LIDA (Learning Intelligent Distribution Agent) cognitive architecture implements Baars's global workspace: percepts enter a "workspace," the most relevant content is broadcast to all modules, then action is chosen [12]. Another approach, OpenCog (open-source by Ben Goertzel), includes virtual agents that learn language commands and control robots, aiming to produce human-like cognition [13]. Other proposals include Shanahan's architectures (simulated imagination combined with global workspace) and Lipson's self-modeling robots

[14] [15]. All remain experimental; by contrast, mainstream AI today (like ChatGPT) mimics understanding without any consensus as to real consciousness [16] [17].

Sample Implementation: As a conceptual demo, one can simulate a simple *global workspace*. For instance, different modules write to a shared workspace, which then broadcasts to others. This is the core idea behind many AC architectures (a "blackboard" or bus):

```
class GlobalWorkspace:
    def __init__(self):
        self.data = {}
    def broadcast(self, channel, message):
        self.data[channel] = message

# Example cognitive modules
def visual_module(workspace, image):
    # detect an object in the "image"
    obj = "cup"  # stub for detected object
    workspace.broadcast('vision', obj)

def planning_module(workspace):
    if 'vision' in workspace.data:
        print("Planning with visual input:", workspace.data['vision'])
        # form action plan based on 'cup'

# Simulate perception and processing
workspace = GlobalWorkspace()
visual_module(workspace, image_data=None)
planning_module(workspace)
```

Here the `GlobalWorkspace` holds information from various "modules." The visual module detects something and broadcasts it; the planning module reads it. This toy example illustrates the idea of a conscious workspace where salient information is shared and acted upon (as in LIDA [12]).

3. Personal Digital Assistant (PDA)

Background: A *Personal Digital Assistant (PDA)* originally meant a handheld device for personal information management. The first PDA was the **Psion Organiser I** (1984) [18], followed by later models with built-in calendars, address books, and memos. Apple coined "Personal Digital Assistant" at CES 1992 referring to the Apple Newton MessagePad [18]. In the late 1990s/2000s, companies like Palm (PalmPilot, Tungsten) and Handspring (Visor) dominated PDAs, adding features like handwriting recognition and wireless connectivity. By the mid-2000s, most PDAs evolved into smartphones. For example, early smartphones like the IBM Simon (1994) combined PDA functions with a cellphone. Today, nearly all PDA features (calendar, notes, contacts) are provided by modern smartphones [19].

Definition & Distinctions: Technically, a PDA is "a multi-purpose mobile device which functions as a personal information manager" [19]. Key features include a touchscreen (or keyboard), organizer software

(calendar, tasks, contacts), and often wireless connectivity. Unlike a *Virtual Assistant* (software agent), a PDA refers to hardware. The PDA term is now largely historical: modern smartphones on iOS/Android are its successors [19]. (Indeed, Wikipedia notes "for the modern successor, see smartphone" [20].) PDAs differ from voice-enabled assistants: a PDA was a stand-alone device, whereas today's "assistants" (Siri/Alexa) are software on phones or cloud services.

Examples: Classic PDA examples are the Apple Newton MessagePad (1993) and the PalmPilot series (launched 1996). The IBM Simon is often cited as a precursor smartphone (combining PDA and cellphone). These devices stored personal schedules, notes, and contacts. Modern equivalents are smartphone apps (Google Calendar, Microsoft Outlook on mobile). While traditional PDAs are obsolete, the term sometimes resurfaces in corporate contexts to refer to any on-demand personal task service. For the purposes of digital assistants, note that PDAs (hardware) are distinct from "virtual assistants" (software bots) [20].

Sample Implementation: A PDA's core function is managing personal information. Below is a simple Python example of a mini "personal information manager" that stores tasks and contacts in memory, mimicking basic PDA features:

```
tasks = {}
contacts = {}

def add_task(time, description):
    tasks[time] = description

def add_contact(name, phone):
    contacts[name] = phone

# Example usage
add_task("09:00", "Team meeting")
add_contact("Alice", "123-4567")
print("Tasks:", tasks)
print("Contacts:", contacts)
```

This script keeps a calendar of tasks and a contacts list, similar to PDA apps. A real PDA program would use persistent storage (e.g. writing to a file or database), but here we demonstrate the concept of storing and retrieving personal data.

4. Virtual Assistant

Background: *Virtual assistants (VAs)* are software agents that perform tasks for users based on input commands or questions [21]. The concept dates back decades in AI history. Early milestones include the **ELIZA** chatbot (1966) which simulated conversation [22]. Voice recognition technology advanced in the mid-20th century (e.g. Bell Labs' Audrey in 1952 and IBM's Shoebox in 1962 [23]). The 2000s saw Internet-enabled assistants: for example, Apple's Siri (2011), which leveraged speech recognition and AI to answer queries. Today's VAs are often cloud-based, using machine learning and big data. Major tech firms deploy voice/text assistants globally (see below).

Definition & Distinctions: A virtual assistant is "a software agent that can perform a range of tasks or services for a user based on user input (such as commands or questions) [21]." Common inputs include voice and text. VAs can interpret natural language, maintain context, and take actions like setting calendar events or controlling devices [21]. They differ from simple chatbots in sophistication: most virtual assistants combine speech recognition, NLP, and task automation. Unlike a *Personal Digital Assistant* (hardware), a VA is purely software (though often run on personal devices). Unlike a human personal assistant, a VA is automated and can scale easily. (Wikipedia even clarifies "not to be confused with personal digital assistant" [24]; PDAs were hardware organizers.)

Examples: Popular consumer virtual assistants include Apple's Siri, Amazon's Alexa, Google Assistant, and Samsung's Bixby [25]. These can answer questions, control smart home gadgets, send messages, and more via voice. Recent AI chatbots like ChatGPT and Microsoft Copilot blur lines, acting as conversational agents for information and commands (ChatGPT's underlying technology increased interest in VAs [26]). In enterprise, VAs appear as customer service bots (e.g. answering FAQs on websites) or productivity assistants (e.g. scheduling meetings).

Typical tasks for virtual assistants include: checking and creating calendar events, setting reminders, retrieving information (weather, news), controlling IoT devices (lights, thermostats), and handling communications (calls, texts) [21] [25]. For instance, you can ask, "Set a meeting tomorrow at 10 AM" or "Play music by Beethoven," and the assistant executes these tasks.

Sample Implementation: A simple virtual assistant loop might parse user input and respond. The example below handles a couple of command keywords. In a real system, one would use NLP libraries or voice APIs, but here we use basic string checks to illustrate the idea:

```
def virtual_assistant(command):
    command = command.lower()
    if "time" in command:
        print("It is 12:00 PM")  # stubbed response
    elif "weather" in command:
        print("Today's forecast: sunny, 75°F")
    else:
        print("I'm sorry, I don't understand the request.")

# Example usage
virtual_assistant("What's the time?")
virtual_assistant("How is the weather today?")
```

This minimal assistant recognizes "time" and "weather" in the input and replies. A production VA would use machine-learning models or rule-based engines to interpret a wider range of language and connect to real data sources.

5. Personal Assistant (Intelligent Personal Assistant)

Background: The term *Personal Assistant*, especially in AI, often refers to an intelligent agent tailored to an individual's needs. Sometimes called an **Intelligent Personal Assistant (IPA)**, it merges concepts of virtual

assistants and personalization. This notion evolved alongside VAs: the early text-based ELIZA was even dubbed an "intelligent personal assistant" in one analysis because it enabled human-like conversation decades before voice-based systems [27]. Modern IPAs leverage user data (contacts, calendar, preferences) and AI to proactively assist. For example, the vision of AI assistants has been that "everyone gets a 'white-collar' personal assistant," as Bill Gates remarked. Many consumer products (Siri, Google Assistant, Alexa) are effectively IPAs.

Definition & Distinctions: An IPA is "an AI-driven software program that helps people complete basic tasks... typically [via] natural language voice commands and location awareness" [28]. It is essentially a personalized virtual assistant, with access to the user's personal information and context (e.g. name, schedule, routines). Some define it as a *smart* or *virtual* assistant that actively manages personal tasks (calendar, email, reminders, etc.) [28]. The distinction from a generic virtual assistant is nuance: a personal assistant is explicitly user-centric and may maintain state about the user over time. Unlike a human PA, it is automated and accessible 24/7. Unlike a pure chatbot, it integrates with personal data (e.g. "Hey assistant, what's on my calendar?") and may have proactive suggestions.

Examples: Current IPAs include Siri (Apple), Google Assistant, Microsoft's Cortana/Copilot, and Amazon's Alexa, which all tie into a user's account and device ecosystem. In business, IPAs can be devices like Amazon Echo Show managing personal tasks at home, or software like Microsoft Viva embedding into the workplace. Emerging IPAs include specialized "AI copilots" (e.g. GitHub Copilot for coding, which assists a developer). Use cases are similar to VAs: answering queries, setting reminders, drafting messages, and controlling devices – but with deeper personalization and multi-step task automation. For example, an IPA might not only set a meeting (VA functionality) but also proactively suggest meeting times based on a user's habits and email contents.

Sample Implementation: A personal assistant program might maintain user-specific state. For instance, it can store the user's name and tasks, then use them in messages:

```
state = {"name": "Alice", "tasks": ["Email Bob at 9AM"]}
# Greet user by name and summarize tasks
print(f"Good morning, {state['name']}!")
print(f"You have {len(state['tasks'])} task(s) today: {state['tasks']}")

# Add a new task as the assistant
new_task = "Prepare report by 5PM"
state["tasks"].append(new_task)
print("Updated tasks:", state["tasks"])
```

This code greets Alice using her stored name and lists her tasks, then adds a new task. A full IPA would integrate with persistent calendars, email APIs, and use context (time of day, location), but this illustrates personalization and memory of user data.

6. Autonomous Agent

Background: An *autonomous agent* is an AI system that operates with a high degree of independence. Early AI research into agents emphasized autonomy: for example, Pattie Maes (1995) defined autonomous

agents as systems that inhabit an environment and act on it to fulfill goals [29]. These agents emerged in research on robotics and adaptive software in the 1990s. Today, the concept underlies applications from self-driving cars to robotic vacuum cleaners. The term also appears in software (e.g. bots in networks) and cognitive science. A key historical note: Franklin and Graesser (1997) coined the widely cited notion that an autonomous agent acts over time to pursue its own agenda [30].

Definition & Distinctions: Wikipedia defines "an autonomous agent [as] an AI system that can perform complex tasks independently" [31]. In other words, it senses its environment and makes decisions and actions without continuous human input. Franklin/Graesser further elaborate: it is "a system situated within and a part of an environment that senses that environment and acts on it, over time, in pursuit of its own agenda" [30]. This aligns closely with the broader "intelligent agent" concept, but emphasizes *independence*. (Some texts distinguish "autonomous agent" as a synonym for "intelligent agent".) Key features include *adaptation* (learning from experience) and *goal-directedness* (it has objectives it tries to fulfill) [30].

Examples: Real-world autonomous agents are everywhere. Notable examples include: - **Self-driving cars:** These vehicles use sensors and AI to perceive roads and drive without a human driver [32]. - **Autonomous drones/robots:** E.g. warehouse robots that navigate and manage inventory. - **Robotic vacuum cleaners:** These household robots map rooms and clean floors on schedules. - **Game AIs:** Non-player characters in video games that perceive the game state and act to achieve game goals. - **Software bots:** Examples include email filters (autonomously sorting messages) and recommendation systems (suggesting content based on user behavior) [32]. - **Smart assistants:** Even chatbots can be considered autonomous agents when they handle queries without a human dispatcher [33]. The BotPenguin blog notes that autonomous agents operate without constant guidance – for instance, a self-driving car "observes its surroundings, decides the best route, and navigates safely — all without human control" [32].

All these systems share that they "analyze data, make decisions, and act to achieve predefined goals" with minimal human intervention [32] [33].

Sample Implementation: We can illustrate an autonomous agent with a simple rule-based loop. Below is a toy "HungryAgent" that decides whether to eat or search based on random perceptions – it operates continuously without human input:

```
import random

class HungryAgent:
    def __init__(self):
        self.energy = 10

    def perceive(self):
        # Randomly perceive food or nothing
        return random.choice(['food', 'nothing'])

    def act(self, perception):
        if perception == 'food':
            self.energy += 5
            print("Found food: ate. Energy =", self.energy)
```

```
        else:
            self.energy -= 1
            print("No food: searching. Energy =", self.energy)

agent = HungryAgent()
for step in range(5):
    p = agent.perceive()
    agent.act(p)
```

In each step the agent *senses* its environment (`perceive`) and *acts* accordingly without any external commands. Its "goal" is to survive by keeping energy up. This loop demonstrates autonomy: the agent independently observes and chooses actions.

7. Agentic AI

Background: *Agentic AI* is a recently coined term (circa 2024) referring to AI systems organized as autonomous agents. It emphasizes AI that *"focuses on autonomous systems that can make decisions and perform tasks without human intervention"* [34]. The roots of this concept lie in decades of agent research: Turing's early AI ideas, Norbert Wiener's feedback systems, and later cognitive theories on agency (Bandura's work on human agency) [8]. The late 1990s saw the phrase "agent-based process management" used in enterprise IT. More recently (2020s), the explosion of large language models and reinforcement learning has given rise to new forms of agentic systems (for example, generative AI tools that autonomously plan multi-step tasks).

Definition & Distinctions: Agentic AI is essentially a subset of intelligent agents focused on **fully autonomous, goal-driven AI programs** [34] [35]. These agents typically *proactively* pursue objectives over extended periods, often learning and adapting as they go [35]. Unlike older systems like simple rule-based bots or RPA (which follow fixed scripts), agentic systems use machine learning to refine decisions [36] [37]. Reinforcement learning often plays a key role; as the Wikipedia article notes, agentic AI agents use RL to "learn best actions through trial-and-error," adjusting to complex data [37]. In short, agentic AI = autonomous AI agents + learning/adaptation. It is closely tied to "digital process automation" and is being marketed in enterprise contexts (e.g. IBM's and Salesforce's agentic platforms).

Examples: Agentic AI appears in emerging products and research. For instance: - **Automated coding agents:** Tools like GitHub Copilot (with AI completion) or newer systems that not only suggest code but autonomously refactor or generate functions. - **Generative AI assistants:** Chatbots with plugins (AutoGPT and similar) that can formulate plans and execute multiple API calls on their own. - **Autonomous monitoring:** Cybersecurity agents that continuously learn patterns and autonomously respond to threats. - **Business process agents:** Systems that monitor business data and optimize decisions (e.g. autonomous scheduling of supply chains). The Wikipedia page lists applications in software development, customer support, cybersecurity, and business intelligence [38]. Historic examples include IBM Deep Blue (chess) and Google's AlphaZero (games) as early goal-driven AI successes [39]. More recently, large language models (GPT-4, Bard) combined with planning layers demonstrate agentic behavior (e.g. drafting plans, then carrying them out through tools) [39]. Salesforce's Agentforce is a commercial platform explicitly for deploying such AI agents [39].

Sample Implementation: An agentic AI often uses reinforcement learning or decision modeling. Here is a toy example of a simple Q-learning agent that chooses actions and updates values based on rewards (simulated randomly). This illustrates the learning aspect of agentic AI:

```
import random

class SimpleAgenticAI:
    def __init__(self):
        self.q_values = {'action1': 0.0, 'action2': 0.0}

    def choose_action(self):
        # randomly choose an action (could use more advanced policy)
        return random.choice(list(self.q_values.keys()))

    def update(self, action, reward):
        # basic Q-learning update for demonstration
        learning_rate = 0.1
        current_q = self.q_values[action]
        self.q_values[action] = current_q + learning_rate * (reward - current_q)

agent = SimpleAgenticAI()
for episode in range(5):
    action = agent.choose_action()
    reward = random.choice([0, 1])  # simulate a reward signal
    print(f"Action: {action}, Reward: {reward}")
    agent.update(action, reward)
print("Learned Q-values:", agent.q_values)
```

This agent randomly picks an action and receives a reward, then updates its internal Q-values (representing its learned preferences) [37]. Although trivial, it embodies the idea of an AI agent improving its behavior through interaction — a hallmark of agentic AI.

8. Intelligent Agent

Figure: A simple reflex intelligent agent. Sensors read the environment state, an agent program decides an action (e.g. Clean or Move in the vacuum world), and actuators execute it.

Background: The concept of an *intelligent agent* is foundational in AI. It dates back to classic AI textbooks (Russell & Norvig define AI as the study of *"rational agents"*). The idea is that any computer program or entity that perceives its environment and takes actions to achieve goals can be considered an intelligent agent [40]. This abstraction became widespread in the late 20th century. Early AI research in the 1950s–60s (like Newell & Simon's General Problem Solver) implicitly used agent concepts, and by the 1980s explicit agent architectures (reactive, belief-desire-intention models) were developed. The intelligent agent paradigm unified many AI subfields under the lens of agents interacting with environments.

Definition & Distinctions: Formally, an intelligent agent *"perceives its environment, takes actions autonomously to achieve goals, and may improve its performance through learning or knowledge acquisition"* [40]. Key terms: - **Perceive:** use sensors or input to observe the world. - **Act:** use effectors or output to change the environment. - **Goal:** internal objective or utility the agent tries to maximize.
Unlike a generic program, an intelligent agent is *goal-directed* and adaptive. The Wikipedia entry emphasizes goal-oriented behavior: "Leading AI textbooks define AI as the study and design of intelligent agents" [40]. A *rational agent* is one that chooses actions expected to best fulfill its objective (per a performance measure) [41]. All agentic and autonomous agents are subsets of intelligent agents by this view. The term "agent" is used broadly: even a thermostat can be an intelligent agent (it perceives temperature, acts to heat/cool, has a goal range) [42], as can a person or a company if they meet the criteria [42].

Examples: Intelligent agents abound in computing. Examples range from trivial to very complex: - **Simple reflex agents:** e.g., a thermostat or light sensor (if-too-cold then turn on heater). The figure above shows a "vacuum-cleaner" reflex agent that cleans if the current square is dirty, or moves to another square otherwise [40]. - **Deliberative agents:** e.g., a self-driving car planning routes, or a chess engine planning moves (Deep Blue defeated Kasparov in 1997 as an intelligent agent optimized for chess [39]). - **Learning agents:** e.g., spam filters that update filters from incoming email, or adaptive web services personalizing content. - **Human and organizations:** Even humans are intelligent agents in the broad AI sense, as are firms or governments in economic modeling [42]. - **Software systems:** Web crawlers, recommendation engines, trading bots, game-playing bots, robotic arms, and smart assistants (Siri/Google) are all instances of intelligent agents – they sense inputs (web pages, user queries, market data, sensor data) and act to achieve some goal (search, answer, profit, assembly) [40] [42].

Each of the previously discussed systems (virtual assistant, autonomous vehicle, etc.) qualifies as an intelligent agent under this definition, since they perceive an environment and take goal-directed actions. The agent paradigm thus unifies them: what makes an agent *intelligent* is its use of knowledge or learning to maximize success. For example, an intelligent vacuum agent might learn which rooms get dirty fastest and adapt its cleaning schedule.

Sample Implementation: A classic AI example is the *vacuum world agent* (from Russell & Norvig). The Python code below models a two-room environment (`A`, `B`) and a simple intelligent agent that cleans dirty rooms and moves between rooms:

```
# Vacuum World Intelligent Agent Example
env = {'A': True, 'B': True}  # True means the room is dirty

class VacuumAgent:
    def __init__(self):
        self.location = 'A'
        self.performance = 0

    def perceive(self):
        # Return (current_location, is_dirty)
        return (self.location, env[self.location])
```

```
    def act(self, percept):
        loc, dirty = percept
        if dirty:
            # Clean if dirty
            env[loc] = False
            self.performance += 10
            return f"Cleaned {loc}"
        else:
            # Move to the other room
            new_loc = 'B' if loc == 'A' else 'A'
            self.location = new_loc
            self.performance -= 1
            return f"Moved to {new_loc}"

agent = VacuumAgent()
for step in range(4):
    p = agent.perceive()
    action = agent.act(p)
    print(f"Step {step}: {action}")
print("Environment state:", env)
print("Performance:", agent.performance)
```

This agent operates autonomously: it senses whether its current location is dirty and either cleans it or moves to the other location. Its performance score increments for cleaning (achieving its goal of cleanliness). This demonstrates a *goal-driven intelligent agent* using feedback from the environment to act rationally.

Sources: The definitions and examples above are supported by AI literature and references [40] [42] [39]. The agent paradigm is central to modern AI and software engineering, underlying many intelligent systems in practice.

---

[1] [5] [6] Codebase - Wikipedia
https://en.wikipedia.org/wiki/Codebase

[2] [3] [4] What is a codebase (code base)? – TechTarget Definition
https://www.techtarget.com/whatis/definition/codebase-code-base

[7] [9] [10] [11] [12] [13] [14] [15] [16] [17] Artificial consciousness - Wikipedia
https://en.wikipedia.org/wiki/Artificial_consciousness

[8] [34] [36] [37] [38] [39] Agentic AI - Wikipedia
https://en.wikipedia.org/wiki/Agentic_AI

[18] [19] [20] Personal digital assistant - Wikipedia
https://en.wikipedia.org/wiki/Personal_digital_assistant

[21] [22] [23] [24] [25] [26] Virtual assistant - Wikipedia
https://en.wikipedia.org/wiki/Virtual_assistant

27 28 Intelligent Personal Assistants (IPA): Examples and Use Cases
https://www.enterpriseaiworld.com/Articles/Editorial/Features/Intelligent-Personal-Assistants-(IPA)-Examples-and-Use-Cases-163787.aspx

29 30 31 Autonomous agent - Wikipedia
https://en.wikipedia.org/wiki/Autonomous_agent

32 33 Introduction to Autonomous AI Agents with Real life Examples
https://botpenguin.com/blogs/introduction-to-autonomous-ai-agents

35 40 41 42 Intelligent agent - Wikipedia
https://en.wikipedia.org/wiki/Intelligent_agent