ChatGPT

Integration von Agent-NN in n8n und Flowise

Hintergrund: Agent-NN Architektur und Komponenten

Agent-NN ist ein fortschrittliches Multi-Agenten-System mit modularer Architektur. Es ermöglicht die **Erstellung, Verwaltung und Optimierung spezialisierter Agenten** für verschiedene Wissensdomänen und integriert diverse LLM-Backends [1]. In der aktuellen *MCP-Version* von Agent-NN sind mehrere Microservices vorgesehen, die zusammenarbeiten. Laut Systemarchitektur gehören dazu insbesondere:

- **Task Dispatcher** – zentraler Orchestrator, der für jede Aufgabe den passenden Worker-Agent auswählt [2].
- **Agent Registry** – eine Registry/Datenbank aller verfügbaren Worker-Services (Agenten) [3].
- **Session Manager** – verwaltet Konversationskontext bzw. Sitzungsdaten, um den Verlauf von Nutzeranfragen nachzuverfolgen [3].
- **Worker Services** – die spezialisierten *Worker-Agenten*, die domänenspezifische Aufgaben ausführen [3]. Jeder Worker-Agent verfügt über eigenes Wissen oder Funktionen für sein Spezialgebiet (z. B. WebScraperAgent, ChatbotAgent etc. laut Code-Dokumentation [4] [5]).
- **Vector Store** – Vektordatenbank für semantische Suche, etwa Chroma oder ein In-Memory-Speicher mit Embeddings [6] [7]. Hier werden Wissensdokumente per Embeddings abgelegt und Ähnlichkeitssuchen ermöglicht.
- **LLM Gateway** – abstrahiert den Zugriff auf Sprachmodelle (OpenAI, lokale LLMs usw.) [8] [9], so dass Agenten über eine einheitliche Schnittstelle Prompts an verschiedene LLM-Backends senden können.
- **API Gateway** – ein optionaler Eintrittspunkt mit Authentifizierung und Routing für externe Anfragen [10] [11]. Nutzeranfragen (z. B. aus CLI oder Web) gelangen zunächst an diese API-Schicht, die dann an den Task Dispatcher und weitere Komponenten weiterleitet.

  Hinweis: Diese Microservices ersetzen die frühere monolithische *SupervisorAgent*-Architektur [12]. Zuvor übernahm ein `SupervisorAgent` intern die Auswahl der Worker-Agenten [5]. Im neuen Design werden diese Funktionen auf getrennte Services verteilt, was eine bessere Skalierbarkeit ermöglicht.

Zusätzlich zu den obigen Hauptkomponenten enthält Agent-NN diverse **Manager-Module** (z. B. `AgentManager`, `KnowledgeManager`, `CacheManager` etc.) und Hilfsklassen, die im Hintergrund für Logging, Performance-Tracking (MLflow) und Lernen sorgen [13] [14]. So existieren beispielsweise ein *HybridMatcher* zur Kombination von Embedding-Ähnlichkeit mit neuronalen Features für die Agentenwahl [15], ein *MetaLearner* für die lernbasierte Agentenauswahl [16] und ein *AgentCreator* zum automatischen Generieren neuer Agenten auf Basis von Aufgabenanforderungen [17]. Diese Logiken sind wichtig, um später **dynamisch Agenten zu erstellen**, wenn keine bestehende Spezialisierung passt [17]. Die Kernidee ist, dass ein Supervisor (bzw. Task Dispatcher) anhand von Ähnlichkeiten und gelernten Kriterien den optimalen Worker auswählt oder sogar neue Worker schafft, um komplexe Nutzeranfragen zu bearbeiten.

Integration von Agent-NN in n8n

n8n ist ein Workflow-Automation-Tool, das über Nodes (Knoten) verschiedenste Services und APIs visuell miteinander verbindet [18] [19]. Für eine vollständige Kompatibilität mit Agent-NN sollten die zentralen Komponenten und Funktionen von Agent-NN als **einzelne Nodes oder Teil-Workflows in n8n** verfügbar sein. Es bieten sich zwei Ansätze an:

1. Microservices als einzelne n8n-Nodes – Jeder relevante Dienst von Agent-NN wird durch einen eigenen Node in n8n repräsentiert. Diese Nodes würden über die REST-API von Agent-NN kommunizieren:

- *Agent-NN Task erstellen* – entspricht dem Task Dispatcher: Dieser Node nimmt eine Nutzeranfrage (Beschreibung der Aufgabe) entgegen und ruft den Agent-NN API-Endpunkt zum Erstellen/Ausführen eines Tasks auf [20]. Intern wählt Agent-NN dafür einen passenden Agenten aus und führt die Aufgabe aus [21] [22]. Der Node gibt das Resultat (Antwort des Agenten) und Metadaten zurück. (Die API stellt dafür z. B. POST `/tasks` bereit, die synchron ein `TaskResponse` mit Ergebnis liefert [20] [23].)
- *Agent-NN Task-Status abfragen* – falls Aufgaben asynchron verarbeitet würden, könnte ein separater Node den Status oder das Ergebnis via GET `/tasks/{task_id}` abrufen [24]. Laut aktuellem Code erfolgt die Ausführung aber direkt und das Ergebnis wird unmittelbar zurückgegeben [21] [22], sodass Polling evtl. nicht nötig ist.
- *Agent registrieren/erstellen* – Node zur Anlage neuer spezialisierter Agenten. Dies nutzt den POST `/agents` Endpoint von Agent-NN, um per Konfiguration einen neuen Worker-Agent anzulegen [25]. Der Node könnte Parameter wie Agentenname, Beschreibung, Wissensbasis etc. aufnehmen. Als Resultat käme eine Agenten-ID oder Config zurück [26].
- *Agenten verwalten* – Weitere Nodes könnten das Auflisten aller vorhandenen Agenten (GET `/agents` [27]) oder Abrufen von Status/Details eines Agenten (GET `/agents/{id}` [28]) ermöglichen. Damit ließe sich z. B. in n8n entscheiden, ob bereits ein passender Agent existiert oder ob ein neuer erstellt werden muss.
- *Modelle verwalten* – Agent-NN erlaubt auch das Registrieren/Laden von Modellen (POST `/models` etc. in den API Endpoints [29]). Ein n8n-Node könnte diese Funktionen ansprechen, falls für bestimmte Workflows z. B. gezielt ein lokales LLM geladen oder ein HuggingFace-Modell heruntergeladen werden soll [30].
- *Vector Store Node* – Zum Hinzufügen und Abfragen von Dokumenten im Vektorspeicher. Agent-NN hat dafür z. B. `VectorStoreService.add_document()` und `.query()` Methoden [31] [32], ggf. exponiert über eigene API-Routen. Ein Node könnte z. B. neue Wissensdokumente indexieren oder relevante Dokumente für eine Anfrage abrufen (ähnlich einem *Datenbank-Query-Node* in n8n).
- *LLM Gateway Node* – Dieser würde Agent-NNs LLM-Abstraktion nutzen. Beispielsweise könnte man einen Node haben, der einen Prompt via Agent-NN an das konfigurierte LLM schickt (OpenAI, lokale Modelle etc., konfiguriert über Agent-NN) und die rohe LLM-Antwort zurückliefert. Allerdings bietet n8n bereits eigene OpenAI- und LLM-Integrationen; dieser Node wäre primär relevant, wenn man Agent-NNs einheitliche Ansteuerung (inkl. z.B. LM Studio via `LMStudioClient` [33]) nutzen möchte.

Technisch würden diese Nodes jeweils einen HTTP Request an Agent-NN senden – n8n hat dafür entweder den generischen HTTP-Request-Node oder man erstellt dedizierte Nodes in TypeScript für wiederverwendbare Operations [34] [35]. Bei einer dedizierten Integration könnte man auch gleich passende **Credential-Typen** in n8n definieren (z. B. Basis-URL des Agent-NN-Servers, Auth-Token vom API Gateway), damit die Verbindung bequem konfigurierbar ist. Die API von Agent-NN erfordert beim Task- und Agenten-Management einen Bearer-Token (Login via `/token` Endpoint) für

Authentifizierung [36] [37] – dies ließe sich in n8n über ein Credential (z. B. OAuth2 oder Generic Token Auth) abbilden und im Node verwenden.

2. Zusammenfassende "Agent-NN-Agent" Node/Workflow – Neben den fein granularen Nodes wird gefordert, eine *große zusammenfassende Node* bereitzustellen, die den gesamten Agent-NN-Agenten zentral kapselt. Diese könnte als **einfacher Einstiegspunkt** dienen, um Agent-NN in andere Workflows einzubinden, ohne jedes Mal die Einzelschritte zu modellieren. Zwei Umsetzungsmöglichkeiten bieten sich an:

- **Monolithischer Custom Node:** Ein n8n-Node, der intern die Agent-NN Pipeline ausführt. Beispielsweise ein Node *"AgentNN Execute"*, dem man nur eine Benutzeranfrage übergibt und der intern beim Agent-NN-API-Gateway einen Task erstellt und auf das Ergebnis wartet. Der Node würde dann direkt die finale Antwort des Agenten als Output liefern (sowie evtl. Infos wie welcher Worker-Agent gewählt wurde). Dieser Ansatz nutzt im Grunde den oben beschriebenen Task-Endpoint und versteckt die Details. Vorteil: sehr einfache Wiederverwendung – man braucht nur diesen einen Node, um die gesamte Intelligenz von Agent-NN zu nutzen.
- **Sub-Workflow (verschachtelter Workflow):** n8n erlaubt das Kapseln von Teilprozessen als sogenannte Sub-Workflows. Man könnte einen Workflow definieren, der die Kette *Task erstellen -> Ergebnis prüfen -> ggf. nachfassen* umfasst. Dieser Sub-Workflow würde aus den einzelnen Agent-NN Nodes bestehen (Task, evtl. Polling, etc.) und als **ein Node** in anderen Workflows aufrufbar sein. So könnte man die Ablauflogik flexibel ändern, behält aber eine einfache Schnittstelle nach außen (Input = Nutzerprompt, Output = Agentenantwort).

Beide Varianten stellen sicher, dass ein Anwender, der z. B. in einem größeren Geschäftsprozess einen KI-Agenten einbinden will, nur eine einzige Einheit integrieren muss, anstatt sich um Details der Agentenkoordination zu kümmern. Die zusammenfassende Node könnte auch konfigurierbare Parameter bieten, z. B. *"Expertenmodus"* wo man bestimmte Optionen setzt (welche Tools der Agent nutzen darf, Zeitlimits, etc.), die dann an Agent-NN durchgereicht werden.

Hinweise & Best Practices für n8n:
- *LangChain-Integration:* n8n besitzt bereits eingebaute KI-Funktionen, die auf LangChain basieren (sogenannte AI Agent Nodes) [38]. Diese erlauben es, einen KI-Agenten mit Tools innerhalb eines Workflows zu betreiben. Für Agent-NN könnte man theoretisch versuchen, dessen Funktionalität in solche bestehenden Nodes zu integrieren – vermutlich ist es aber sinnvoller, Agent-NN als externen Service anzusprechen, da es eine eigenständige komplexe Logik mitbringt (inkl. eigenem LLM-Gateway, Vector Store etc.). Eine Integration über die offiziellen n8n-AI-Nodes würde bedeuten, die gesamte Logik von Agent-NN in die LangChain-Struktur zu übersetzen, was aufwändig wäre. Stattdessen ist die **REST-API-Kopplung** ein direkterer Weg.
- *Performance*: n8n ist primär ein Orchestrierungstool – rechenintensive Vorgänge sollten außerhalb (z. B. in Agent-NN selbst) ablaufen. Daher ist es gut, dass Agent-NN als separater Service arbeitet. Die Nodes initiieren Aufgaben oder holen Ergebnisse, aber das eigentliche LLM-Processing findet in Agent-NN statt. So bleibt n8n responsiv. Man kann in n8n ggf. mit *"Execute Workflow"* in Hintergrund oder *Async-Workflow* arbeiten, falls Agenten länger laufen.
- *Error Handling*: Es sollte berücksichtigt werden, Fehler klar zu handhaben. Beispielsweise, wenn der Agent-NN-Service nicht erreichbar ist oder eine Aufgabe fehlschlägt, sollten die Nodes entsprechende Fehlermeldungen liefern. n8n erlaubt Try-Catch oder Error-Workflows, was genutzt werden kann, um etwa einen zweiten Versuch zu starten oder einen Fallback-Agenten (z. B. einfacher ChatGPT-Node) zu verwenden, falls Agent-NN ausfällt.
- *Wartbarkeit*: Die Nutzung der API bedeutet, dass Verbesserungen in Agent-NN (z. B. neue Endpoints) ohne große Änderungen in n8n sofort nutzbar sind. Die Node-Implementierungen sollten möglichst generisch sein (z. B. *Agent-NN API Call* Node, wo man Endpoint/Method als Parameter wählt). Allerdings

erhöht zu generische Gestaltung die Komplexität für den Anwender. Ein Mittelweg wäre, die wichtigsten Aktionen als separate Nodes bereitzustellen und weniger häufige als Option oder via generischem HTTP-Node zu erledigen.

Integration von Agent-NN in Flowise

Flowise ist ein Low-Code-Tool zur Erstellung von LLM-gestützten Workflows, das auf LangChain.js basiert. Flowise bietet ein Baukastensystem mit vordefinierten Komponenten (*Nodes*) für LLMs, Tools, Agenten, VectorStores usw., und es lässt sich durch Custom Components erweitern [39]. Die Zielsetzung hier ist, Agent-NN in Flowise so einzubinden, dass **neue Agenten erstellt und verwaltet werden können**, die Agent-NN verwenden. Außerdem soll man Agent-NN's Fähigkeiten in Flowise-gestalteten Chatbots/Flows nutzen können.

Mögliche Schritte und Strategien:

- **Custom Nodes für Agent-NN**: Flowise erlaubt eigene Nodes durch Erweiterung des Komponenten-Systems (durch Erstellung neuer TS-Klassen im Projekt) [40] [41]. Ein sinnvoller Ansatz ist, analoge Nodes wie bei n8n zu entwickeln: z. B. *AgentNN CreateAgent*, *AgentNN ListAgents*, *AgentNN ExecuteTask*. Diese würden intern per `fetch` oder HTTP-Aufruf die Agent-NN-API ansprechen. Da Flowise in Node.js läuft, kann man mit `node-fetch` oder Axios Bibliothek in einem Custom Node die REST-Endpunkte von Agent-NN aufrufen [42]. Ein Beispiel: Ein *AgentNN-ExecuteTask* Node nimmt einen Prompt entgegen und ruft `POST /tasks` beim Agent-NN Service auf. Die zurückgegebene Antwort des Agenten (inkl. `result` Text und Metadaten) wird dann in Flowise an die nächsten Nodes weitergereicht (z. B. an einen Antwort-Formatter). Genauso könnte *AgentNN-CreateAgent* einen neuen Agenten anlegen, was im Flowise-Interface genutzt werden kann, um z. B. vor einer Unterhaltung bestimmte Agenten dynamisch bereitzustellen.

- **Verwendung bestehender Flowise-Knoten**: Alternativ oder ergänzend könnte man versuchen, vorhandene Flowise-Knoten zu nutzen. Flowise hat z. B. einen *API Loader* (für Dokumente) und *API Chain* Nodes (GET/POST Chains) [43], sowie die Möglichkeit OpenAPI-Spezifikationen zu laden (*OpenAPI* Chain). Falls Agent-NN eine OpenAPI-Dokumentation bereitstellt (FastAPI generiert ein Schema), könnte man theoretisch diese in Flowise importieren, so dass ein LLM-Agent in Flowise eigenständig die Agent-NN-API als Tool verwenden kann. Beispielsweise könnte man einen LangChain-**OpenAPI Tool** generieren, der es dem LLM erlaubt, je nach Bedarf `/agents` oder `/tasks` von Agent-NN aufzurufen. In Flowise ließe sich das als *OpenAPI Agent* Node konfigurieren. Dieses Vorgehen erfordert jedoch eine robuste Prompt- und Tools-Definition, damit das LLM korrekt mit der API interagiert – es ist fortgeschritten und weniger deterministisch.

- **Agent-NN als Flowise-Agent verwenden**: Ein anderer Ansatz ist, Agent-NN als Ganzes wie einen *Agenten* zu behandeln. Flowise definiert *Agents* allgemein als Systeme, die ein LLM als Reasoning Engine nutzen, um Aktionen/Tools auszuführen [44]. Flowise bringt Out-of-the-Box Integrationen für bekannte Agent-Frameworks mit – z. B. gibt es Nodes für **AutoGPT** und **BabyAGI**, die als autonome Agenten mit eigener Logik fungieren [45]. Dies zeigt, dass sich komplexe Agentensysteme in Flowise einfügen lassen. Für Agent-NN könnte man ähnlich einen *AgentNN Agent* Node konzipieren: intern würde dieser Node einen laufenden Agent-NN Supervisor-Agenten anstoßen. Praktisch würde auch das wieder auf einen API-Aufruf hinauslaufen (dem `/tasks`-Endpoint) – allerdings könnte man in der UI von Flowise diesen Node in die Kategorie *Agents* einordnen, mit passendem Icon und Beschreibung, sodass ein

Nutzer erkennt, dass hier ein externer Agentensystem genutzt wird (vergleichbar mit dem AutoGPT-Node). Vorteil: Der Node könnte die gesamte interne Logik kapseln (ähnlich der monolithischen Node in n8n). Der Flowise-User verbindet dann diesen Agent-NN-Agent Node mit anderen Komponenten, etwa vorgelagerten Dokumenten-Lade-Nodes (um Wissen bereitzustellen) und nachgelagerten Antwort-Displays.

- **Agentenverwaltung über UI**: Die Forderung, neue Agenten *mit Flowise* erstellen und verwalten zu können, legt nahe, dass man Flowise als *Frontend* für Agent-NNs Agenten verwendet. Ein denkbares Muster: Ein Flow, der einen vom Benutzer eingegebenen Agenten-Beschreibungswunsch entgegennimmt (z. B. *„Erstelle einen neuen Agenten für Finanzdaten-Analyse.“*), dann den Node *AgentNN CreateAgent* ausführt. Dessen Ausgabe (Agenten-ID oder Konfiguration) könnte man anzeigen oder weiterverwenden. Außerdem könnte ein *ListAgents* Node die verfügbaren Agenten ins UI bringen (Flowise hat ein Chat-Interface; man könnte z. B. mit einem *Select* oder *Dropdown* in einer benutzerdefinierten UI-Komponente die Liste anzeigen, aber das erfordert ggf. erweiterte UI-Anpassungen). Grundsätzlich aber ermöglicht Flowise's Visualisierung, Agent-NN zu steuern, ohne direkt API-Calls per Hand machen zu müssen.

Synergien mit Flowise-Konzepten: Interessant ist, dass Flowise selbst Multi-Agent-Patterns unterstützt. In der Dokumentation wird z. B. das *Supervisor-and-Workers* Muster vorgestellt [46] [47]. Dabei agiert ein **Supervisor-LLM** in Flowise als Orchestrator, der anhand von eingehenden Aufgaben entscheidet, welcher spezialisierte Worker (z. B. *Software Engineer* vs. *Code Reviewer* Agent) als nächstes antworten soll [48] [49]. Die Umsetzung erfolgt durch entsprechende Verkettung von LLM-Nodes und logischen Schleifen im Flowise-Workflow [50] [51]. Agent-NN verfolgt ein sehr ähnliches Ziel – allerdings übernimmt hier Agent-NNs interne Logik (teils regelbasiert, teils ML-basiert) die Rolle des Supervisors. Man könnte also sagen: **Agent-NN als Komponente in Flowise implementiert das Supervisor/Worker-Muster "out of the box"**. Für den Flowise-Nutzer vereinfacht das einiges: statt das Routing zwischen spezialisierten Agenten mit zahlreichen Flow-Nodes manuell abzubilden, könnte ein Agent-NN-Agent Node dies implizit erledigen. Flowise würde Agent-NN gewissermaßen als *Meta-Tool* nutzen, das intern wieder LLMs und Tools orchestriert. Dieser Aufbau kombiniert die Stärken beider Systeme – Flowise für die einfache Integration mit anderen Anwendungen (Datenbanken, E-Mail, Benutzer-Frontend etc.), und Agent-NN für die intelligente Entscheidungslogik bei komplexen Anfragen.

Technische Umsetzung in Flowise:
Die Implementation eines Custom-Nodes in Flowise erfordert TypeScript-Kenntnisse. Laut Entwickler-Dokumentation legt man im Repo unter `packages/components/nodes/...` einen neuen Node-Ordner an und definiert dort eine Klasse, die das `INode` Interface implementiert [40] [52]. Außerdem wird üblicherweise eine Core-Klasse angelegt, falls komplexe Logik nötig ist [53] [54] – im Falle eines Agent-NN-Tool-Nodes könnte man aber direkt im `init` oder `execute` die HTTP-Calls durchführen. Wichtig ist, den Node so zu gestalten, dass er die erforderlichen Inputs/Outputs aufnimmt: z. B. bräuchte *CreateAgent* als Input ein JSON oder Form-Felder für die Agentenkonfiguration (Name, Domain, evtl. Basismodell) und liefert als Output die Agenten-ID und Default-Config zurück [55]. Ein *ExecuteTask* Node nimmt einen Prompt (und optional eine Agenten-ID, falls man einen bestimmten Agenten forcen will) und gibt die Antwort sowie Metriken zurück [22] [56]. Mit solchen Custom-Nodes kann man Agent-NN vollständig in die Flowise-Oberfläche integrieren.

Sicherheit und Zugriff: Ähnlich wie bei n8n sollte man in Flowise auch die API-Zugriffsdaten managen. Flowise erlaubt das Speichern von API-Schlüsseln in Umgebungsvariablen oder Konfigurationsdateien (abhängig von Implementierung des Nodes). Der Agent-NN-Node müsste daher konfigurierbar sein, z. B. Basis-URL des Agent-NN Servers und ein Auth-Token. In self-hosted Umgebungen könnte man Flowise und Agent-NN im gleichen Netzwerk betreiben, um die Latenz gering zu halten.

Mögliche Tools, APIs und Frameworks für die Integration

Zusammenfassung der Integrationsansätze: Die Integration wird hauptsächlich über **REST-APIs** stattfinden, daher sind Tools/SDKs zur API-Kommunikation zentral. Sowohl n8n als auch Flowise haben interne Mechanismen dafür (n8n HTTP Request Node; in Flowise via Node-Fetch in Custom Code [42]). Dennoch gibt es weitere Optionen und Best Practices:

- **OpenAPI/Swagger**: Da Agent-NN auf FastAPI basiert, liegt automatisch ein OpenAPI-Schema vor. Dieses könnte genutzt werden, um Client-Code zu generieren (es existieren Generatoren für TypeScript/JavaScript-Clients aus OpenAPI). Alternativ kann man das Schema in Tools wie Postman oder Insomnia importieren, um die API zu testen, und in Flowise's OpenAPI Chain Node einbinden. Dies standardisiert die Schnittstelle.
- **n8n Nodes Starter**: Für n8n stellt das Team ein **Node Starter Template** bereit [34] [35], das die Entwicklung eigener Nodes erleichtert. Es enthält Beispiele für Node-Strukturen und die benötigte Projektkonfiguration, um den Custom Node als npm-Package zu veröffentlichen. Dieses Template wäre ein guter Ausgangspunkt, um Agent-NN Nodes zu entwickeln – man könnte damit einen *n8n-nodes-agentnn* Package bauen und im n8n einsetzen.
- **LangChain SDK**: Agent-NN nutzt zum Teil LangChain (z. B. Embeddings, Tools in `agentic_worker.py` [57]). Es lohnt sich zu prüfen, ob ein Teil der Funktionen als LangChain-Komponenten vorliegt. Falls ja, könnte man in Flowise eventuell direkt auf diese Komponenten zurückgreifen. Beispielsweise, wenn Agent-NN VectorStore auf Chroma basiert, könnte Flowise's bestehender Chroma-VectorStore-Node genutzt werden, um denselben Speicher anzusprechen (vorausgesetzt, man konfiguriert beide auf dieselbe DB). Allerdings sind viele Teile von Agent-NN sehr spezifisch (HybridMatcher, MetaLearner etc. haben kein direktes LangChain-Analogon). Daher wird man größtenteils die Agent-NN API als *Connector* behandeln.
- **Bestehende Connectors vergleichen**: Blick auf ähnliche Open-Source-Projekte zeigt, dass Integration oft via APIs oder Plugin-Schnittstellen erfolgt. Beispielsweise bietet OpenAI's Funktionen-API die Möglichkeit, Tools in Chatbots einzubinden – Agent-NN könnte perspektivisch eine solche Funktionen-Spezifikation anbieten, die n8n oder Flowise dann nutzen, aber das wäre ein Zusatzfeature. Für jetzt ist der direkte API-Aufruf robuster.
- **Flowise Custom Tools vs. Agents**: Flowise unterscheidet zwischen Tools (eigenständige Aktionen) und Agents (LLM-gesteuerte Aktionsauswähler) [44]. Je nachdem, wie tief man Agent-NN integrieren will, kann man es als Tool auffassen (eine Blackbox, die eine Frage in eine Antwort verwandelt) oder als Agent mit internen Tools. Letzteres auszunutzen könnte bedeuten, Agent-NN an Flowise's Tool-Schnittstelle anzubinden – z. B. könnte man Agent-NN "denken" lassen, es sei ein Tool im LangChain-Sinne. Andersherum könnte man Agent-NN erlauben, die Flowise/LangChain-Tools zu nutzen. Dies würde aber eine Custom-Integration auf Code-Ebene erfordern, in der man Agent-NN mit neuen Tool-Interfaces versieht. Aktuell scheint sinnvoller, Agent-NN als abgeschlossene Komponente einzusetzen und die Orchestrierung durch n8n/Flowise auf der Makro-Ebene zu machen (also: Flowise ruft Agent-NN, Agent-NN ruft ggf. eigene Tools).
- **Konnektoren zu Datenquellen**: Da n8n/Flowise viele Integrationen (Datenbanken, E-Mails, Cloud-Services) haben, könnte Agent-NN davon profitieren. Ein Best Practice könnte sein, Agent-NN *Werkzeuge* zu geben, die über n8n/Flowise ausgeführt werden. Theoretisch ließe sich z. B. ein Agent-NN Worker-Agent designen, der per API-Aufruf einen n8n-Workflow anstößt (etwa um in einem CRM Daten nachzuschlagen) – das geht aber über die unmittelbare Integration hinaus. Erwähnenswert ist nur: bei der Integration sollte man definieren, welche Aufgaben vom Agentensystem und welche vom Workflow-System übernommen werden. In der Regel übernimmt Agent-NN die KI-internen Entscheidungen, während n8n/Flowise den Zugang zur Außenwelt (Drittsysteme) bereitstellt.

Fazit

Die Analyse des EcoSphereNetwork/Agent-NN Repositories zeigt, dass eine **Kopplung mit n8n und Flowise** vor allem durch die wohlstrukturierte API von Agent-NN erleichtert wird. Indem man die Microservice-Komponenten von Agent-NN jeweils als **Nodes in n8n Workflows** abbildet, kann man die volle Funktionalität (vom Task-Dispatch über Agentenerstellung bis zur Vektorsuche) in automatisierte Abläufe einbinden. Gleichzeitig ermöglicht ein *zusammenfassender Agent-NN-Node* die einfache Wiederverwendung des gesamten Agentenprozesses in beliebigen Workflows, ohne dass Endnutzer die interne Logik kennen müssen. Für Flowise wiederum bietet sich die Erstellung **kundenspezifischer Komponenten** an, durch die Agent-NN als Tool oder Agent in visuellen Chat-Flows genutzt werden kann. So können Anwender über die No-Code-Oberfläche neue spezialisierte Agenten kreieren und komplexe Anfragen bearbeiten lassen, während im Hintergrund Agent-NN die intelligente Aufgabenvermittlung übernimmt.

Durch die Beachtung vorhandener *Best Practices* – etwa Nutzung vorhandener API-Standards, Trennung von KI-Logik und Orchestrierung, sowie Wiederverwendung von bewährten Mustern (Supervisor-Worker) – lässt sich die Integration effizient gestalten. Letztlich ergänzen sich Agent-NN und Tools wie n8n/Flowise: **n8n** sorgt für die Anbindung an die *Welt der Daten und Dienste*, **Flowise** für die *No-Code-Interaktion und schnelle Prototypenerstellung*, und **Agent-NN** steuert die *kognitive Intelligenz* und Lernfähigkeit bei. Eine gelungene Integration würde es ermöglichen, das Potential von Agent-NN in vielfältigen praktischen Szenarien auszuschöpfen – von Unternehmensautomatisierungen in n8n bis hin zu interaktiven LLM-Apps in Flowise.

Quellen: Die Darstellung stützt sich auf die Agent-NN Projektdokumentation [1] [2], Code-Einblicke (API-Endpunkte [25] [21]) sowie offizielle Docs von n8n [18] [19] und Flowise [39] [48], die Auskunft über Integrationsmöglichkeiten und bestehende KI-Framework-Integrationen geben.

---

[1] [4] [5] [9] [13] [14] [15] [16] [17] [33] [57] Code-Dokumentation.md
https://github.com/EcoSphereNetwork/Agent-NN/blob/b02fefbe52545c8846cd3b8c9180193c863f0b84/docs/architecture/Code-Dokumentation.md

[2] [3] [8] [10] [11] [12] system_architecture.md
https://github.com/EcoSphereNetwork/Agent-NN/blob/b02fefbe52545c8846cd3b8c9180193c863f0b84/docs/architecture/system_architecture.md

[6] Roadmap.md
https://github.com/EcoSphereNetwork/Agent-NN/blob/b02fefbe52545c8846cd3b8c9180193c863f0b84/Roadmap.md

[7] [31] [32] service.py
https://github.com/EcoSphereNetwork/Agent-NN/blob/b02fefbe52545c8846cd3b8c9180193c863f0b84/mcp/vector_store/service.py

[18] [19] [34] [35] Creating custom integration in N8N
https://blog.elest.io/creating-customer-integration-in-n8n/

[20] [21] [22] [23] [24] [25] [26] [27] [28] [29] [30] [36] [37] [55] [56] endpoints.py
https://github.com/EcoSphereNetwork/Agent-NN/blob/b02fefbe52545c8846cd3b8c9180193c863f0b84/api/endpoints.py

[38] AI Agent node documentation | n8n Docs
https://docs.n8n.io/integrations/builtin/cluster-nodes/root-nodes/n8n-nodes-langchain.agent/

[39] FlowiseAI - Stork.AI
https://www.stork.ai/ai-tools/flowiseai-92b18

[40] [41] [52] [53] [54] Building Node | FlowiseAI
https://docs.flowiseai.com/contributing/building-node

[42] Custom Tool | FlowiseAI - Flowise Docs
https://docs.flowiseai.com/integrations/langchain/tools/custom-tool

[43] [46] [47] [48] [49] [50] [51] Supervisor and Workers | FlowiseAI
https://docs.flowiseai.com/tutorials/supervisor-and-workers

[44] Agents | FlowiseAI - Flowise Docs
https://docs.flowiseai.com/integrations/langchain/agents

[45] AutoGPT | FlowiseAI
https://docs.flowiseai.com/integrations/langchain/agents/autogpt