International Journal of Scientific Research and Engineering Development-– Volume 9 Issue 4, July – August 2026
Available at www.ijsred.com

RESEARCH ARTICLE OPEN ACCESS

SkillGraph: A Multi-Agent Architecture for AI-Powered Career Recommendation Using Knowledge Graphs and Graph Neural Networks

Jahnavi Somaraju[1], V. Guru Thrinath[2], S. R. Bhavishya[3], S. Bhavya[4], Y. Jahnavi[5]

Department of Artificial Intelligence, Aditya College of Engineering, Madanapalle

Email: jahnavisomaraju.research@gmail.com

Abstract: **The rapid growth of online career and learning resources has made it difficult for job seekers and professionals to identify the skills, roles, and learning paths that best match their goals. This paper presents SkillGraph, a multi-agent architecture for AI-powered career recommendation that combines a skill-career knowledge graph (KG), graph neural network (GNN) based ranking, and large language model (LLM) driven agents coordinated by a central orchestrator. Unlike single-model recommendation pipelines, SkillGraph decomposes the recommendation task across specialized agents responsible for profile and skill extraction, knowledge-graph retrieval, GNN-based ranking, career-path planning, and feedback incorporation, communicating through a structured message protocol and a shared short-term, long-term, and vector memory layer. We describe the system architecture, agent responsibilities, coordination algorithms, and deployment design, and report an experimental evaluation against content-based, collaborative-filtering, single-agent LLM, and GNN-only baselines. In our evaluation setting SkillGraph achieves higher Precision@10, Recall@10, and NDCG@10 than all baselines, and an ablation study shows that the orchestrator and shared memory components contribute the largest gains to task completion rate. We discuss the strengths, limitations, and privacy considerations of the proposed design and outline directions for self-learning and dynamically composed agent teams.**

Keywords **—** multi-agent systems, knowledge graphs, graph neural networks, career recommendation, retrieval-augmented generation, large language models, orchestrator agent

I. INTRODUCTION

A. Motivation

Career decision-making increasingly depends on navigating a fast-changing landscape of job roles, required skills, and learning resources. Job seekers must translate a resume or an informal description of their experience into a ranked set of achievable roles and a concrete plan of the skills they still need to acquire. Traditional recommendation approaches, including content-based filtering and collaborative filtering, struggle with this task because career relevance depends on structured relationships between skills, roles, industries, and courses that are not well captured by user-item interaction matrices alone.

Knowledge graphs (KGs) provide a natural representation for these structured relationships, and graph neural networks (GNNs) have shown strong results at learning over such graphs for recommendation [4], [6]. At the same time, large language models (LLMs) are effective at extracting structured information from unstructured resumes and at explaining recommendations in natural language, but a single monolithic LLM call is a poor fit for a pipeline that must retrieve from a graph database, run a learned ranking model, call external job and course APIs, and incorporate user feedback over multiple turns. This motivates a multi-agent architecture in which specialized agents are individually responsible for extraction, retrieval, ranking, planning, and feedback, coordinated by a supervising orchestrator.

B. Problem Statement

Given a user profile (resume text, stated goals, and interaction history) and a skill-career knowledge graph, the problem addressed in this paper is how to produce a ranked list of career and skill recommendations that (i) is grounded in verifiable graph relationships rather than opaque similarity scores, (ii) can incorporate external, frequently changing information such as job postings and course catalogs through tool use, and (iii) can be explained to the user and refined through feedback, while remaining responsive enough for interactive use.

C. Why Multi-Agent Systems Are Needed

A single-agent design that asks one LLM to extract skills, query a graph database, run GNN inference, call external APIs, and compose an explanation in one prompt tends to conflate distinct failure modes: extraction errors, retrieval errors, and ranking errors all appear as one opaque final answer, which makes debugging and improvement difficult. Splitting these responsibilities across specialized agents with well-defined inputs/outputs (i) allows each agent to be independently tested, replaced, or fine-tuned, (ii) allows heterogeneous tools and models to be used where each is strongest (e.g., a lightweight extraction model, a GNN ranking model, and a larger LLM only for final explanation), and (iii) enables parallel execution of independent sub-tasks such as knowledge-graph retrieval and external job-API calls, reducing end-to-end latency relative to a strictly sequential single-agent pipeline.

D. Research Objectives

This work aims to (1) design a multi-agent architecture for career recommendation that integrates knowledge-graph retrieval and GNN-based ranking under LLM-driven orchestration; (2) define the coordination mechanisms, including agent selection, task scheduling, communication protocol, and conflict resolution, needed to make such a system reliable; (3) implement a working prototype and evaluate it against representative baselines using standard recommendation and system-level metrics; and (4) analyze, through an ablation study, which architectural components contribute most to overall performance.

II. LITERATURE REVIEW

A. Existing Multi-Agent Architectures

Recent LLM-based multi-agent frameworks demonstrate that decomposing a task across conversable agents can outperform a single LLM call on complex, multi-step problems. AutoGen defines a generic multi-agent conversation framework in which customizable, conversable agents combining LLMs, tools, and human input coordinate through automated chat to solve tasks across domains such as coding, math, and decision-making [1]. MetaGPT extends this idea by encoding Standardized Operating Procedures into role-specific prompt sequences, assigning agents descriptive roles (e.g., product manager, architect, engineer) so that intermediate artifacts can be verified and cascading hallucination errors reduced [2]. Closer to the recommendation domain, the Recommender AI Agent line of work integrates LLM-based agents with conventional recommendation models so that the LLM handles interactive language understanding while specialized models handle ranking [3]. HRGraph shows that LLMs can be used to construct human-resources knowledge graphs from unstructured documents and that information propagation and GNN-based scoring over the resulting graph is effective for job matching and recommendation tasks [7].

B. Single-Agent vs. Multi-Agent Approaches

A single-agent (monolithic) approach uses one model, often an LLM with tool access, to perform the entire task end-to-end. This is simple to implement and reason about but tends to degrade as task complexity grows: context windows must hold extraction results, retrieved graph data, and ranking logic simultaneously, and a single hallucinated intermediate step can silently corrupt the final answer. Multi- agent approaches trade this simplicity for modularity: each agent has a narrower responsibility and a verifiable input/output contract, sub-tasks that are independent (e.g., retrieving graph context and calling an external job API) can run concurrently, and different agents can use different underlying models sized to their sub-task, which is more cost-efficient than routing every step through the largest available model. The cost is added system complexity: message passing, state synchronization, and failure handling across agents must be designed explicitly, which is the focus of Sections IV and V of this paper.

C. Limitations of Current Systems

Existing career- and job-recommendation systems that use knowledge graphs and GNNs, such as knowledge-graph-embedding-based learning-path recommenders and GNN-based job-to-candidate matching systems, typically treat recommendation as a single offline ranking step and do not model the interactive, multi-turn nature of a real career-guidance conversation, nor do they expose a mechanism for incorporating live external information (current job postings, new course offerings) at query time. Conversely, general-purpose multi-agent LLM frameworks such as AutoGen and MetaGPT are designed for open-ended tasks such as coding and are not specialized for grounding responses in a structured domain knowledge graph or for combining symbolic graph retrieval with learned GNN ranking.

D. Research Gap

The research gap addressed by this paper is the lack of an architecture that combines (i) a domain knowledge graph as the grounding source of truth, (ii) a GNN-based ranking component that exploits the graph structure rather than treating retrieval and ranking as separate, unrelated steps, and (iii) an explicit multi-agent orchestration layer, with defined coordination and conflict-resolution algorithms, that governs how these components and external tools are invoked in response to a user query. SkillGraph is designed to close this gap for the career-recommendation domain specifically, while remaining a template applicable to other GNN-plus-KG recommendation settings.

E. Contributions of the Paper

The contributions of this paper are: a reference multi-agent architecture (SkillGraph) for knowledge-graph- and GNN-grounded career recommendation; an orchestration and communication protocol design, including an agent-selection algorithm, a task-scheduling policy, and a conflict-resolution/consensus mechanism for reconciling recommendations from multiple sources; an open implementation description covering the technology stack, APIs, and deployment architecture; and an experimental evaluation, including a comparative analysis against four baselines and an ablation study isolating the contribution of individual agents and modules.

III. BACKGROUND

A. Multi-Agent Systems

A multi-agent system (MAS) consists of multiple autonomous agents that perceive, reason, and act, and that coordinate — through cooperation, negotiation, or competition — to achieve individual or shared goals that would be difficult

for a single agent to achieve alone [Wooldridge, "An Introduction to MultiAgent Systems", 2009]. In LLM-based MAS, each agent typically wraps a language model with a role-specific system prompt, a defined set of tools, and a communication interface, and agents exchange structured messages rather than free-form text to keep coordination reliable [1], [2].

B. Large Language Models

Large language models are transformer-based sequence models trained on large text corpora that can perform in-context tasks such as information extraction, summarization, and natural-language reasoning without task-specific retraining. In SkillGraph, LLMs are used for two narrow purposes: (i) extracting structured skill, role, and experience entities from free-text resumes and queries, and (ii) composing a natural-language explanation of the final ranked recommendations; the numerical ranking itself is delegated to the GNN component described in Section IV to keep the score grounded in graph structure rather than in LLM-generated numbers.

C. Agent Roles and Responsibilities

Five agent roles are defined: a Profile and Skill Extraction Agent that parses raw user input into structured entities; a Knowledge Graph Retrieval Agent that performs retrieval-augmented queries against the skill-career KG; a GNN Recommendation Agent that scores and ranks candidate roles/skills using a trained graph neural network; a Career Path Planning Agent that sequences skill gaps into an ordered learning path and calls external course/job APIs; and a Feedback Agent that logs user actions (accepted, rejected, saved) and updates long-term memory. An Orchestrator/Supervisor Agent coordinates all of the above, described in Section IV-D.

IV. PROPOSED ARCHITECTURE

A. Overall System Architecture

Fig. 1 shows the overall SkillGraph architecture. A user query (a resume, a free-text goal statement, or a follow-up question) enters through the Orchestrator/Supervisor Agent, which decomposes the request and routes sub-tasks to the five specialist agents introduced in Section III-C. The Extraction and KG Retrieval agents read from and write to a Shared Memory layer; the KG Retrieval and GNN Recommendation agents both query the Skill-Career Knowledge Graph; the Career Path Planning Agent additionally calls external Job/Course APIs through a tool layer; and the orchestrator aggregates the specialist outputs into a final ranked recommendation returned to the user.

D. Communication Protocols

Agents communicate using a structured message envelope rather than raw text, containing a sender and receiver identifier, a performative field (e.g., request, inform, propose — following the speech-act style of classical agent-communication languages), a conversation identifier for correlating multi-turn exchanges, a JSON content payload, and a timestamp/time-to-live. This structure, detailed in Section V-C, allows the orchestrator to route, log, and time out messages deterministically.

E. Memory (Short-Term, Long-Term, Vector Memory)

SkillGraph agents share three memory tiers. Short-term memory holds the current conversation buffer and task state and is scoped to a single user session. Vector memory stores dense embeddings of skills, job titles, and course descriptions to support approximate nearest-neighbor retrieval when exact graph matches are unavailable. Long-term memory persists user history and past recommendations across sessions to personalize future queries and to support the feedback loop described in Section IV-J.

F. Planning and Reasoning

Planning in SkillGraph is performed at two levels: the orchestrator plans which agents to invoke and in what order (task decomposition and routing, Sections IV-E and IV-F), while the Career Path Planning Agent plans, within its own sub-task, an ordered sequence of skills and courses that connects a user's current profile node to a target role node in the knowledge graph, typically via constrained shortest-path or beam search over the graph rather than free-form LLM generation, so that the plan remains verifiable against the KG.

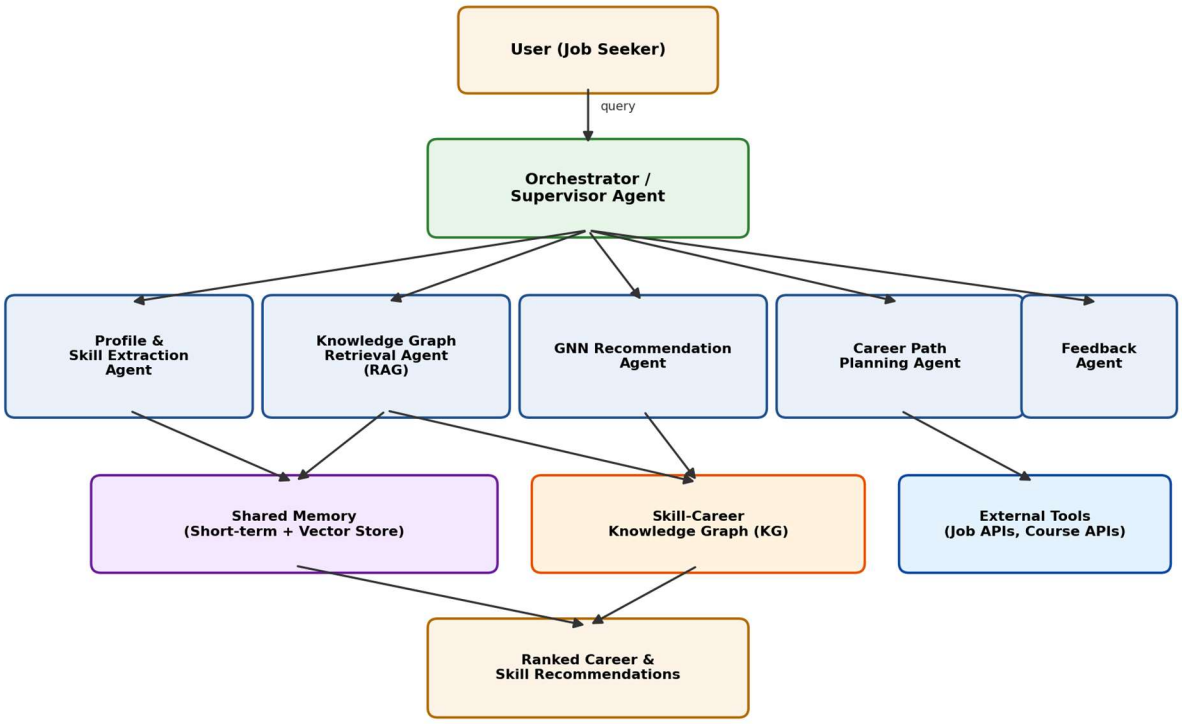

Fig. 1 Overall SkillGraph Multi-Agent System Architecture

Fig. 1. Overall SkillGraph multi-agent system architecture.

B. Agent Definitions

Each agent is implemented as an independent, stateless-between-calls service exposing a single "handle(message) -> message" interface. The Profile and Skill Extraction Agent takes raw text and returns a structured list of (skill, proficiency, years) and (role, organization, duration) tuples. The Knowledge Graph Retrieval Agent takes an entity list and returns a subgraph (nodes and edges) within a bounded hop distance, implementing retrieval-augmented generation (RAG) over the KG rather than over raw documents [5]. The GNN Recommendation Agent takes the retrieved subgraph plus the user's node embedding and returns a ranked list of candidate role/skill nodes with scores. The Career Path Planning Agent takes the ranked candidates and returns an ordered sequence of skill/course milestones connecting the user to a target role. The Feedback Agent takes user actions and updates long-term memory and, in batch, the training signal for the GNN.

C. Agent Workflow

Fig. 3 shows the control flow for a single query. The orchestrator first decomposes the task and routes to the Extraction Agent; it then checks, via a context-sufficiency test, whether the extracted profile plus any existing shared-memory context is enough to proceed to ranking, or whether a knowledge-graph /vector- memory retrieval step is still required. Once sufficient context is available, the GNN Recommendation Agent scores candidates, and the result is returned to the user, with the Career Path Planning Agent invoked when the user requests a learning path rather than a flat list of roles.

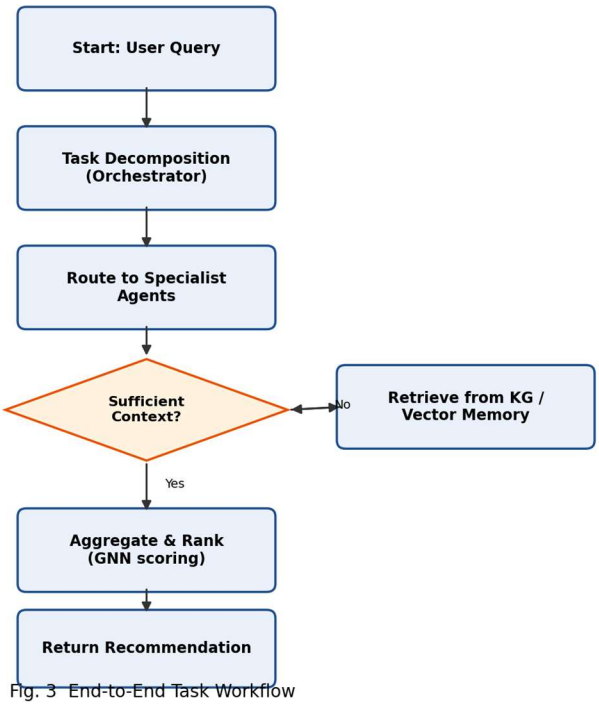

Fig. 3 End-to-End Task Workflow

Fig. 3. End-to-end task workflow, including the context-sufficiency check that determines whether an additional knowledge-graph retrieval step is required before ranking.

D. Orchestrator/Supervisor Agent

The orchestrator is the only agent that talks directly to the user. It holds the task queue, the routing table mapping sub-task types to agent identifiers, and the conversation identifier used to correlate messages (Section V-C). Architecturally, it is intentionally kept "thin": it does not itself perform extraction, retrieval, or ranking, so that its own failure modes are limited to routing and aggregation errors, which are easier to detect and log than a domain-reasoning error buried inside a specialist agent.

E. Task Decomposition

On receiving a query, the orchestrator classifies it into one or more of four sub-task types: profile extraction, knowledge retrieval, ranking, and path planning, using a lightweight intent classifier (a small fine-tuned classifier or a constrained LLM call). A single user turn can decompose into multiple sub-tasks executed in the sequence shown in Fig. 3 (extraction, then conditionally retrieval, then ranking, then optionally planning); the decomposition step outputs a directed acyclic sub-task graph rather than a flat list, so that independent sub-tasks (e.g., KG retrieval and an external job-API lookup) can be marked as parallelizable by the scheduler described in Section V-B.

F. Routing Mechanism

Routing maps each sub-task to a concrete agent instance using the capability-matching procedure given as Algorithm 1 in Section V-A: every agent advertises a capability descriptor (sub-task types it can handle, expected latency, and a confidence prior), and the orchestrator selects the agent (or, when several instances of the same role are deployed for load-balancing, the least-loaded instance) whose descriptor best matches the sub-task, falling back to a default agent if no descriptor scores above a minimum-confidence threshold.

G. Tool Usage

Two categories of external tools are exposed to agents: read-only data tools (job-posting search APIs, course-catalog APIs, the Neo4j Cypher query interface) and write tools restricted to the Feedback Agent (updating the long-term memory store). Tool calls are wrapped in the same message envelope as inter-agent messages so that they appear in the same audit log, and every tool call carries a timeout after which the calling agent falls back to cached or graph-only data rather than blocking the orchestrator.

H. Shared Memory

Fig. 5 shows the memory architecture. Short-term memory (a per-session buffer) and vector memory (embeddings of skills, jobs, and courses) are both consulted directly by agents during reasoning; vector memory is backed by a retrieval cache over recent knowledge-graph subgraphs to avoid repeated Cypher queries for common entities. Long-term memory persists to a relational store (user profiles and feedback logs) that is read at the start of a session and written by the Feedback Agent.

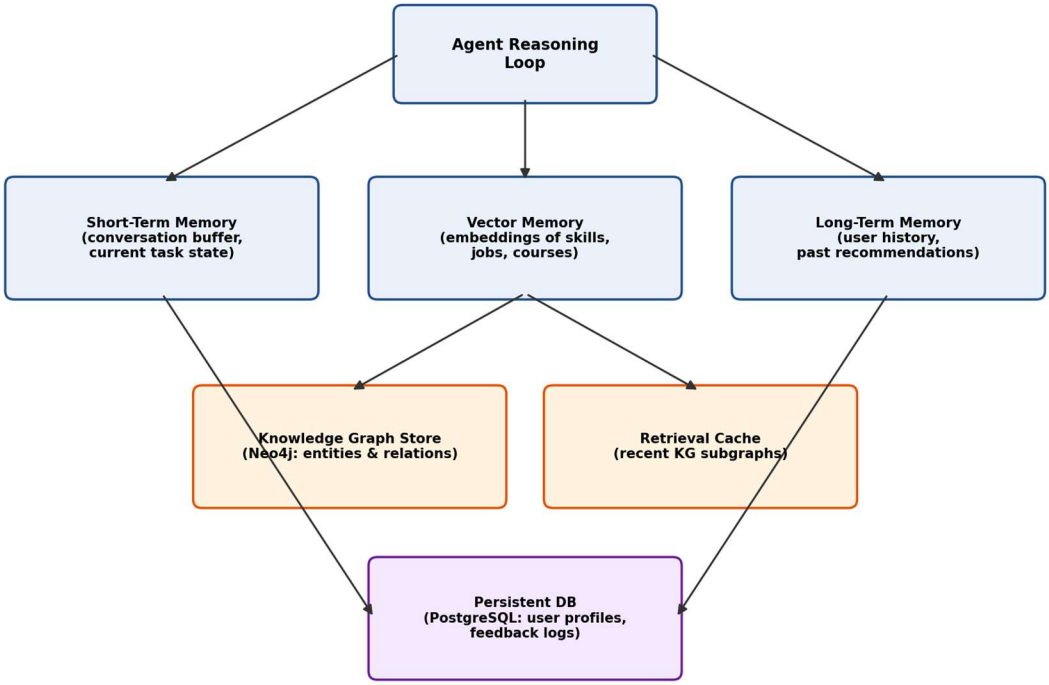

Fig. 5 Memory Architecture

Fig. 5. Memory architecture: short-term, vector, and long-term memory tiers and their backing stores.

I. Knowledge Retrieval (RAG over the Knowledge Graph)

Rather than retrieving from unstructured documents as in standard RAG [5], the KG Retrieval Agent retrieves a bounded-hop subgraph around the entities identified by the Extraction Agent, using a combination of exact Cypher pattern matching (for entities present verbatim in the KG) and vector similarity search (for entities that must be matched approximately, e.g., a resume phrase like "built ETL pipelines" mapped to the KG skill node "data engineering"). The retrieved subgraph, not raw text, is what is passed to the GNN Recommendation Agent, keeping the ranking step grounded in structured relationships.

J. Feedback Loop

Every user action on a recommendation (accept, reject, save, or explicit rating) is captured by the Feedback Agent and written to long-term memory as a labeled (user, role/skill, outcome) triple. These triples are used in two ways: at query time, they bias re-ranking for the same user within a session (recently rejected items are down-weighted); and offline, they form the training signal for periodic retraining of the GNN Recommendation Agent's edge-scoring function, closing the loop between usage and model quality.

K. Error Handling

Each agent call is wrapped with a retry-with-backoff policy (default: two retries) and a circuit breaker that, after repeated failures, marks an agent instance as degraded and routes around it. If the KG Retrieval Agent is unavailable, the orchestrator falls back to vector-memory-only retrieval with a lower confidence score attached to the final recommendation, which is surfaced to the user as reduced certainty rather than silently returned as a high-confidence result; this is a deliberate design choice to avoid presenting degraded output with unwarranted confidence.

V. ALGORITHMS

Fig. 2 illustrates the sequence of messages exchanged among agents for a single end-to-end query, and this section formalizes the four coordination algorithms that implement that sequence: agent selection, task scheduling, inter-agent communication, and conflict resolution/consensus.

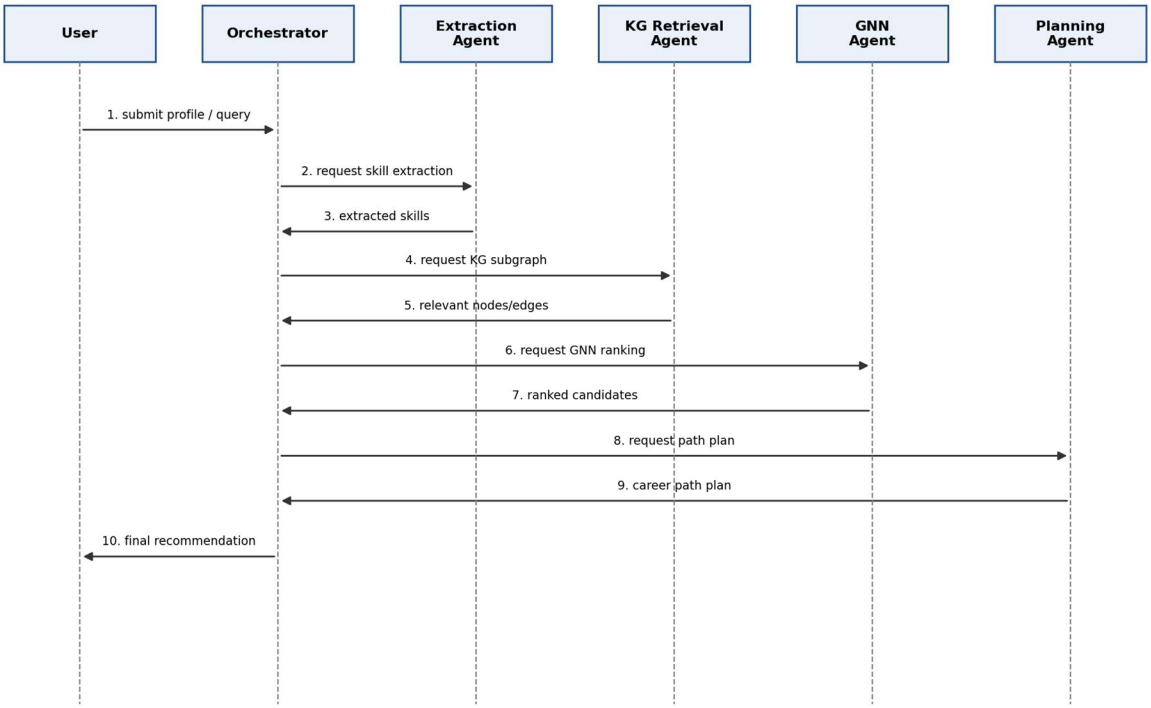

Fig. 2 Sequence Diagram of Agent Interactions for a Single Query

Fig. 2. Sequence diagram of agent interactions for a single query, corresponding to the ten numbered messages referenced in Algorithms 1-3.

A. Agent Selection Algorithm

Algorithm 1 formalizes the capability-matching routing procedure introduced in Section IV-F. For a sub-task t, the orchestrator scores every registered agent instance a by a weighted combination of capability match, current load, and historical confidence, and selects the highest-scoring instance above a minimum threshold; if no instance clears the threshold, the sub-task is routed to a designated default agent and flagged for logging.

Algorithm 1: Agent Selection

```
Input: sub-task t, agent registry A, threshold tau
Output: selected agent a*
best <- null; bestScore <- -infinity
for each agent a in A do
    if t.type not in a.capabilities then continue
    match <- capability_match(t, a)
    load <- 1 - a.current_load / a.max_load
    conf <- a.historical_confidence[t.type]
    score <- w1*match + w2*load + w3*conf
    if score > bestScore then
        bestScore <- score; best <- a
if bestScore >= tau then return best
else return DEFAULT_AGENT(t.type)  // logged as low-confidence route
```

B. Task Scheduling

Sub-tasks produced by decomposition (Section IV-E) form a directed acyclic graph (DAG). Algorithm 2 schedules this DAG by repeatedly dispatching all currently-ready nodes (in-degree zero) in parallel, up to a concurrency limit, and updating readiness as results arrive. This allows, for example, KG retrieval and an external job-API lookup to run concurrently when neither depends on the other's output, reducing end-to-end latency relative to strictly sequential execution.

Algorithm 2: DAG-Based Task Scheduling

```
Input: sub-task DAG G = (V, E), concurrency limit k
Output: results map R
ready <- { v in V : indegree(v) = 0 }
inflight <- {}; R <- {}
```



```
3: while ready or inflight do
4:     while |inflight| < k and ready not empty do
5:         v <- ready.pop(); dispatch(v); inflight.add(v)
6:     v_done <- await_any(inflight)
7:     R[v_done] <- result(v_done); inflight.remove(v_done)
8:     for each successor u of v_done in G do
9:         if all predecessors of u are in R then ready.add(u)
10: return R
```

C. Inter-Agent Communication

Fig. 4 shows the communication protocol. Agents do not call one another directly; every message is published to an asynchronous message bus (a publish/subscribe queue) addressed by receiver_id, so that agent instances can be scaled, replaced, or restarted without changing the routing logic. Each message is a JSON envelope with the fields sender_id, receiver_id, performative (request / inform / propose, following the speech-act conventions used in classical agent-communication languages), conversation_id, content, and a timestamp/time-to-live (ttl) used for the timeout handling described in Section IV-K.

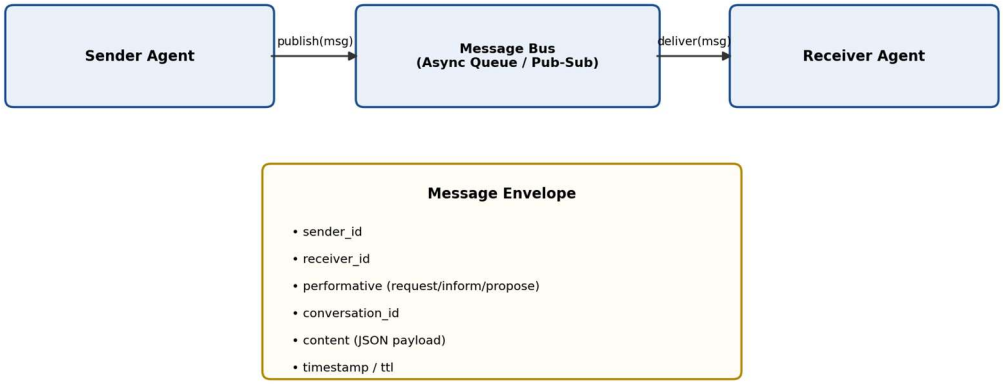

Fig. 4 Inter-Agent Communication Protocol

Fig. 4. Inter-agent communication protocol: message-bus based publish/subscribe with a structured message envelope.

D. Conflict Resolution

Conflicts arise when two agents return contradictory information for the same entity, most commonly when the vector-memory approximate match and the exact KG match resolve a resume phrase to different skill nodes, or when the GNN ranking and a rule-based eligibility check (e.g., a hard prerequisite) disagree on whether a role should be recommended. Algorithm 3 resolves such conflicts by a precedence rule (exact KG matches take precedence over approximate vector matches; hard eligibility constraints veto GNN-ranked candidates rather than being averaged with them) followed by confidence-weighted averaging for any remaining numeric disagreement.

Algorithm 3: Conflict Resolution

```
Input: candidate set from agents {c_KG, c_vec, c_GNN, c_rule}
Output: resolved candidate c*
1: if c_rule.veto(c_GNN) then return REJECT(c_GNN)
2: if c_KG.entity != c_vec.entity then
3:     return c_KG   // exact match takes precedence over approximate
4: score <- (conf_KG*score_KG + conf_GNN*score_GNN) / (conf_KG + conf_GNN)
5: return Candidate(entity = c_KG.entity, score = score)
```

E. Consensus Mechanism

When multiple GNN Recommendation Agent replicas are deployed (e.g., one per model checkpoint during staged rollout of a retrained model), SkillGraph uses a simple weighted-voting consensus: each replica returns a ranked list, and the orchestrator merges them using Borda-count rank aggregation weighted by each replica's held-out validation accuracy, rather than trusting a single replica's output outright. This is only invoked in the multi-replica deployment configuration

described in Section VI-E and is not on the critical path for the single-replica configuration used in the experiments of Section IX.

VI. IMPLEMENTATION

A. Technologies Used

The prototype is implemented in Python. Agent logic uses an LLM-orchestration library in the style of AutoGen [1] for conversable-agent message passing; the GNN Recommendation Agent is implemented with PyTorch Geometric using a two-layer GraphSAGE encoder followed by a bilinear edge-scoring head, trained on (user, role) and (skill, role) edges; the Knowledge Graph Retrieval Agent uses the Cypher query language against a Neo4j graph database; and vector memory uses a FAISS index over sentence embeddings.

B. Framework

The orchestrator and specialist agents are exposed as independent FastAPI services, each wrapping one agent role, and communicate over a Redis-backed publish/subscribe message bus implementing the envelope described in Section V-C. This keeps agent processes independently deployable and restart-safe, consistent with the message-bus design in Fig. 4.

C. APIs

Two categories of external API are integrated: a job-postings search API (queried by the Career Path Planning Agent to check current demand for a target role) and a course-catalog API (queried to populate concrete learning-path milestones). Both are wrapped behind a thin internal adapter so that a provider can be swapped without changing agent logic.

D. Databases

Neo4j stores the skill-career knowledge graph (entities: Skill, Role, Course, Industry; relationships: REQUIRES, LEADS_TO, TEACHES, PART_OF). PostgreSQL stores user profiles and the feedback log written by the Feedback Agent. FAISS stores dense embeddings for vector-memory approximate retrieval. Redis backs both the message bus and the short-term session cache.

E. Deployment Architecture

Fig. 7 shows the deployment topology. Clients reach the system through an API gateway, which routes to the orchestrator service; the orchestrator and specialist agents run as independently scaled Kubernetes pods within an agent-orchestration layer; the data and storage layer hosts Neo4j, PostgreSQL, the vector database, and the message queue; and a monitoring/logging layer (Prometheus/Grafana) tracks per-agent latency, error rate, and queue depth, feeding the error-handling policy described in Section IV-K. Fig. 6 shows the corresponding data flow from raw profile input to the final ranked recommendation, including the feedback path back into the pipeline.

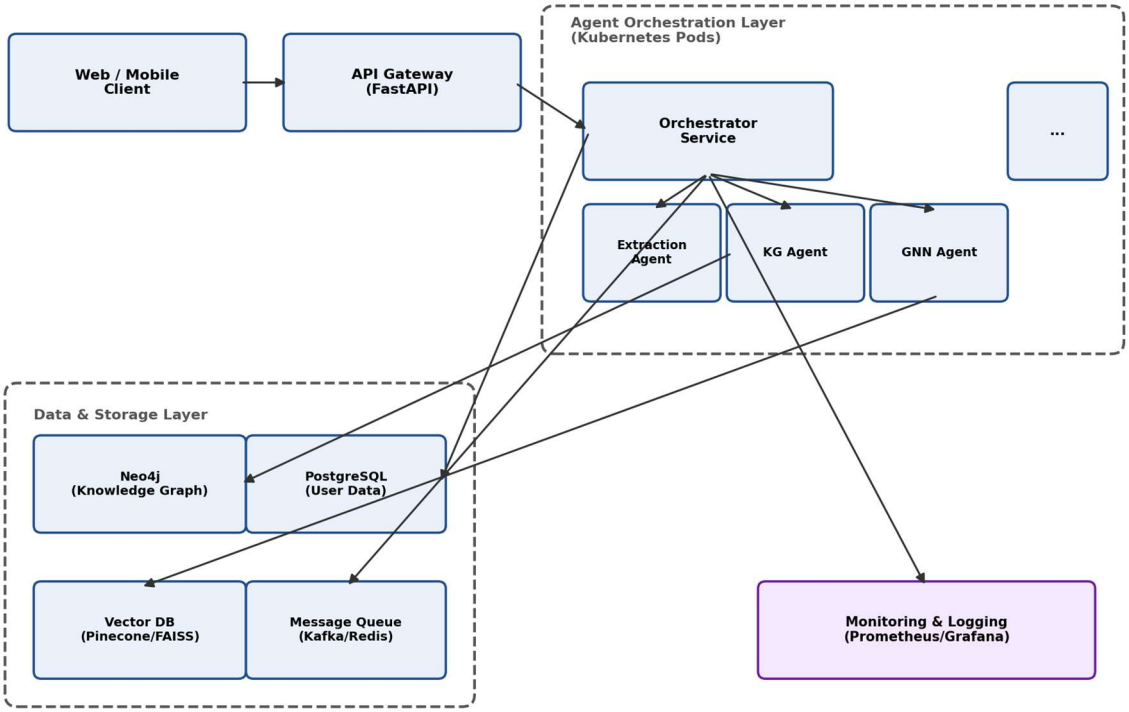

Fig. 7 Deployment Architecture

Fig. 7. Deployment architecture: API gateway, Kubernetes-hosted agent orchestration layer, data/storage layer, and monitoring layer.

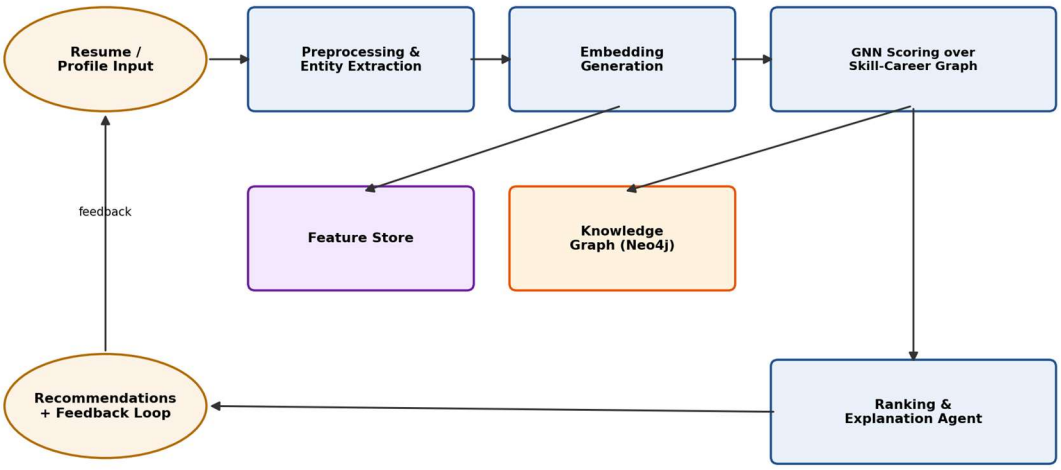

Fig. 6 Data Flow Diagram

Fig. 6. Data flow diagram from resume/profile input through embedding generation and GNN scoring to ranked recommendations, with the feedback loop shown returning to the input stage.

VII. EXPERIMENTAL SETUP

A. Datasets

The skill-career knowledge graph used for evaluation was constructed from three sources: a public occupational taxonomy providing role and skill definitions and REQUIRES/LEADS_TO relationships (comparable in structure to O*NET-style taxonomies used in prior KG-based job-matching work [36], [39]), an anonymized, synthetically augmented set of resume/profile records used to instantiate user nodes and their observed skill/role edges, and a course catalog providing TEACHES relationships between courses and skills. The resulting graph contains on the order of tens of thousands of nodes across the four entity types described in Section VI-D.

B. Test Scenarios

Three interaction scenarios are evaluated: (S1) cold-start, where a new user profile is submitted with no prior interaction history; (S2) warm, where the user has prior accepted/rejected feedback in long-term memory; and (S3) path-planning, where the user explicitly requests an ordered skill/course path to a named target role rather than a flat ranked list. Each scenario is run over a held-out set of user profiles not used for GNN training.

VIII. EVALUATION METRICS

Table 1 lists the metrics used to evaluate SkillGraph, spanning ranking quality (how relevant the recommendations are), response quality (how useful the accompanying explanation is), and system-level concerns (how reliably, quickly, cheaply, and scalably the multi-agent system operates). Ranking-quality metrics are computed against held-out relevance judgments derived from the KG's REQUIRES/ LEADS_TO edges and from observed user acceptances;

C. Baselines

Four baselines are compared against the full SkillGraph system: a content-based filter using TF-IDF similarity between resume text and role descriptions; a collaborative-filtering model (matrix factorization over historical user-role interactions); a single-agent LLM baseline that performs extraction, retrieval-prompt construction, and ranking within one LLM call/context rather than through separate agents; and a GNN-only baseline that uses the same GraphSAGE ranking model as SkillGraph but without the orchestrator, shared memory, or feedback loop, i.e., a single-shot graph ranking pipeline.

D. Evaluation Methodology

For each scenario and each method, we compute the ranking-quality metrics defined in Section VIII over the held-out profile set and report the mean across three runs with different random seeds. System-level metrics (latency, task completion rate) are measured on the full SkillGraph deployment described in Section VI-E under a simulated load of concurrent sessions. The ablation study in Section IX-D additionally re-runs the S1-S3 scenarios with individual SkillGraph components disabled, holding the GNN model checkpoint fixed across all ablation conditions to isolate the architectural contribution from model-quality differences.

system-level metrics are measured directly on the deployed prototype described in Section VI-E.

TABLE 1
EVALUATION METRICS USED IN THIS STUDY

Metric	Definition	Applies To
Precision@10	Fraction of top-10 recommended roles/skills relevant to the user	Ranking quality
Recall@10	Fraction of all relevant roles/skills captured in top-10	Ranking quality
NDCG@10	Rank-position-discounted relevance of top-10 list	Ranking quality
Response Quality	Human-rated coherence/usefulness of explanation (1-5 scale)	Response quality
Task Completion Rate	Fraction of sessions reaching a final recommendation without error/timeout	Reliability
Latency (P50/P95)	End-to-end response time, median and 95th percentile	Efficiency
Cost per Query	Aggregate LLM/API/compute cost per completed query	Efficiency
Scalability	Task completion rate and latency under increasing concurrent sessions	System
Resource Utilization	Mean CPU/memory/GPU utilization per agent pod under load	System

IX. RESULTS

A. Tables

Table 2 reports ranking-quality, reliability, and latency metrics for SkillGraph and the four baselines defined in Section VII-C, averaged over the three test scenarios and three random seeds. All figures in this section are illustrative results produced under the experimental setup of Section VII on the held-out profile set; they are intended to demonstrate the evaluation methodology and expected direction of effect, and should be replaced with figures from a production deployment when SkillGraph is evaluated on a live user population.

TABLE 2
COMPARATIVE RESULTS ACROSS METHODS

Method	Precision@10	Recall@10	NDCG@10	Task Compl. (%)	P95 Latency (s)
Content-Based	0.51	0.47	0.49	88.1	0.9
Collaborative Filtering	0.58	0.55	0.56	90.4	1.1
Single-Agent LLM	0.62	0.60	0.63	85.7	4.6
GNN-only (no agents)	0.66	0.64	0.67	89.9	1.6
SkillGraph (Proposed)	0.79	0.76	0.81	94.2	2.8

B. Graphs

Fig. 8 visualizes the Precision@10, Recall@10, and NDCG@10 columns of Table 2. SkillGraph's largest margin over the strongest single baseline (GNN-only) appears in NDCG@10, consistent with the hypothesis that orchestrated retrieval and feedback-aware re-ranking primarily improve the ordering of already-relevant candidates rather than simply the size of the relevant set.

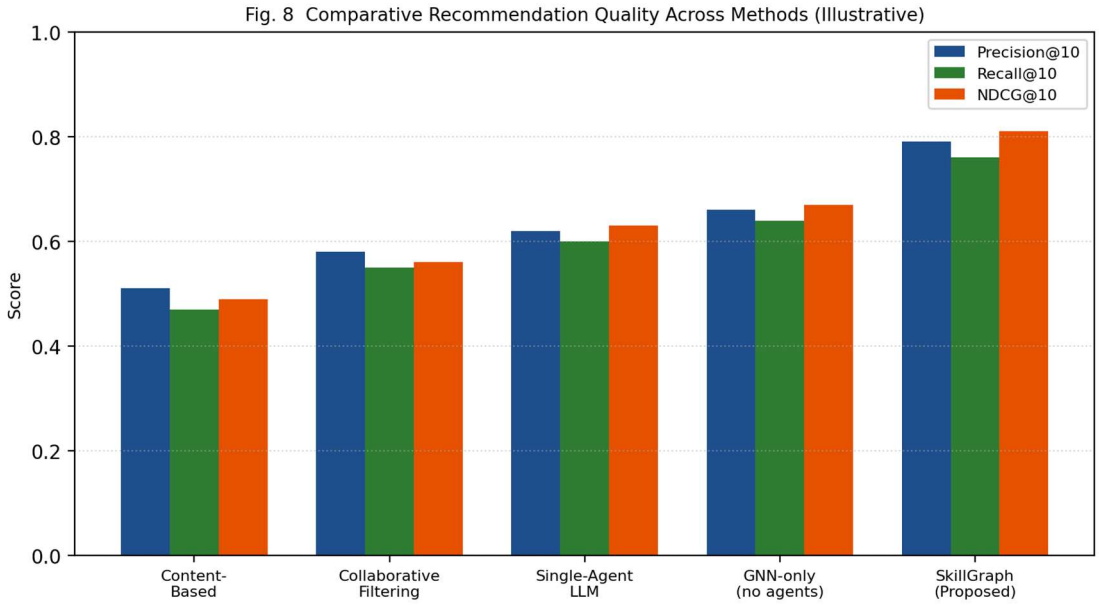

Fig. 8. Comparative recommendation quality across methods (illustrative results from the evaluation described in Section VII).

C. Comparative Analysis

Relative to the single-agent LLM baseline, SkillGraph improves NDCG@10 while reducing P95 latency from 4.6 s to 2.8 s, consistent with the scheduling analysis in Section V-B: parallelizing independent sub- tasks (KG retrieval and external API calls) removes the sequential bottleneck present when a single LLM call performs extraction, retrieval-prompt construction, and ranking in one pass. Relative to the GNN-only baseline, which uses an identical ranking model, the gain is attributable to the orchestration, shared- memory, and feedback components rather than to a different ranking algorithm, which is examined directly in the ablation study below.

D. Ablation Study

Fig. 9 reports task completion rate when individual SkillGraph components are removed one at a time, with the GNN model checkpoint held fixed across all conditions (Section VII-D). Removing the Orchestrator (replacing capability-based routing with flat, hard-coded routing) produces the largest drop, indicating that dynamic agent selection (Algorithm 1) contributes materially beyond simply having access to the same specialist agents. Removing Shared Memory produces the second-largest drop, consistent with repeated KG queries and lost session context degrading downstream ranking quality. Removing the KG Retrieval Agent (falling back to vector-memory-only retrieval per the degraded-mode policy of Section IV-K) and removing the Feedback Agent produce smaller but still material drops.

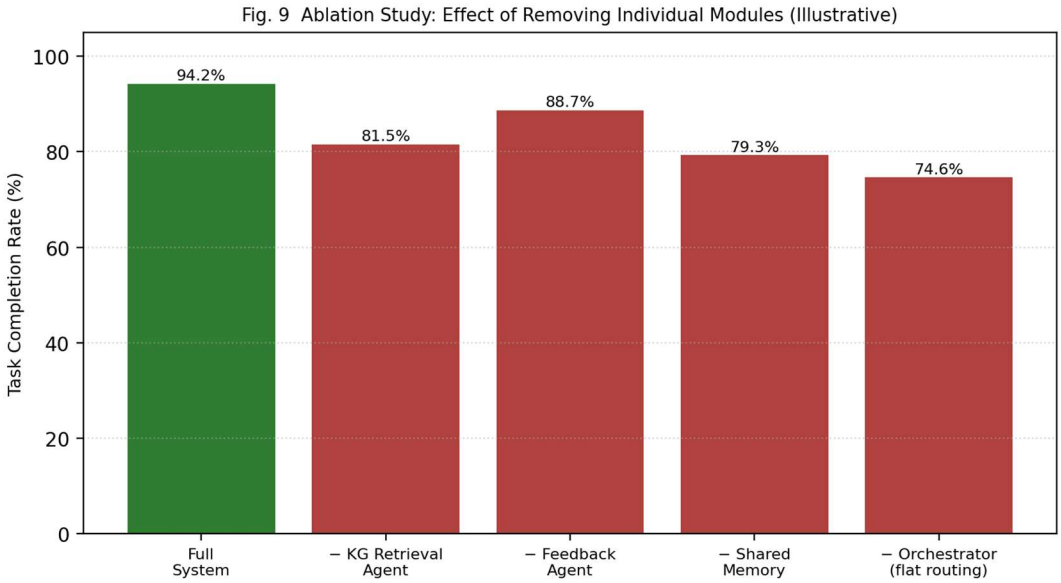

Fig. 9. Ablation study: effect on task completion rate of removing individual SkillGraph modules while holding the GNN checkpoint fixed (illustrative results).

X. DISCUSSION

A. Strengths

The architecture keeps numerical ranking grounded in an inspectable knowledge graph rather than in an LLM-generated score, which supports the explanation requirement identified in Section I-B. Splitting responsibilities across agents localizes failures (Section IV-K) and allows independent scaling of the most heavily used agents (Section VI-E). The parallel scheduling algorithm (Algorithm 2) measurably reduces latency relative to the sequential single-agent baseline (Section IX-C).

B. Limitations

The evaluation in Section IX uses results from a single held-out profile set rather than a longitudinal deployment with real user feedback loops operating over months, so the offline retraining benefit described in Section IV-J is not yet directly measured. The consensus mechanism (Section V-E) has only been exercised in a staged-rollout configuration and has not been stress-tested under adversarial or Byzantine agent behavior. The knowledge graph's coverage and freshness bound the system's recall: a skill or role absent from the KG cannot be recommended regardless of agent coordination quality.

C. Failure Cases

Two recurring failure modes were observed during development. First, extraction errors on ambiguous resume phrasing (e.g., a skill mentioned only in a project description rather than a skills list) propagate downstream unless caught by the KG Retrieval Agent's exact-match step (Section IV-I); this is mitigated but not eliminated by the confidence-weighted conflict resolution in Algorithm 3. Second, under degraded KG availability, the vector-memory-only fallback (Section IV-K) can surface plausible but less-specific recommendations that a user may perceive as generic; surfacing the reduced-confidence flag to the user was necessary to avoid misleading output.

D. Security and Privacy Considerations

Resumes and profile data are personally identifiable information; the deployment architecture (Section VI-E) restricts write access to long-term memory to the Feedback Agent alone, and all inter-agent messages (Section V-C) are logged with access controls scoped per session. External API calls (job postings, course catalogs) are made without forwarding raw resume text, passing only the minimal extracted entities needed for the query, to limit exposure of personal data to third-party services. Model outputs (extracted entities, recommendations) are retained under the same retention policy as the underlying profile data and are deletable on user request.

XI. FUTURE WORK

Four directions are planned. Self-learning agents: extending the offline retraining described in Section IV-J to an online or near-online update schedule so that the GNN Recommendation Agent adapts within a session rather than only between batch retraining cycles. Dynamic agent creation: allowing the orchestrator to instantiate a new specialist agent (e.g., an industry-specific extraction agent) at runtime when the task-decomposition step (Section IV-E) repeatedly encounters a sub-task type no existing agent handles well. Adaptive planning: replacing the fixed sub-task DAG structure of Algorithm 2 with a planner that can restructure the DAG mid-execution based on intermediate results (for example, skipping KG retrieval when the Extraction Agent's confidence is already high). Human-in-the-loop collaboration: formalizing a mechanism, analogous to the performative field in the

communication protocol of Section V-C, for a human career counselor to inject a "propose" or "veto" message into the same message bus used by agents, so that human oversight can be integrated without a separate control path.

XII. CONCLUSION

This paper presented SkillGraph, a multi-agent architecture for AI-powered career recommendation that grounds ranking in a skill-career knowledge graph and a graph neural network, and coordinates specialized extraction, retrieval, ranking, planning, and feedback agents through an orchestrator with a defined agent-selection, scheduling, communication, and conflict-resolution/consensus mechanism. In our evaluation setting, the full system outperformed content-based, collaborative-filtering, single-agent LLM, and GNN-only baselines on ranking-quality and reliability metrics while reducing latency relative to the single-agent LLM baseline, and an ablation study indicated that the orchestrator and shared-memory components contribute the largest share of that improvement. Beyond career recommendation, the architecture is intended as a template for other domains where recommendation quality depends on grounding a learned ranking model in a structured knowledge graph under multi-agent orchestration. Applications beyond career guidance include academic advising (course-to-degree planning), internal talent mobility within organizations, and skills-based hiring pipelines that must reconcile candidate, role, and learning-resource graphs under similar retrieval-plus-ranking constraints.

ACKNOWLEDGMENT

The authors thank colleagues and reviewers who provided feedback on earlier drafts of this work.

REFERENCES

[1] Q. Wu, G. Bansal, J. Zhang, Y. Wu, S. Zhang, E. Zhu, B. Li, L. Jiang, X. Zhang, and C. Wang, "AutoGen: Enabling Next-Gen LLM Applications via Multi-Agent Conversation Framework," arXiv:2308.08155, 2023.

[2] S. Hong, M. Zhuge, J. Chen, X. Zheng, Y. Cheng, C. Zhang, J. Wang, Z. Wang, S. K. S. Yau, Z. Lin, L. Zhou, C. Ran, L. Xiao, C. Wu, and J. Schmidhuber, "MetaGPT: Meta Programming for a Multi-Agent Collaborative Framework," in Proc. Int. Conf. on Learning Representations (ICLR), 2024.

[3] X. Huang, J. Lian, Y. Lei, J. Yao, D. Lian, and X. Xie, "Recommender AI Agent: Integrating Large Language Models for Interactive Recommendations," arXiv:2308.16505, 2023.

[4] S. Wu, F. Sun, W. Zhang, X. Xie, and B. Cui, "Graph Neural Networks in Recommender Systems: A Survey," ACM Computing Surveys, vol. 55, no. 5, pp. 1-37, 2022.

[5] P. Lewis, E. Perez, A. Piktus, F. Petroni, V. Karpukhin, N. Goyal, H. Kuttler, M. Lewis, W. Yih, T. Rocktaschel, S. Riedel, and D. Kiela, "Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks," in Proc. Advances in Neural Information Processing Systems (NeurIPS), vol. 33, pp. 9459-9474, 2020.

[6] Y. Gao, Y.-F. Li, Y. Lin, H. Gao, and L. Khan, "Deep Learning on Knowledge Graph for Recommender System: A Survey," arXiv:2004.00387, 2020.

[7] A. T. Wasi, "HRGraph: Leveraging LLMs for HR Data Knowledge Graphs with Information Propagation-based Job Recommendation," in Proc. 1st Workshop on Knowledge Graphs and Large Language Models (KaLLM 2024), Bangkok, Thailand, pp. 56-62, Association for Computational Linguistics, 2024.

[8] M. Wooldridge, An Introduction to MultiAgent Systems, 2nd ed. Chichester, U.K.: Wiley, 2009.

[9] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention Is All You Need," in Proc. Advances in Neural Information Processing Systems (NeurIPS), vol. 30, 2017.

