Orcheo: A Modular Full-Stack Platform for Conversational Search

Shaojie Jiang
AI Colleagues & University of Amsterdam
Amsterdam, The Netherlands
s.jiang@ai-colleagues.com

Svitlana Vakulenko
WU Vienna University of Economics and Business
Vienna, Austria
svitlana.vakulenko@wu.ac.at

Maarten de Rijke
University of Amsterdam
Amsterdam, The Netherlands
m.derijke@uva.nl

Abstract

Conversational search (CS) requires a complex software engineering pipeline that integrates query reformulation, ranking, and response generation. CS researchers currently face two barriers: the lack of a unified framework for efficiently sharing contributions with the community, and the difficulty of deploying end-to-end prototypes needed for user evaluation. We introduce Orcheo, an open-source platform designed to bridge this gap. Orcheo offers three key advantages: (i) **A modular architecture** promotes component reuse through single-file node modules, facilitating sharing and reproducibility in CS research; (ii) **Production-ready infrastructure** bridges the prototype-to-system gap via dual execution modes, secure credential management, and execution telemetry, with built-in AI coding support that lowers the learning curve; (iii) **Starter-kit assets** include 50+ off-the-shelf components for query understanding, ranking, and response generation, enabling the rapid bootstrapping of complete CS pipelines. We describe the framework architecture and validate Orcheo's utility through case studies that highlight modularity and ease of use. Orcheo is released as open source under the MIT License at https://github.com/ShaojieJiang/orcheo.

CCS Concepts

• **Information systems** → **Information retrieval**; *Retrieval models and ranking*; *Question answering*.

Keywords

Conversational search, retrieval-augmented generation, LLM agents, research infrastructure

ACM Reference Format:

Shaojie Jiang, Svitlana Vakulenko, and Maarten de Rijke. 2026. Orcheo: A Modular Full-Stack Platform for Conversational Search. In *Proceedings of Proceedings of the 49th International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR '26).* ACM, New York, NY, USA, 7 pages.

1 Introduction

Conversational search (CS) has transformed traditional information retrieval (IR) into a dynamic, multi-turn dialogue, allowing users to iteratively refine their information needs [1, 12, 26]. The promise of an end-to-end CS system brings significant architectural challenges. To function effectively, these systems must orchestrate a complex pipeline that includes query rewriting, initial passage retrieval, re-ranking, and dialogue state management [12, 27]. While research has yielded significant advances in individual components, such as dense retrieval [14], sparse representations [9], and contextual query rewriting [2, 29], the compounded impact of these innovations on the end-to-end CS pipeline remains underexplored. Consequently, it is often unclear how local optimizations translate to global improvements in user experience.

We identify two persistent challenges that hinder progress, adoption, and impact in CS research:

Challenge 1: Lack of a unified modular framework. Advancements in individual CS components are often developed within isolated, ad hoc pipelines, obscuring their true utility and complicating cross-paper comparisons. This fragmentation forces researchers to reinvent the underlying infrastructure for every new experiment, leading to inconsistent implementations and a "reproducibility gap." There is a critical need for reusable, standardized frameworks that lower the barrier to entry and provide a stable foundation for establishing reliable baselines [31].

Challenge 2: Barriers to deployment and holistic evaluation. Moving from a research prototype to a functional application is a major challenge. It requires engineering skills, such as backend orchestration and web development, that often fall outside the scope of core research. Without standardized deployment, researchers cannot easily run user-facing studies or apply robust conversational evaluation protocols [7, 23]. This lack of empirical feedback creates a bottleneck that prevents theoretical models from being systematically tested and stifles the iterative insights needed to refine future frameworks.

To address these challenges, we introduce **Orcheo**, an open-source platform designed to unify and accelerate CS research *and* development. Orcheo treats CS pipelines as first-class graph-structured workflow artifacts that can be easily shared, versioned, and collaboratively improved. We offer three main contributions:

(1) **A modular CS framework designed for reproducibility.** Orcheo is a Python platform built on LangGraph that lets researchers package contributions as plug-and-play, single-file modules. This modular architecture enables seamless integration of new models, such as query rewriters or re-rankers, without the overhead of a monolithic codebase, enabling replication and comparison. Additionally, Orcheo integrates with an open-source "Agent Skills" repository (https://agentskills.io) to support AI-assisted "vibe-coding," where coding assistants guide users through development and deployment.
(2) **Research-to-production pipeline.** Orcheo provides a full-stack environment with dual execution modes: a local CLI for

SIGIR '26, Melbourne, Australia

experimentation and a remote backend for production. To simplify engineering, it includes an encrypted Credential Vault, automated telemetry, and integrated tooling for systematic prompt engineering. This integrated stack bridges the gap between isolated prototypes and deployable applications.

(3) **An extensive CS starter kit.** We provide a curated library of 50+ off-the-shelf nodes covering the entire CS lifecycle, from query understanding to response generation, enabling rapid prototyping and benchmarking.

While optimized for CS, Orcheo's domain-agnostic, graph-based architecture generalizes to adjacent tasks (e.g., conversational recommendation) by swapping in domain-specific nodes. Orcheo's primary value lies in its broader ecosystem: by unifying algorithms with execution environments, it moves the community beyond "code sharing" to "system sharing," ensuring experiments remain executable and extensible after publication.

Section 2 reviews related work. Section 3 details Orcheo's node architecture. Section 4 describes the full-stack platform. Section 5 validates Orcheo through case studies, and Section 6 concludes.

2 Related Work

We review relevant CS research platforms and identify the gaps Orcheo addresses. Orcheo uniquely integrates modular CS pipelines, full-stack deployment, and built-in IR evaluation metrics, which are typically fragmented across existing frameworks (Table 1).

IR Toolkits for Retrieval Modeling. The IR community relies on mature retrieval toolkits such as Pyserini [19] (Python bindings for Anserini [28]) and PyTerrier [21] (a declarative framework for composing complex IR pipelines). Pyserini is built on Apache Lucene through Anserini, while PyTerrier centers on the Terrier platform with interoperability to other backends. Both are widely used for reproducible IR research, supporting traditional sparse models (e.g., BM25) and neural retrieval pipelines.

Specialized libraries have helped to advance neural IR: sentence-transformers [24] for dense embeddings, and OpenMatch-v2 [30] for training and fine-tuning neural models. Standardized evaluation and management are supported by BEIR [25] for zero-shot benchmarks and XpmIR [33] for modular neural IR experimentation.

While the established toolkits listed above excel at batch evaluation and static ranking, they are not architected for the multi-turn, interactive nature of CS. They lack built-in dialogue management for tracking context across turns and the real-time deployment infrastructure required for user-facing applications. Orcheo addresses these gaps with a unified, graph-based architecture that supports stateful CS workflows and provides extensible interfaces for integrating external retrieval and embedding components.

The effort required to integrate disparate tools creates a **standardization gap**, not just an engineering burden. When researchers use ad hoc scripts to connect components, subtle variations in context handling make it difficult to isolate the impact of specific algorithmic contributions. Orcheo bridges this gap by providing a standardized platform where modular innovations are evaluated within a consistent, reproducible system.

RAG Orchestration Frameworks. The rapid growth of retrieval-augmented generation (RAG) has introduced orchestration frameworks that link retrieval with generation. LangChain [15] is widely used for composing LLM-based pipelines, often integrating with LlamaIndex [20] for LLM-driven document indexing. Similarly, Haystack [6] offers a structured, pipeline-based approach for diverse NLP applications.

While powerful, these frameworks prioritize application development over research, creating gaps that Orcheo addresses: (i) *Ease of use*: They often require manual, code-heavy configurations rather than the AI-assisted "vibe-coding" and transparency needed for rapid experimentation. (ii) *Evaluation infrastructure*: Most orchestrators focus on execution but lack built-in, standardized evaluation routines. (iii) *CS-specific nodes*: General RAG frameworks lack native components for specialized conversational tasks like contextual query rewriting that researchers frequently reuse.

Orcheo builds on LangGraph [16], which adds stateful, graph-based orchestration to LangChain and is essential for the iterative nature of CS. For deployment, LangGraph Server [17] provides a commercial infrastructure for streaming and multi-tenancy.

Conversational AI Frameworks. Several frameworks support conversational AI and CS research. ConvLab [18] and ConvLab-2 [32] offer modular toolkits for task-oriented dialogue, but focus on slot filling rather than open-domain search. ParlAI [22] provides a unified platform for dialogue datasets and agents, though its emphasis is on language models rather than IR. More general-purpose platforms like Rasa [4] and DeepPavlov [5] provide production-ready NLU and dialogue management for building conversational assistants, yet they are designed around intent classification and skill-based architectures rather than retrieval pipelines.

These frameworks support dialogue research but lack explicit support for the full CS lifecycle, from query processing and retrieval through re-ranking and grounded response generation. Orcheo fills this void by providing a unified environment for designing modular, shareable workflows as dialogue-aware graphs.

IR Reproducibility Platforms. The IR community has standardized evaluation through platforms like TIREx [10] and its infrastructure layer, TIRA.io [11], which enable large-scale evaluation via containerized submissions. These are bolstered by TIREx Tracker [13] for automatic metadata capture and ranxhub [3] for sharing and comparing run files. These platforms excel at archiving results, but they do not address the complexities of designing and packaging reproducible CS workflows. Orcheo complements them by serving as the environment that produces the research artifacts these platforms are intended to evaluate.

3 The Orcheo Framework

This section details the architectural foundations of Orcheo. By leveraging LangGraph, Orcheo formalizes CS pipelines as stateful directed graphs in which each component is encapsulated as a discrete, interchangeable node.

3.1 Design Principles

Orcheo's framework is built around three core principles directly motivated by the challenges identified in Section 1:

(P1) *Minimal Shareable Code*: Researchers should be able to share novel algorithmic contributions as a single Python file implementing a node.

Table 1: Comparison of Orcheo with related frameworks and platforms.

Framework	Primary focus	Conversational search pipeline support	Modular sharing	Deployment tools	Evaluation support
Orcheo	Workflow orchestration	**Native**	**Node registry**	**Full-stack**	**Built-in nodes**
ConvLab-2 [32]	Task-oriented dialogue	Partial	Plugin system	Limited	Dialogue metrics
ParlAI [22]	Conversational AI	Limited	Agent zoo	Limited	Dialogue metrics
Pyserini [19]	Sparse/dense retrieval	Component only	N/A	N/A	External
LangChain [15]	LLM applications	Via composition	Hub	External	External
LangGraph Srv. [17]	Graph workflow deployment	Via composition	Via code	**Server/Cloud**	External
LlamaIndex [20]	Data + retrieval	Via composition	Hub	External	External
Haystack [6]	NLP pipelines	Via composition	Components	Cloud offering	External
TIREx [10]	IR evaluation	N/A	N/A	Evaluation only	**Comprehensive**

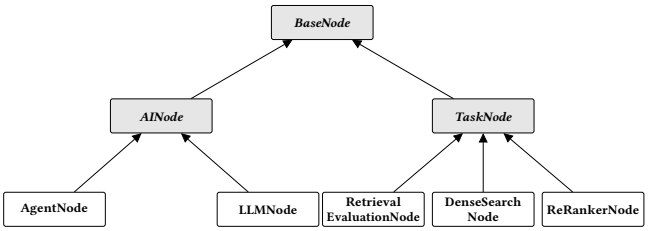

Figure 1: Orcheo's node hierarchy. BaseNode provides shared infrastructure; AINode and TaskNode specialize for LLM-based and utility operations, respectively.

(P2) *Composability with Existing Components*: To foster an ecosystem for reproducible research, new nodes should integrate seamlessly with existing upstream and downstream components.

(P3) *Type-safe Interfaces*: To move beyond "ad hoc scripts," Orcheo enforces strict schema validation. By explicitly typing node inputs and outputs, we move error detection from runtime to development time, catching integration errors early and enabling automated compatibility tests.

3.2 Node Architecture

Orcheo provides a three-tier node hierarchy designed for different use cases (Figure 1):

BaseNode. Provides core functionality shared by all nodes: (i) *Variable interpolation*: Node attributes can reference outputs from previous nodes using `{{node_name.field}}` syntax (e.g., `{{retriever.documents}}`), enabling dynamic configuration. (ii) *Credential resolution*: Secure access to API keys and secrets via `[[name]]` or `[[name#field]]` references, resolved at runtime. (iii) *Tool interface*: Nodes can be exposed as tools for agent nodes.

AINode. Extends BaseNode for nodes that involve LLM interactions (e.g., query rewriting, response generation). Outputs are wrapped in LangChain message format to ensure compatibility with conversation history. Subclasses implement an async `run()` method that returns messages.

TaskNode. Extends BaseNode for utility and integration nodes (e.g., retrieval, re-ranking, data transformation). Outputs are structured dictionaries that are merged into the workflow state. Subclasses implement an async `run()` method that returns structured data.

3.3 Built-in Conversational Search Nodes

Orcheo includes approximately 100 built-in nodes in total, including 50+ designed specifically for conversational search pipelines that cover ingestion, query understanding, retrieval, re-ranking, conversation management, generation, and evaluation. Additional nodes support data transformation, external integrations (e.g., Slack, Telegram, Discord), database operations, and workflow triggers. A comprehensive node catalog is available in the documentation at https://orcheo.readthedocs.io/en/latest/node_catalog/.

3.4 Implementing Custom Nodes

Researchers contribute new components by implementing custom nodes. Listing 1 shows a minimal example of a custom query rewriting node:

Listing 1: A custom query rewriting node.

```
from orcheo.nodes.base import AINode
from orcheo.nodes.registry import registry, NodeMetadata

@registry.register(NodeMetadata(
    name="MyQueryRewriter",
    description="Novel query rewriting method",
    category="query_processing"
))
class MyQueryRewriterNode(AINode):
    model: str = "gpt-4"
    rewrite_prompt: str = "..."

    async def run(self, state, config):
        history = state.get("messages", [])
        query = state["inputs"]["query"]
        rewritten = await self._rewrite(query, history)
        return {"rewritten_query": rewritten}
```

Key aspects of this design include: (i) *Registry-based discovery*: The `@registry.register` decorator makes the node discoverable via the CLI without modifying core code. (ii) *Pydantic configuration*: Node parameters are typed fields enabling validation and schema generation. (iii) *State access*: The `run()` method receives workflow state containing conversation history, previous outputs, and inputs. (iv) *Minimal dependencies*: The implementation depends only on Orcheo's base classes and standard LangChain types. A custom node typically requires only 20–50 lines of code.

3.5 State Management and Data Flow

Each Orcheo workflow uses a typed `State` schema that extends LangChain's `MessagesState`. The schema stores workflow inputs,

accumulated node results (keyed by node name), a structured response, and runtime configuration. The `results` dictionary accumulates outputs from `TaskNodes`, keyed by node name; downstream nodes can reference upstream outputs via variable interpolation (e.g., `{{retriever.documents}}`).

3.6 Graph Building and Execution

Workflows are defined as directed graphs with nodes and edges. Orcheo supports three edge types: (i) *Sequential edges*: Simple A → B transitions. (ii) *Conditional edges*: Routing based on state values (e.g., intent classification results). (iii) *Parallel branches*: Concurrent execution with result aggregation.

The graph builder compiles workflow definitions into LangGraph's `StateGraph`, which handles execution, checkpointing, and streaming. This architecture inherits LangGraph's benefits: async execution, automatic state persistence, and observability through LangSmith integration. Additionally, Orcheo provides native OpenTelemetry (OTel) tracing with configurable exporters, enabling integration with observability platforms such as Jaeger or Datadog. Canvas supports real-time trace streaming.

3.7 Sharing Contributions

Orcheo minimizes the code researchers need to share. At its simplest, a researcher shares a single Python file containing a custom node; adopters can install it in their registry and immediately use it alongside built-in nodes. Complete pipelines can also be shared as Python scripts that others import, inspect, and modify; credential references (e.g., `[[api_key]]`) resolve at runtime from the user's vault, so workflows can be shared without exposing tokens. The core node library (`orcheo`) can additionally be used as a standalone Python package, though the full platform is recommended for deployment and reproducibility.

3.8 Evaluation Support

Orcheo provides two complementary approaches for different evaluation scenarios. Built-in evaluation nodes can be embedded directly in workflows for online monitoring, quality gating, and offline evaluation. `ABTestingNode` routes live traffic between pipeline variants; `LLMJudgeNode` applies configurable rubrics as real-time quality gates; `RetrievalEvaluationNode` computes standard metrics (Recall@k, MRR, NDCG, MAP) compatible with TREC-style evaluation; and `AnswerQualityEvaluationNode` assesses responses using reference-based metrics and faithfulness checks. These nodes can also be composed into batch evaluation pipelines for offline benchmarking, with outputs formatted for submission to platforms like TIREx [10] or ranxhub [3].

Reproducibility support. Orcheo's evaluation infrastructure is designed with reproducibility in mind. Workflow definitions, node configurations, and prompt versions are stored as versionable artifacts alongside evaluation checkpoints, enabling researchers to precisely reconstruct experimental conditions and share complete experimental setups with minimal documentation overhead.

4 The Orcheo Platform

Beyond the core framework, Orcheo provides a full-stack platform designed to reduce the engineering burden throughout the conversational search system lifecycle. This section describes the platform components: backend services, the integrated ChatKit (https://platform.openai.com/docs/guides/chatkit) server and UI, SDK/CLI, deployment tools, tracing/observability capabilities, and agent skills for AI-assisted workflow development and deployment. While the Orcheo core node library can be used independently in custom Python projects (Section 3), the platform components described in this section provide infrastructure that makes workflows easy to deploy, share, and reproduce—capabilities that would otherwise need to be implemented and maintained by project authors.

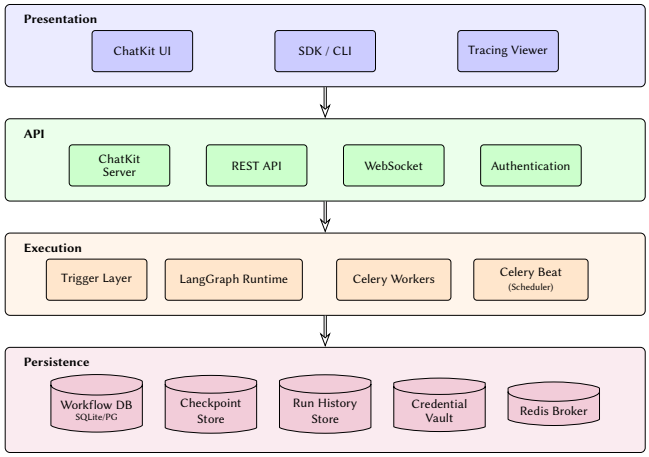

Figure 2: Orcheo platform architecture showing the interaction between the presentation layer, backend, and execution components.

4.1 Architecture Overview

Figure 2 illustrates Orcheo's platform architecture. The system comprises four main layers: (i) *Presentation layer*: ChatKit UI for embedding a full-featured chat experience (custom theming, widgets, tool invocation, file attachments, reasoning visualizations), as well as the SDK/CLI for workflow authoring and management and a tracing viewer for run inspection. (ii) *API layer*: FastAPI services exposing REST and WebSocket endpoints, the integrated ChatKit server, and authentication. (iii) *Execution layer*: Trigger layer and LangGraph runtime for orchestration, with Celery workers for background execution and Celery Beat for scheduled jobs. (iv) *Persistence layer*: Workflow DB (SQLite/PostgreSQL), checkpoint store, run history store, credential vault, and Redis broker.

4.2 Backend Services

Orcheo's FastAPI backend provides:

Workflow management. CRUD operations for workflows with versioning support. Workflows can be created via the SDK/CLI or imported from Python scripts.

Execution engine. Synchronous and asynchronous workflow execution with support for streaming responses. WebSocket connections enable real-time progress monitoring and intermediate result inspection.

ChatKit server integration. ChatKit's integration mode targets teams that need custom authentication, data residency, on-premises deployment, or bespoke orchestration. Orcheo implements the ChatKit server contract so the ChatKit frontend can connect directly to Orcheo-hosted workflows, while the server handles request processing, tool execution, streaming responses, and persistence of threads/messages/files (including uploads and action payloads).

Tracing interface. Each workflow run is captured as an execution trace, exposing node-level timings, inputs, and outputs. A tracing interface surfaces these traces for debugging, provenance inspection, and performance tuning.

Background processing. Celery workers with a Redis message broker handle long-running executions. Celery Beat supports cron-style scheduled workflows for periodic tasks (e.g., daily index updates).

Authentication and access control. Configurable authentication modes (disabled, optional, or required) with JWT support. Service tokens enable programmatic access with scoped permissions.

4.3 SDK and CLI

The Orcheo SDK (`OrcheoClient`) provides async programmatic access for workflow execution, management, and credential handling. The accompanying CLI (`orcheo`) exposes the same capabilities as shell commands for node discovery (e.g., `orcheo node list`), workflow management, and execution.

4.4 AI-Assisted Workflow Development and Deployment

Rather than requiring users to learn visual editor paradigms, Orcheo makes code-first development accessible through AI-assisted "vibe-coding." Orcheo provides an open-source *Agent Skills* repository (https://github.com/ShaojieJiang/agent-skills) that enables modern AI coding assistants (Claude Code, Cursor, OpenAI Codex CLI) to help users work with Orcheo. With the Orcheo agent skill installed, users can issue natural language requests such as "*install Orcheo*," "*start the full Orcheo stack with Docker*," or "*create a conversational search workflow with query rewriting and hybrid retrieval*," and the AI assistant generates appropriate code, commands, and configurations. This approach offers key advantages over visual workflow tools: users describe intent in plain English rather than learning tool-specific UI conventions; generated workflows are standard Python code that can be versioned and reviewed; and the same code runs locally during development and on remote backends in production without export/import steps.

4.5 Deployment Tools

Orcheo includes deployment infrastructure for moving from development to production: (i) *Docker support*: Dockerfiles and Docker Compose configurations enable single-command deployment of the full stack (backend, Redis, workers, Postgres, and Canvas). (ii) *System units*: Production deployment templates for Linux servers with proper service management and logging. (iii) *Environment configuration*: Environment variable support for database connections, authentication settings, vault configuration, and external service credentials. (iv) *Workflow publishing*: Workflows can be published with configurable access controls, enabling researchers to share interactive demos through the ChatKit frontend without building bespoke frontends for each workflow.

4.6 Credential Vault and Secret Management

Conversational search systems often require access to external services (embedding APIs, vector stores, LLM providers). Orcheo provides an AES-256 encrypted vault with runtime credential resolution via `[[credential_name]]` references, scoped access controls, and automatic OAuth token refresh. This separates credential management from workflow definitions, enabling secure sharing without exposing sensitive information.

4.7 Integration Points

Orcheo is designed to integrate with the broader conversational search research ecosystem: (i) *Vector stores*: Pluggable vector store abstraction supporting in-memory stores (development), Pinecone, and custom backends. (ii) *Embedding providers*: Registry-based embedding method resolution supporting LangChain embeddings, Pinecone inference, and custom implementations. (iii) *LLM providers*: Any LangChain-compatible LLM provider (OpenAI, Anthropic, local models via Ollama, etc.). (iv) *External toolkits*: The node architecture allows wrapping external IR toolkits to incorporate established components into Orcheo workflows, including built-in support for Pinecone text encoders (BM25, SPLADE).

5 Case Studies

We validate Orcheo's practical utility through three case studies that highlight its modularity and ease of use. The first demonstrates how built-in nodes compose into a grounded generation pipeline; the remaining two show end-to-end evaluation on established benchmarks, illustrating how Orcheo's workflow model naturally extends to evaluation tasks while substantially reducing engineering overhead. Companion workflows and documentation are available in the project repository and online documentation.

5.1 Grounded Generation Pipeline

To demonstrate how Orcheo's components work together, we present a retrieval-augmented generation pipeline (Figure 3), which is later used as the system under test in the MultiDoc2Dial evaluation (§5.3).

Pipeline composition. This pipeline chains four nodes:

(i) *QueryRewrite* rewrites the current turn's query using conversation history.
(ii) *DenseSearch* retrieves passages from a vector store.
(iii) *ContextCompressor* deduplicates context and enforces token budgets.
(iv) *GroundedGenerator* produces a cited response grounded in the retrieved context.

Modularity in practice. Each node is independently configurable: the retrieval model, vector store, and generation model are set via a JSON configuration file, with no code changes required. A researcher developing a novel query rewriting method can implement it as a custom `QueryRewrite` node, test it within this pipeline

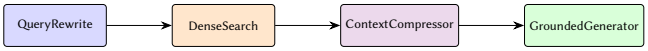

Figure 3: Grounded generation pipeline expressed as an Orcheo graph. This pipeline is used as the system under test in the MultiDoc2Dial evaluation (§5.3).

using Orcheo's built-in upstream and downstream nodes, and share only the custom node implementation (typically fewer than 200 lines). Adopters swap the node into existing workflows with a one-line configuration change, demonstrating principles P1 (minimal shareable code) and P2 (composability). Thanks to Orcheo's high-level abstractions, the entire four-stage pipeline requires only approximately 50 lines of Python code, including configuration.

5.2 QReCC Query Rewriting Evaluation

Our first evaluation targets query rewriting on the QReCC benchmark [2].

Evaluation workflow. The QReCC study is implemented as a native Orcheo workflow:

(i) `QReCCDatasetNode` loads conversations with gold rewrites.
(ii) `ConversationalBatchEvalNode` iterates through each conversation's turns, invokes a `QueryRewriteNode` subgraph, and collects predicted rewrites alongside gold labels.
(iii) `RougeMetricsNode` and `SemanticSimilarityMetricsNode` run in *parallel branches* to produce ROUGE-1 recall and embedding cosine similarity.
(iv) `AnalyticsExportNode` merges results into a unified report.

The entire pipeline executes with a single `orcheo workflow run` command; the rewriting model or embedding provider can be swapped via the JSON configuration file without code changes.

5.3 MultiDoc2Dial Grounded Generation Evaluation

Our second evaluation targets a full retrieval-generation pipeline on MultiDoc2Dial [8].

Nested workflow design. This evaluation (Figure 4) follows the same pattern but with a more complex pipeline under test: the `ConversationalBatchEvalNode` wraps the four-stage generation pipeline of Figure 3 as a composable subgraph—a workflow that evaluates another workflow. Three metric nodes—`TokenF1`, `BleuMetrics`, and `RougeMetrics` (ROUGE-L)—run in parallel downstream, and `AnalyticsExportNode` produces the final report. The same metric nodes used for QReCC (e.g., `RougeMetricsNode`) are reused here with different configurations, confirming their task-agnostic design.

5.4 Evaluation Results and Engineering Efficiency

Compared with typical manual evaluation scripts that require 500–1,000+ lines of code for data loading, history management, pipeline orchestration, metric computation, and reporting, equivalent Orcheo workflows require only 85–150 lines of code. This reduction stems from composable metric nodes with a uniform output contract, automatic conversation history threading, and configuration-driven execution that separates model choices from workflow logic.

Table 2 reports corpus-level scores for both case studies.

Table 2: Evaluation results on QReCC and MultiDoc2Dial using Orcheo workflows with GPT-4o-mini.

Benchmark	Metric	Score
QReCC	ROUGE-1 Recall	75.25
	Semantic Similarity	79.00
MultiDoc2Dial	Token F1	8.34
	SacreBLEU	13.32
	ROUGE-L	6.82

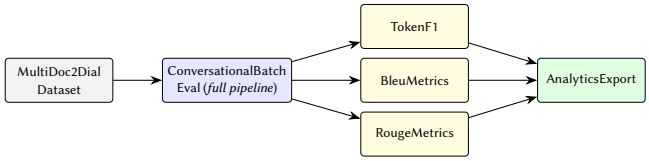

Figure 4: MultiDoc2Dial evaluation workflow. Metric nodes execute in parallel branches and results are merged by the analytics node.

6 Conclusion

We have presented Orcheo, an open-source platform designed to address two persistent challenges in conversational search research: the lack of a unified framework for sharing modular contributions, and the difficulty of deploying and sharing working systems with end users. Through its modular LangGraph-based framework with first-class AI coding support, full-stack development-to-deployment infrastructure, and a catalog of 50+ off-the-shelf CS nodes, Orcheo lowers barriers for prototyping and sharing reproducible research artifacts. We treat conversational search pipelines as first-class workflow artifacts, and seek to accelerate research progress and facilitate knowledge transfer in conversational search.

Orcheo's current design involves deliberate trade-offs worth noting. Performance on large-scale document collections (millions of passages) has not been extensively benchmarked. The modular architecture promotes flexibility, but it also means researchers must make integration decisions that monolithic systems handle implicitly. As an open-source project, Orcheo welcomes contributions to expand the ecosystem with new nodes, integrations, and best practice patterns. Finally, while Orcheo's vibe-coding support flattens the learning curve, extending and improving the framework still requires Python proficiency. We view these as necessary costs for the transparency and extensibility required by researchers.

We plan to extend Orcheo with: (i) Deeper integrations with conversational search evaluation benchmarks and shared task infrastructure, including direct TIREx submission support. (ii) A community node and workflow registry for discovering and sharing contributions. (iii) Formal node versioning with compatibility checking. (iv) Enhanced human-in-the-loop evaluation support with annotation tools and benchmark visualization in Canvas.

We invite the conversational search community to adopt, extend, and contribute to Orcheo as shared infrastructure for accelerating research progress.

Availability. Orcheo is released as open source at https://github.com/ShaojieJiang/orcheo under the MIT License. The repository includes the core framework, backend services, SDK/CLI, deployment guides, and a conversational search demo suite with runnable workflows and evaluation assets (gold queries and relevance labels).

References

[1] Avishek Anand, Lawrence Cavedon, Hideo Joho, Mark Sanderson, and Benno Stein. 2020. Conversational Search (Dagstuhl Seminar 19461). *Dagstuhl Reports* 9, 11 (2020), 34–83. doi:10.4230/DagRep.9.11.34

[2] Raviteja Anantha, Svitlana Vakulenko, Zhucheng Tu, Shayne Longpre, Stephen Pulman, and Srinivas Chappidi. 2021. Open-Domain Question Answering Goes Conversational via Question Rewriting. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. Association for Computational Linguistics, 520–534. doi:10.18653/v1/2021.naacl-main.44

[3] Elias Bassani. 2023. ranxhub: An Online Repository for Information Retrieval Runs. In *Proceedings of the 46th International ACM SIGIR Conference on Research and Development in Information Retrieval*. ACM. doi:10.1145/3539618.3591823

[4] Tom Bocklisch, Joey Faulkner, Nick Pawlowski, and Alan Nichol. 2017. Rasa: Open Source Language Understanding and Dialogue Management. *arXiv preprint arXiv:1712.05181* (2017).

[5] Mikhail Burtsev, Alexander Seliverstov, Rafael Airapetyan, Mikhail Arkhipov, Dilyara Baymurzina, Nickolay Bushkov, Olga Gureenkova, Taras Khakhulin, Yuri Kuratov, Denis Kuznetsov, Alexey Litinsky, Varvara Logacheva, Alexey Lymar, Valentin Malykh, Maxim Petrov, Vadim Polulyakh, Leonid Pugachev, Alexey Sorokin, Maria Vikhreva, and Marat Zaynutdinov. 2018. DeepPavlov: Open-Source Library for Dialogue Systems. In *Proceedings of ACL 2018, System Demonstrations*. Association for Computational Linguistics, Melbourne, Australia, 122–127. doi:10.18653/v1/P18-4021

[6] deepset. 2020. Haystack: The Open Source NLP Framework. https://haystack.deepset.ai/. Accessed: 2026-01-08.

[7] Guglielmo Faggioli, Marco Ferrante, Nicola Ferro, Raffaele Perego, and Nicola Tonellotto. 2021. Hierarchical Dependence-aware Evaluation Measures for Conversational Search. In *Proceedings of the 44th International ACM SIGIR Conference on Research and Development in Information Retrieval*. ACM, 1935–1939. doi:10.1145/3404835.3463090

[8] Song Feng, Siva Sankalp Patel, Hui Wan, and Sachindra Joshi. 2021. MultiDoc2Dial: Modeling Dialogues Grounded in Multiple Documents. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, Online and Punta Cana, Dominican Republic, 6162–6176. https://aclanthology.org/2021.emnlp-main.498/

[9] Thibault Formal, Benjamin Piwowarski, and Stéphane Clinchant. 2021. SPLADE: Sparse Lexical and Expansion Model for First Stage Ranking. In *Proceedings of the 44th International ACM SIGIR Conference on Research and Development in Information Retrieval*. ACM, 2288–2292. doi:10.1145/3404835.3463098

[10] Maik Fröbe, Jan Heinrich Reimer, Sean MacAvaney, Niklas Deckers, Simon Reich, Janek Bevendorff, Benno Stein, Matthias Hagen, and Martin Potthast. 2023. The Information Retrieval Experiment Platform. In *Proceedings of the 46th International ACM SIGIR Conference on Research and Development in Information Retrieval*. ACM. doi:10.1145/3539618.3591888

[11] Maik Fröbe, Matti Wiegmann, Nikolay Kolyada, Bastian Grahm, Theresa Elstner, Frank Loebe, Matthias Hagen, Benno Stein, and Martin Potthast. 2023. Continuous Integration for Reproducible Shared Tasks with TIRA.io. In *Advances in Information Retrieval: 45th European Conference on Information Retrieval (ECIR 2023), Part III*. Springer. doi:10.1007/978-3-031-28241-6_20

[12] Jianfeng Gao, Chenyan Xiong, Paul Bennett, and Nick Craswell. 2023. Neural Approaches to Conversational Information Retrieval. *Foundations and Trends in Information Retrieval* 17, 1 (2023), 1–163. doi:10.1561/1500000074

[13] Tim Hagen, Maik Fröbe, Jan Heinrich Reimer, Harrisen Scells, Matthias Hagen, and Martin Potthast. 2025. TIREx Tracker: The Information Retrieval Experiment Tracker. In *Proceedings of the 48th International ACM SIGIR Conference on Research and Development in Information Retrieval*. ACM. doi:10.1145/3726302.3730297

[14] Vladimir Karpukhin, Barlas Oğuz, Sewon Min, Patrick Lewis, Ledell Wu, Sergey Edunov, Danqi Chen, and Wen-tau Yih. 2020. Dense Passage Retrieval for Open-Domain Question Answering. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Association for Computational Linguistics, 6769–6781. doi:10.18653/v1/2020.emnlp-main.550

[15] LangChain. 2022. LangChain: Building Applications with LLMs through Composability. https://www.langchain.com/. Accessed: 2026-01-08.

[16] LangChain. 2024. LangGraph: Build Stateful, Multi-Actor Applications with LLMs. https://www.langchain.com/langgraph. Accessed: 2026-01-08.

[17] LangChain. 2024. LangGraph Server: Deployment Infrastructure for Stateful LLM Applications. https://langchain-ai.github.io/langgraph/. Accessed: 2026-01-23.

[18] Sungjin Lee, Qi Zhu, Ryuichi Takanobu, Xiang Li, Yaoqin Zhang, Zheng Zhang, Jinchao Li, Baolin Peng, Xiujun Li, Minlie Huang, and Jianfeng Gao. 2019. ConvLab: Multi-Domain End-to-End Dialog System Platform. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics: System Demonstrations*. Association for Computational Linguistics, 64–69. doi:10.18653/v1/P19-3011

[19] Jimmy Lin, Xueguang Ma, Sheng-Chieh Lin, Jheng-Hong Yang, Ronak Pradeep, and Rodrigo Nogueira. 2021. Pyserini: A Python Toolkit for Reproducible Information Retrieval Research with Sparse and Dense Representations. In *Proceedings of the 44th International ACM SIGIR Conference on Research and Development in Information Retrieval*. ACM, 2356–2362. doi:10.1145/3404835.3463238

[20] LlamaIndex. 2022. LlamaIndex: Data Framework for LLM Applications. https://www.llamaindex.ai/. Accessed: 2026-01-08.

[21] Craig Macdonald and Nicola Tonellotto. 2020. Declarative Experimentation in Information Retrieval using PyTerrier. In *Proceedings of the 2020 ACM SIGIR International Conference on the Theory of Information Retrieval*. ACM, 161–168. doi:10.1145/3409256.3409829

[22] Alexander Miller, Will Feng, Dhruv Batra, Antoine Bordes, Adam Fisch, Jiasen Lu, Devi Parikh, and Jason Weston. 2017. ParlAI: A Dialog Research Software Platform. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*. Association for Computational Linguistics, 79–84. doi:10.18653/v1/D17-2014

[23] Gustavo Penha and Claudia Hauff. 2020. Challenges in the Evaluation of Conversational Search Systems. In *Proceedings of the KDD 2020 Workshop on Conversational Systems (Converse)*. CEUR-WS.org. https://ceur-ws.org/Vol-2666/KDD_Converse20_paper_5.pdf

[24] Nils Reimers and Iryna Gurevych. 2019. Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*. Association for Computational Linguistics, 3982–3992. doi:10.18653/v1/D19-1410

[25] Nandan Thakur, Nils Reimers, Andreas Rücklé, Abhishek Srivastava, and Iryna Gurevych. 2021. BEIR: A Heterogeneous Benchmark for Zero-shot Evaluation of Information Retrieval Models. In *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks*.

[26] Svitlana Vakulenko, Evangelos Kanoulas, and Maarten de Rijke. 2021. A Large-Scale Analysis of Mixed Initiative in Information-Seeking Dialogues for Conversational Search. *ACM Transactions on Information Systems* 39, 4 (2021), 1–32. doi:10.1145/3466796

[27] Svitlana Vakulenko, Kate Revoredo, Claudio Di Ciccio, and Maarten de Rijke. 2019. QRFA: A Data-Driven Model of Information-Seeking Dialogues. In *Advances in Information Retrieval: 41st European Conference on IR Research (ECIR 2019)*. Springer, 541–557. doi:10.1007/978-3-030-15712-8_35

[28] Peilin Yang, Hui Fang, and Jimmy Lin. 2017. Anserini: Enabling the Use of Lucene for Information Retrieval Research. In *Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval*. ACM, 1253–1256. doi:10.1145/3077136.3080721

[29] Shi Yu, Jiahua Liu, Jingqin Yang, Chenyan Xiong, Paul Bennett, Jianfeng Gao, and Zhiyuan Liu. 2020. Few-Shot Generative Conversational Query Rewriting. In *Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval*. ACM, 1933–1936. doi:10.1145/3397271.3401323

[30] Shi Yu, Zhenghao Liu, Chenyan Xiong, et al. 2023. OpenMatch-v2: An All-in-one Multi-Modality PLM-based Information Retrieval Toolkit. In *Proceedings of the 46th International ACM SIGIR Conference on Research and Development in Information Retrieval*. ACM. doi:10.1145/3539618.3591813

[31] Edwin Zhang, Sheng-Chieh Lin, Jheng-Hong Yang, Ronak Pradeep, Rodrigo Nogueira, and Jimmy Lin. 2021. Chatty Goose: A Python Framework for Conversational Search. In *Proceedings of the 44th International ACM SIGIR Conference on Research and Development in Information Retrieval*. ACM, 2521–2525. doi:10.1145/3404835.3462782

[32] Qi Zhu, Zheng Zhang, Yan Fang, Xiang Li, Ryuichi Takanobu, Jinchao Li, Baolin Peng, Jianfeng Gao, Xiaoyan Zhu, and Minlie Huang. 2020. ConvLab-2: An Open-Source Toolkit for Building, Evaluating, and Diagnosing Dialogue Systems. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics: System Demonstrations*. Association for Computational Linguistics, 142–149. doi:10.18653/v1/2020.acl-demos.19

[33] Yuxuan Zong and Benjamin Piwowarski. 2023. XpmIR: A Modular Library for Learning to Rank and Neural IR Experiments. In *Proceedings of the 46th International ACM SIGIR Conference on Research and Development in Information Retrieval*. ACM. doi:10.1145/3539618.3591818