Master Thesis Exposé

Fachbereich Medien

Hochschule für angewandte Wissenschaft Datum: 26.08.2025

Autor:	**Frederik Brinkmann**	Supervisor:	Prof. Dr. Tillmann Schwörer
E-Mail:	frederik.brinkmann@student.fh-kiel.de	E-Mail:	Tillmann.Schwoerer@fh-kiel.de
Telefon:	01746538855	Telefon:	Telefon: +49 431 210-4541

Titel

LLM-gestützte End-to-End-Pipeline zur Überführung unstrukturierter E-Mails in API-kompatible JSON-Strukturen

Hintergrund/Motivation

Unternehmen verarbeiten täglich große Mengen geschäftsrelevanter Informationen, die primär in unstrukturierter Form vorliegen, insbesondere als E-Mails. Diese enthalten Vorgangsnummern, Ansprechpartner, Fristen oder Problemkategorien, sind jedoch für APIs, CRM- oder ERP-Systeme nicht unmittelbar verwertbar. Moderne Large Language Models (LLMs) eröffnen hier praxisnahe Optionen: Sie können unstrukturierte Texte semantisch erfassen und strukturierte Ausgaben erzeugen, die sich softwareseitig zuverlässig weiterverarbeiten lassen. Aktuelle Entwickler-Guides der Plattformanbieter zeigen, wie Function Calling und Structured Outputs die Modellantwort unmittelbar an ein JSON-Schema binden; dadurch sinkt die Fehleranfälligkeit beim Parsen und die Integration in Backend-Workflows wird deutlich robuster. Auch aus ökonomischer Perspektive ist das Thema hoch relevant: Laut McKinsey (2023) liegt das globale Wertpotenzial von Generative AI bei bis zu 4,4 Billionen USD pro Jahr – mit einem erheblichen Anteil in der Automatisierung unstrukturierter Texte wie E-Mails. Deloitte (2023) bestätigt diese Einschätzung und berichtet, dass nahezu zwei Drittel der befragten Unternehmen hier die größten Effizienzgewinne erwarten. PwC (2023) prognostiziert sogar, dass KI-Technologien insgesamt bis 2030 einen weltweiten Wirtschaftsbeitrag von 15,7 Billionen USD generieren könnten, vor allem durch die Optimierung wissensintensiver Prozesse.

Diese Arbeit untersucht die Konzeption und prototypische Umsetzung einer End-to-End-Pipeline, die unstrukturierte E-Mails automatisiert in formal validierbare, API-kompatible Datenstrukturen transformiert. Im Mittelpunkt steht dabei nicht die Gestaltung einzelner Prompts, sondern die systematische Einbettung von LLMs in eine überprüfbare und integrierbare Architektur, die wissenschaftlich fundierte Erkenntnisse liefert und zugleich praktische Relevanz besitzt.

Stand der Technik

Der Einsatz von Large Language Models (LLMs) zur strukturierten Informationsgewinnung aus Freitexten ist ein aktives Forschungsfeld. Aktuelle Surveys zeigen, dass LLMs flexibel zur Generierung strukturierter Daten eingesetzt werden können, jedoch häufig Probleme bei der Einhaltung von Schemata auftreten und Halluzinationen sowie semantische Inkonsistenzen die Verlässlichkeit einschränken (Xu et al., 2024; Zhang et al., 2025). Arbeiten wie Dagdelen et al. (2024) verdeutlichen, dass sich Strukturtreue durch grammar-constrained Decoding verbessern lässt, eine robuste Validierungsschicht bleibt jedoch unerlässlich.

Ein wesentlicher Trend ist die Einführung von Structured Outputs und Function Calling, wie sie inzwischen von Anbietern wie OpenAI und Google angeboten werden. Diese Mechanismen erlauben es, Modellantworten unmittelbar an JSON-Schemata zu binden, wodurch Parsing-Fehler reduziert und die Integration in API-Workflows erleichtert wird (OpenAI, 2024; Google Cloud, 2025). Zugleich zeigen erste Benchmarks, dass die Zuverlässigkeit zwischen Modellen stark variiert und Evaluationsmethoden wie Exact-Match, Slot-F1 und Schema-Valid-Rate etabliert bleiben, während „LLM-as-a-Judge“ aufgrund bekannter Bias-Risiken kritisch betrachtet werden muss (Gu et al., 2024).

Neben der reinen Extraktion gewinnt das Schema-Matching und Mapping an Bedeutung, da reale Zielsysteme heterogene Schnittstellen aufweisen. Erste Studien zu LLM-gestütztem Schema-Matching berichten von hoher Prompt-Sensitivität und Kontextabhängigkeit (Gupta et al., 2024), was zusätzliche Mapping- und Validierungslogik erforderlich macht. Ergänzend wird RAG als leichtgewichtiges Grounding genutzt, etwa für Enums oder Glossare, um Genauigkeit und Robustheit zu erhöhen (Huang et al., 2024).

Auch die technische Infrastruktur prägt den Stand der Technik. Frameworks wie LangChain erleichtern die Orchestrierung von LLM-Aufrufen mit Schema-Validierung, während FastAPI und Pydantic sich in der Praxis für die Umsetzung API-kompatibler Workflows etabliert haben. Für skalierbare Pipelines werden Queue-Systeme (RQ/Redis) und leichtgewichtige Speicher (SQLite) eingesetzt, oft orchestriert via Docker Compose. Diese Infrastrukturkomponenten werden nicht primär in der Forschung diskutiert, sind aber in industriellen Prototypen entscheidend, um Nachvollziehbarkeit (Logging), Resilienz und Compliance-Anforderungen – etwa im Rahmen des EU AI Acts (European Union, 2024) – sicherzustellen.

Zusammenfassend zeigt die Literatur, dass zwar zentrale Bausteine wie Structured Outputs, Validierung und RAG existieren, die Kombination in einer End-to-End-Pipeline mit robuster Infrastruktur, Schema-Mapping und regulatorischer Einbettung bislang nur unzureichend betrachtet ist. Hier setzt die vorliegende Arbeit an, indem sie eine modellagnostische Pipeline entwirft und prototypisch umsetzt.

Ziele und Zielsetzung

Ziel ist die Konzeption und Umsetzung einer LLM-gestützten End-to-End-Pipeline, die unstrukturierte E-Mails in formal validierbare, API-kompatible JSON-Strukturen überführt und exemplarisch an ein (Dummy)-Ticketsystem weiterleitet. Die Pipeline ist modellagnostisch, sodass unterschiedliche LLMs unter identischen Bedingungen verglichen werden können.

Die zentrale Forschungsfrage, die sich hieraus ableitet, lautet:

Wie lässt sich eine modellagnostische LLM-Pipeline realisieren, die unstrukturierte Texte in schema-konforme, API-taugliche JSON-Objekte transformiert und nachgelagerte Aktionen wie die Ticketanlage ausführt?

Unterfragen

1. Wie zuverlässig gelingt die strukturierte Ausgabe mit Function Calling bzw. Structured Outputs?

2. Welche Unterschiede zeigen sich zwischen verschiedenen LLMs im direkten Vergleich innerhalb der Pipeline?

3. Welche Integrationsaspekte – etwa Hosting, Auditierbarkeit und Compliance – sind für einen praxistauglichen Einsatz entscheidend?

Methodik

Zur systematischen Beantwortung der Forschungsfragen wird ein mehrstufiger methodischer Ansatz verfolgt, der die Konzeption, die prototypische Implementierung und die anschließende empirische Evaluation einer End-to-End-Pipeline umfasst. Dieser Prozess gewährleistet, dass die gewonnenen Erkenntnisse sowohl praktisch fundiert als auch wissenschaftlich nachvollziehbar sind.

Die technische Umsetzung orientiert sich am vorgestellten Architekturentwurf. Eine minimale Frontend-Applikation auf Basis von React/Next.js dient der Eingabe der E-Mails. Die Verarbeitung wird von einer Backend-API gesteuert, die mittels FastAPI/Django DRF und Pydantic implementiert wird und die eingehenden Anfragen als Jobs in eine SQLite-basierte Warteschlange einstellt. Die Kernlogik liegt in einem asynchronen Backend-Worker, der diese Jobs abarbeitet. Innerhalb des Workers erfolgt die Anbindung von zwei bis drei verschiedenen LLMs über deren dedizierte Schnittstellen für strukturierte Ausgaben wie Function Calling. Die vom LLM generierte Ausgabe wird programmatisch gegen das vordefinierte kanonische JSON-Schema validiert und anschließend über eine Mapping-Logik in das Zielschema einer Dummy-Ticket-API überführt.

Die Evaluation kombiniert qualitative Analyse und die funktionale Verifizierung der Pipeline. Anstatt aggregierter statistischer Metriken steht die Einzelfallanalyse ausgewählter, anspruchsvoller Testfälle im Vordergrund. Es wird untersucht, ob die Pipeline in der Lage ist, die semantischen Inhalte korrekt zu erfassen, sie valide zu strukturieren und fehlerfrei an das Zielsystem zu übergeben.

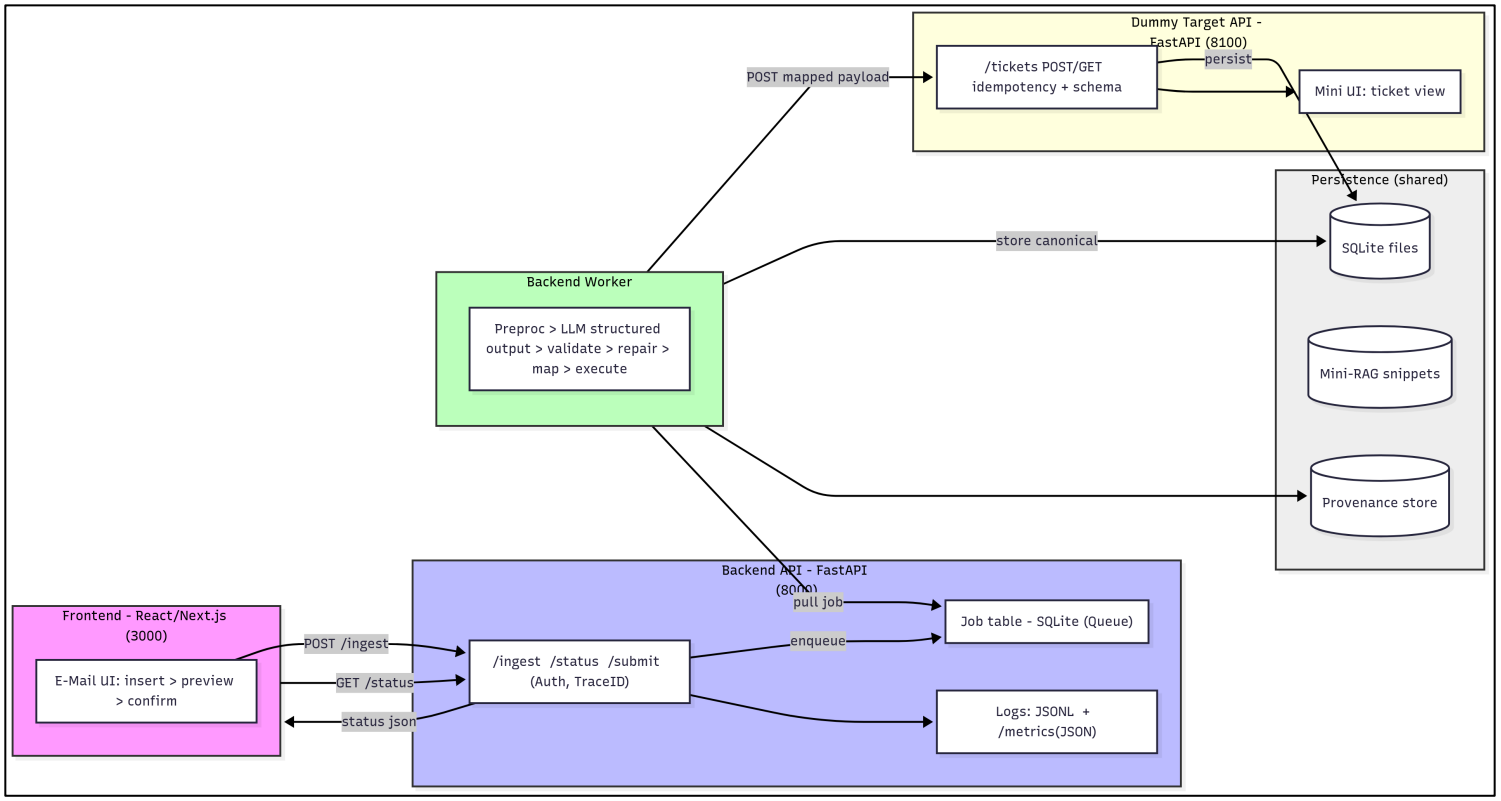
Zusätzlich werden auf Basis eines kleinen, kuratierten, synthetischen E-Mail-Sets die zentralen Leistungskennzahlen der Pipeline erhoben. Im Vordergrund stehen die Schema-Valid-Rate als Anteil strikt gültiger JSON-Ausgaben, der Record-Exact-Match für vollständige Übereinstimmungen mit der Referenz, Slot-F1-Werte auf Feldebene nach Normalisierung sowie die Latenz.

Ergänzend werden Fehlfälle analysiert, um typische Muster wie Halluzinationen, semantische Inkonsistenzen oder Formatfehler sichtbar zu machen und Stärken bzw. Schwächen der Modelle einzuordnen. Darüber hinaus wird mittels „LLM-as-Judge“ versucht erste Aussagen bzgl. der Genauigkeit und der Qualität zu treffen.

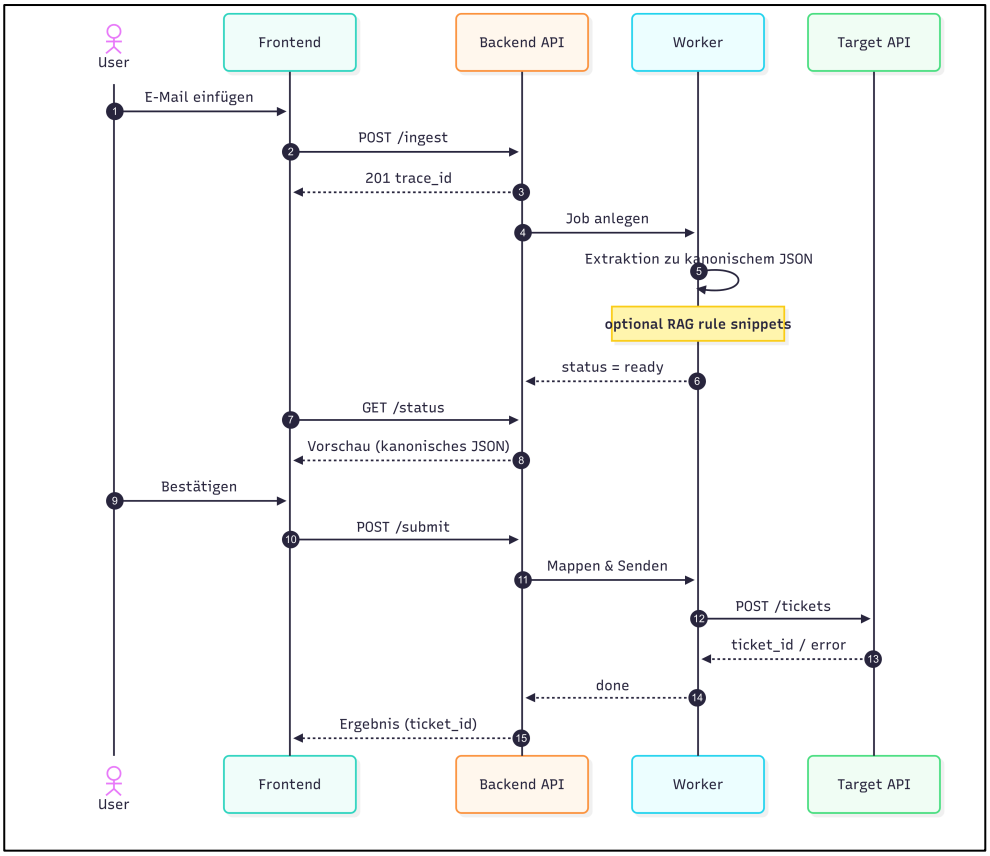
Entscheidend ist nicht, ob dieser Datensatz „besonders realistisch“ ist, sondern ob die Modelle die vorgegebenen Inhalte korrekt erkennen, sauber ins Schema übertragen und zuverlässig an die Ziel-API weitergeben. Weil alle Beispiele eindeutig festgelegt, geprüft und versioniert sind, lassen sich die Ergebnisse jederzeit nachvollziehen und wiederholen.

Architekturentwürfe

High-Level-Architekturentwurf



Entwurf Sequenzdiagramm:



Vorläufige Gliederung

1. Einleitung
1.1 Motivation
1.2 Zielsetzung
1.3 Forschungsfrage
1.4 Aufbau der Arbeit

2. Stand der Technik
2.1 LLMs mit strukturierten Ausgaben (Function Calling)
2.2 Information Extraction & RAG als Kontextgabe
2.3 Agentische Systeme
2.4 Evaluationsarten in der Literatur
2.5 Modellaufbau (Architektur)

3. Theoretischer Hintergrund
3.1 JSON-Schemata
3.2 API-Grundlagen
3.3 Mapping-Prinzip
3.4 Infrastruktur

4. Konzeption der Pipeline
4.1 Use-Case: E-Mail → Ticket
4.2 Definition der Zielstruktur
4.3 High-Level-Architekturentwurf (Frontend, Backend-API, Worker, Dummy-Zielsystem)
4.4 Anforderungen
4.5 Einsatz der Komponenten

5. Methodik
5.1 Datenbasis (synthetische E-Mails)
5.2 Modellvergleich
5.3 Evaluationsmetriken (klassisch + LLM-as-a-Judge)

6. Prototypische Implementierung
6.1 Technologie-Stack (FastAPI/DjangoDRF, Pydantic, Docker)
6.2 LLM-Anbindung mit Structured Output
6.3 Validierung & Repair
6.4 Mapping
6.5 Dummy-Ticket-API und Frontend

7. Evaluation
7.1 Ergebnisse (Tabellen/Plots)
7.2 Vergleich Modelle
7.3 Fehleranalyse (typische Fehlklassen)

8. Diskussion
8.1 Einordnung der Ergebnisse
8.2 Übertragbarkeit
8.3 Grenzen
8.4 Compliance & EU-AI-Act

9. Fazit und Ausblick
9.1 Beantwortung der Forschungsfrage
9.2 Wichtigste Learnings
9.3 Ausblick (weitere Kanäle, On-Prem-LLMs, RAG)

10. Literatur

11. Anhang

Ressourcen

- Sprache & Laufzeit
  Python für Backend/Worker; optional TypeScript/React für eine kleine UI.

- Frameworks
  Leichtgewichtiges Web-Framework für die API (FastAPI oder Django REST Framework); Validierung mit JSON Schema/Pydantic; optional React/Next.js fürs Frontend.

- LLM & Anbindung
  2–3 Modelle/Anbieter zum Vergleich (z. B. OpenAI, Anthropic, Hugging Face Inference); Nutzung von Structured Output / Function Calling; API-Keys via Umgebungsvariablen.

- APIs
  Kleine REST-API für /ingest → /status → /submit; Dummy-Ticket-API zum Anlegen/Anzeigen von Tickets (mit einfacher Idempotenz); automatische OpenAPI-Doku.

- Daten, Schemata, Mapping
  Kanonisches JSON-Schema als neutrales Ziel der Extraktion; Ziel-API-Schema für das Dummy-System; einfache Mapping-Regeln (kanonisch → Ziel); „200–500 synthetische E-Mails für Entwicklung/Evaluation“

- Speicher & Infrastruktur
  Docker-Compose für lokale Services (API, Worker, optional Frontend); SQLite für Jobs/Ergebnisse/Tickets; optional Mini-Datei-/Vector-Store für kleine RAG-Snippets (Enums/Regeln).

- Evaluation
  klassische Metriken (Schema-Valid-Rate, Slot-F1, Exact-Match, Latenz);
  LLM-as-a-Judge

- Logging & Betrieb (prototypisch)
  Leichtgewichtig: strukturierte JSON-Logs plus einfache /health und /metrics (JSON)

- Ressourcenbedarf
  API-basierter Modellzugriff (keine eigene GPU nötig); Entwicklung auf Standard-Laptop/Workstation; leichtes Cloud-Hosting für die Demo.

Zeitplan

Monat	Arbeitspaket / Meilenstein
September 2025	Projektstart, Literaturrecherche, Definition der Use Cases
Oktober 2025	Systemkonzeption & Architekturentwurf, Generierung synthetischer E-Mails
November 2025	Entwicklung LLM-Pipeline (Backend/Worker),Dummy-API aufsetzen, erste Tests
Dezember 2025	Evaluation & Analyse (Modellvergleich, Metriken), Visualisierung der Ergebnisse
Januar 2026	Schreibphase, Diskussion, Feedbackschleifen, Formatierung, Feinschliff

Literaturrecherche

Dagdelen, J., Dunn, A., Jain, A., & Persson, K. (2024). Structured information extraction from scientific text with LLMs. *Nature Communications, 15*, 1234. https://doi.org/10.1038/s41467-024-45563-x

Deloitte. (2023). *State of Generative AI in the enterprise*. Deloitte Insights. Retrieved August 31, 2025, from https://www2.deloitte.com/us/en/pages/consulting/articles/state-of-generative-ai-in-the-enterprise.html

European Union. (2024). Regulation (EU) 2024/1689 of the European Parliament and of the Council (Artificial Intelligence Act). *Official Journal of the European Union, L 2024/1689*. https://eur-lex.europa.eu/eli/reg/2024/1689/oj/eng

Es, S., James, J., Espinosa Anke, L., & Schockaert, S. (2024). RAGAs: Automated evaluation of retrieval augmented generation. In *Proceedings of the 18th EACL: System Demonstrations* (pp. 150–158). Association for Computational Linguistics. https://aclanthology.org/2024.eacl-demo.16/

Gu, J., Wang, S., Liu, Q., Zhang, Y., & Huang, M. (2024). A survey on LLM-as-a-Judge. *arXiv preprint* arXiv:2411.15594. https://arxiv.org/abs/2411.15594

Gupta, A., Zhang, H., & Chen, J. (2024). LLMs for schema matching: Opportunities and limitations. *arXiv preprint* arXiv:2407.11234. https://arxiv.org/abs/2407.11234

Huang, Y., Zhu, Q., Wang, B., & Li, S. (2024). A survey on retrieval-augmented text generation for large language models. *arXiv preprint* arXiv:2404.10981. https://arxiv.org/abs/2404.10981

Klimt, B., & Yang, Y. (2004). The Enron corpus: A new dataset for email classification research. In *Proceedings of CEAS*. https://www.ceas.cc/papers-2004/168.pdf

LangChain. (n. d.). *Structured outputs*. Retrieved August 31, 2025, from https://python.langchain.com/docs/concepts/structured_outputs/

McKinsey Global Institute. (2023, June 14). *The economic potential of generative AI: The next productivity frontier*. McKinsey & Company. Retrieved August 31, 2025, from https://www.mckinsey.com/capabilities/mckinsey-digital/our-insights/the-economic-potential-of-generative-ai-the-next-productivity-frontier

OpenAI. (2024, August 6). *Introducing structured outputs in the API*. Retrieved August 31, 2025, from https://openai.com/index/introducing-structured-outputs-in-the-api/

OpenAI. (2024). *Function calling in the API*. Retrieved August 31, 2025, from https://help.openai.com/en/articles/8555517-function-calling-in-the-openai-api

OpenAI. (n. d.). *Structured outputs guide*. Retrieved August 31, 2025, from https://platform.openai.com/docs/guides/structured-outputs

Pydantic. (n. d.). *JSON schema concepts*. Retrieved August 31, 2025, from https://docs.pydantic.dev/latest/concepts/json_schema/

PwC. (2017). *Sizing the prize: What's the real value of AI for your business and how can you capitalise?* PwC. Retrieved August 31, 2025, from https://www.pwc.com/gx/en/issues/analytics/assets/pwc-ai-analysis-sizing-the-prize-report.pdf

Redis. (n. d.). *Redis documentation*. Retrieved August 31, 2025, from https://redis.io/docs/latest/

RQ. (n. d.). *RQ: Python job queues*. Retrieved August 31, 2025, from https://python-rq.org/docs/

SQLite. (n. d.). *SQLite features*. Retrieved August 31, 2025, from https://www.sqlite.org/features.html

Docker. (n. d.). *Docker Compose documentation*. Retrieved August 31, 2025, from https://docs.docker.com/compose/

Google. (n. d.). *Structured output | Gemini API documentation*. Retrieved August 31, 2025, from https://ai.google.dev/gemini-api/docs/structured-output

Google Cloud. (2025). *Control generated output in Vertex AI*. Retrieved August 31, 2025, from https://cloud.google.com/vertex-ai/generative-ai/docs/multimodal/control-generated-output

Xu, D., Chen, W., Peng, W., Zhang, C., Xu, T., Zhao, X., & Chen, E. (2024). Large language models for generative information extraction: A survey. *Frontiers of Computer Science, 18*(6), 186357. https://doi.org/10.1007/s11704-024-40555-y

Yu, H., Zhao, R., Zhang, Y., & Wang, X. (2024). Evaluation of retrieval-augmented generation: A survey. *arXiv preprint* arXiv:2405.07437. https://arxiv.org/abs/2405.07437

Zhang, Z., Li, X., Chen, M., & Wang, H. (2025). A survey of generative information extraction. In *Proceedings of COLING 2025* (pp. 3800–3815). Association for Computational Linguistics. https://aclanthology.org/2025.coling-main.324/