Media-to-Insights: A Multi-Agent AI System for Continuous Media Monitoring, Analysis, and Reporting

Ashraf Elneima, Ozan Yilmaz, Hadi Nasrallah,
Sanjika Hewavitharana, Mohamed Al-Badrashiny, Hassan Sawaf

aiXplain, Inc., San Jose, CA, USA
{ashraf.hatim,ozan,hadi,sanijka,mohamed,hassan}@aixplain.com

Abstract

Continuous monitoring of high-volume media streams requires systems that go beyond keyword alerts to deliver structured, actionable intelligence. We present a multi-agent media monitoring system[1] that processes streaming articles through three stages: (1) a Matching Agent that uses a hybrid keyword-then-semantic matching approach, reducing agent invocations by $\sim$20%; (2) a batched multi-agent feature extraction, reducing core feature-extraction calls from 7 to 2 per article —a 71% reduction—with bounded quality tradeoffs; and (3) a Report Generation Agent that uses deterministic deduplication and density-based clustering. Four autonomous life-cycle agents manage the evolution of watchers.

1 Introduction

Organizations increasingly depend on real-time media intelligence to monitor emerging risks, policy development, and reputational signals (Leban et al., 2014; Germann et al., 2018; Amaral and Miranda, 2022). Traditional keyword-based media monitoring solutions suffer from two fundamental limitations: rigid lexical matching misses semantically relevant articles, while broad keyword configurations produce overwhelming volumes of false positives. At the other extreme, routing every article through a large language model (LLM) for semantic classification is prohibitively expensive at scale and introduces latency incompatible with real-time monitoring workflows.

Operational systems must further extract structured features efficiently, and aggregate the insights into recurring reports (Panagiotou et al., 2021; Leban et al., 2014). Furthermore, monitoring needs evolve: new topics emerge, keyword relevance shifts, and monitoring configurations must adapt over time without manual engineering effort.

We present an efficient multi-agent architecture for continuous media monitoring that balances semantic quality with cost control[2]. The system processes streaming articles through three stages: a hybrid article matching, batched multi-agent feature extraction, and report generation with deterministic deduplication and density-based clustering.

Our contributions are:

1. An efficient hybrid matching strategy combining keyword pre-filtering with agent-based semantic classification.
2. An empirical study of multi-feature batching, identifying which NLP features can be co-generated with bounded quality loss.
3. A frequency-adaptive report generation framework that is optimized for low latency and high reliability.

2 System Overview

2.1 Core Concepts

The system is organized around three primary entities: Figure 1 summarizes how these entities connect to the two-tier agent pipeline and watcher life-cycle management.

Watcher. A persistent, user-defined monitoring configuration consisting of a natural language description, topics, regions, keywords, and source filters. Each watcher defines a monitoring intent (e.g., "Track diplomatic activities in the Middle East") and continuously accumulates matched articles, extracted features, and computed embeddings, enabling longitudinal analysis and recurring reports. Previous systems provide scalable monitoring pipelines (Germann et al., 2018; Amaral and Miranda, 2022; Panagiotou et al., 2021), while our contribution is watcher-centric agent decomposition with explicit cost-quality controls.

[1]Media Monitor: https://mediamonitor.aixplain.com/

[2]Video Demo: https://youtu.be/BlUGTY3A6oI

Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 3: System Demonstrations), pages 445–452

Article. A news article ingested from one or more streaming sources. Each article can be classified under zero or more watchers (multi-label assignment). Articles that match no watcher are stored but do not undergo intensive analysis, providing cost control.

Report. A user-configured output artifact specifying a name, template, frequency (daily/weekly/monthly), recipient list, included watchers, and schedule. Reports compile analyzed articles into structured, deduplicated summaries.

2.2 Two-Tier Agent Architecture

The system employs two tiers of agents with distinct roles and capabilities:

Tier 1: Task-Specific Pipeline Agents. The three core pipeline stages (matching, analysis, report generation) are each implemented as task-specific agents optimized for throughput, consistency, and cost efficiency. Their specialization enables predictable behavior, consistent output schemas, and the batching optimizations described in Section 4.

Tier 2: Autonomous Lifecycle Agents. Four advanced agents manage the watcher lifecycle with tool access, external data retrieval, and multi-step reasoning capabilities. These agents—watcher creation, trend detection, watcher update, and watcher health— operate autonomously to assist users in configuring, maintaining, and improving watchers over time (Section 6).

2.3 Design Principles

Three principles guide the system architecture: **(a)** Cost control through staged processing: keyword pre-filtering reduces agent invocations in matching; only matched articles undergo analysis; feature batching reduces per-article feature-extraction calls substantially. **(b)** Reuse of precomputed features: features and embeddings computed during analysis are persisted and reused in report generation, avoiding redundant computation and ensuring cross-report consistency. **(c)** Right tool for the task: embedding-based methods handle structured operations (deduplication, similarity, clustering) where agents are unreliable, while agents handle semantic tasks requiring reasoning. These cost reductions translate directly to latency improvements: per-article processing time is dominated by LLM inference, so reducing agent calls from 7 to 2 for feature extraction proportionally lowers end-to-end latency; matching and report generation each add one to two calls per article and per report, respectively.

3 Matching: Article-to-Watcher Classification

3.1 Problem Definition

Given an article text a and a watcher configuration $w = (\text{description}, \text{topics}, \text{keywords}, \ldots)$, the matching agent outputs a binary decision: whether article a is relevant to watcher w. Since each article is evaluated against all N active watchers, the task is a multi-label classification problem where each article receives zero or more watcher labels.

3.2 Approaches

We evaluated three matching strategies:

Keyword Matching (Exact Match). Checks whether the article text contains any of the watcher's keywords. Fast and zero agent cost, but operates at the lexical level, producing high recall at the expense of very low precision.

Agent-Based Semantic Matching. The matching agent reasons over article content against the watcher's description, topics, and keywords, capturing nuanced semantic relationships beyond keyword overlap. This achieves high precision and recall but requires one agent invocation per article-watcher pair.

Hybrid Matching (Keyword Filter + Agent). Keyword filtering serves as an initial coarse filter, allowing only articles that contain at least one keyword to advance to agent-based semantic classification, thereby reducing agent invocations without compromising semantic matching quality.

4 Analysis: Multi-Agent Feature Extraction

4.1 Analysis Architecture

The architecture employs a team of specialized sub-agents coordinated by an output aggregation agent:

- **Main Analysis Agent:** Extracts seven core features from the article text: language, topics, summary, title, keywords, entities, and entity-level sentiment (sentiment_by_entity). A key design question—addressed in our evaluation (Section 7.2)— is how many of these features can be

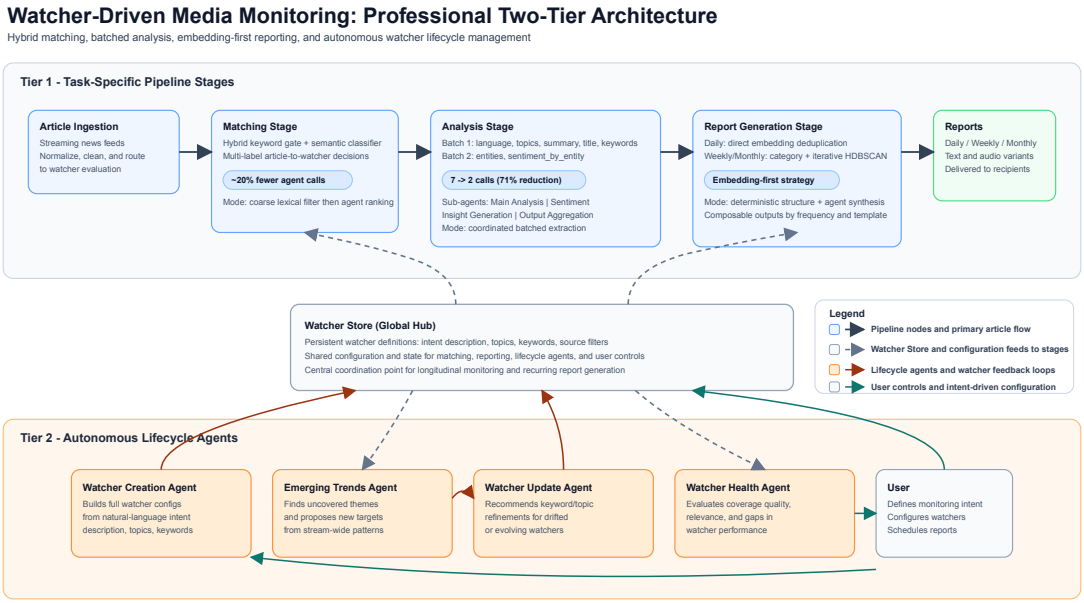

Figure 1: System architecture with two tiers: task-specific pipeline stages (Ingestion → Matching Stage → Analysis Stage → Report Generation Stage), a global Watcher Store as the central configuration hub, autonomous watcher lifecycle agents (creation, trends, update, health), and report outputs (daily/weekly/monthly).

generated in a single agent call without quality degradation.

- **Sentiment Agent:** Computes a global article-level sentiment distribution as percentages across three classes: positive, negative, and neutral. This is separated from the main analysis agent because it requires a different analytical framing (document-level proportion estimation vs. extraction).
- **Insight Agent:** Generates high-level qualitative insights per article—interpretive statements about significance, implications, or trends. These insights serve as building blocks for report generation.
- **Output Aggregation Agent:** Collects outputs from all sub-agents and merges them into a single structured dictionary per article, ensuring schema consistency for downstream consumption.

Additionally, an embedding model computes article-level and insight-level embeddings, which are persisted alongside the extracted features for use in deduplication and clustering during report generation.

4.2 Feature Batching: From 7 Agent Calls to 2

For the main analysis agent's seven features, a naive approach would dedicate one call per feature, requiring 7 calls per article (the sentiment and insight agents each contribute a fixed additional call). We investigate whether multiple features can be co-generated in a single call without significant quality loss; our evaluation (Section 7.2) shows the tradeoff is acceptable for the majority of features. Based on these findings, we adopt the following configuration:

- **Batch 1** (single agent call): language, topics, summary, title, keywords. These five features are semantically related and can be co-generated with acceptable quality (Tie+Better $\geq$ 76% relative to single-feature calls).
- **Batch 2** (separate agent call): entities and sentiment_by_entity. These features benefit from being co-generated together (entities reaching 71% Tie+Better when paired with sentiment_by_entity) but degrade significantly when included in the larger batch.
- **Optional Batch 3**: sentiment_by_entity alone, for quality-critical deployments where entity-sentiment accuracy is paramount.

This reduces per-article feature-extraction calls from 7 to 2 (71% reduction). The system also supports requesting a feature subset for further cost optimization when only specific features are needed.

5 Report Generation

5.1 Report Configuration

Users configure reports by specifying: report name, template, frequency (daily, weekly, or monthly), recipient list, one or more watchers to include, and a delivery schedule.

5.2 Deduplication and Aggregation Strategies

The choice of strategy is determined by what structure is available as input and what granularity of output is required.

Direct Embedding-Based Deduplication: When the report presents individual article-level features (e.g., a digest of summaries), the only requirement is removing redundant coverage. We apply cosine similarity on precomputed article or insight embeddings to identify near-duplicates, retaining the most recent article from the most credible source. This approach is fast, deterministic, and leverages embeddings already persisted during analysis. An early agent-based deduplication approach failed consistently, motivating the switch to embeddings. Two daily report variants are supported: text reports and audio reports (with an additional text-to-speech pipeline).

Category-Specific Deduplication: When the report template specifies thematic sections (e.g., "Economy," "Diplomacy"), each article is assigned to its matching category and deduplicated within that category independently. Generate insights per category.

Iterative Clustering and Grouping: When topic structure must emerge from the data, we employ a density-based clustering pipeline: (i) cluster articles using HDBSCAN (McInnes et al., 2017) on article embeddings; (ii) iteratively relax clustering parameters (`min_cluster_size`, `cluster_selection_method`) and recluster standalone articles until all are assigned or the iteration limit is reached; (iii) merge smaller clusters into larger groups when possible (max three subgroups per merge); (iv) generate summaries and headlines for final clusters.

These strategies are composable—a single report can employ different strategies in different sections.

6 Watcher Lifecycle Agents

Four autonomous lifecycle agents manage watcher configuration and evolution, each with tool access and multi-step reasoning:

Watcher Creation Agent: Given a short natural-language description of a monitoring intent (e.g., "I want to track renewable energy policy in Southeast Asia"), this agent produces a fully specified watcher configuration including a detailed description, curated keywords, relevant topics, and suggested source filters.

Emerging Trends Detection Agent: Monitors the daily article stream over a configurable period to identify emerging topics and keywords not currently captured by any active watcher, surfacing suggestions for new monitoring targets.

Watcher Update Agent: Analyzes existing watcher configurations against the evolving media landscape and recommends specific updates - adding new keywords, expanding topic lists, or refining descriptions - to keep watchers current.

Watcher Health Agent: Evaluates the operational health of each watcher by assessing article volume adequacy, relevance of matched articles to the watcher's intent, and coverage gaps suggesting misconfiguration.

7 Evaluation

7.1 Matching Evaluation

We evaluate matching on an internally annotated dataset spanning **nine** diverse watchers and **1,634** article–watcher instances in total. The evaluation is organized in two tiers: (i) an initial pilot of four watchers (96 articles each), with positive-class prevalence from 4.2% (4/96) to 25.0% (24/96); and (ii) an expanded follow-up of five additional watchers (250 articles each), with prevalence from 3.6% (9/250) to 32.4% (81/250). The second tier increases sample size and stresses the system under a wider range of base rates and topical profiles. Labels are binary relevance judgments from domain experts. We compare three matching strategies: keyword, agent-only, and hybrid.

Table 1 reports precision, recall, and F1 for each method and watcher.

	Keyword			Agent-Only			Hybrid			n
	P	R	F1	P	R	F1	P	R	F1	n
W1	.05	1.00	.10	1.00	.75	.86	1.00	.75	.86	96
W2	.10	.67	.18	1.00	.92	.96	1.00	.58	.74	96
W3	.31	1.00	.48	1.00	1.00	1.00	1.00	1.00	1.00	96
W4	.19	1.00	.33	1.00	.93	.97	1.00	.93	.97	96
W5	.07	1.00	.12	1.00	1.00	1.00	1.00	1.00	1.00	250
W6	.62	.71	.67	1.00	.57	.73	1.00	.57	.73	250
W7	.67	.75	.71	1.00	.75	.86	1.00	.50	.67	250
W8	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	250
W9	.55	1.00	.71	1.00	.83	.91	1.00	.83	.91	250
Avg	.39	.90	.48	1.00	.86	.92	1.00	.80	.87	1,634

Table 1: Matching evaluation across nine watchers (1,634 article–watcher instances). n is the number of labeled articles per watcher. P = Precision, R = Recall. Rows W1–W4 and W5–W9 correspond to the pilot (96 per watcher) and expanded (250 per watcher) tiers.

Keyword matching achieves high recall (macro avg. 0.90) but low precision (0.39, F1 0.48), making it impractical as a standalone filter. Agent-based semantic matching retains perfect precision (1.00) across all nine watchers with strong recall (avg. 0.86, F1 0.92), nearly doubling keyword F1.

Hybrid matching preserves perfect precision and matches agent-only recall on seven of nine watchers while reducing matching agent invocations by $\sim$20%. Recall drops relative to agent-only only for Watchers 2 and 7: in both cases the topic is semantically broad relative to a lexically narrow keyword list, so the keyword gate removes relevant articles that use different terminology. For well-covered keyword sets, hybrid tracks agent-only; where maximum recall is critical, users can fall back to agent-only matching.

7.2 Analysis Feature Batching Evaluation

We evaluate whether multiple features can be extracted in a single agent call without significant quality loss, compared to the baseline of one dedicated agent call per feature (7 calls total for 7 features). We use an LLM-as-judge protocol with **GPT-4o** as the judge model (API access, temperature 0): for each of 100 randomly sampled articles, the judge compares the output of the batched configuration against the single-feature baseline for each of the 7 features (language, topics, summary, title, keywords, entities, sentiment_by_entity). The judge assigns one of three labels: A = single-feature is better, B = batched is better, Tie = equivalent quality. We report the Tie+B percentage as the primary metric, the fraction of cases where the batched output is at least as good as the single-feature baseline.

We evaluate three batching configurations:

- **Config 1** (All-in-one): All 7 features in a single agent call ($7 \rightarrow 1$, 86% call reduction).
- **Config 2** (Two-call split): {language, topics, summary, title, keywords} in one call + {entities, sentiment_by_entity} in a second call ($7 \rightarrow 2$, 71% call reduction).
- **Config 3** (All except sentiment): {language, topics, summary, title, keywords, entities} in one call, sentiment_by_entity excluded ($7 \rightarrow 1{+}1$, 71% call reduction, but requires a separate sentiment_by_entity call if needed).

Table 2 reports the LLM-as-judge results for each configuration.

To validate the judge in this setting, three domain experts independently labeled the same batched-vs.-single-feature comparisons using the A/B/Tie scheme; expert disagreements were resolved by **majority vote** to obtain a human reference label for each instance. Pooling all comparisons across Configs 1–3 yields $N{=}2000$ side-by-side judgments. Table 3 reports the confusion matrix (rows = LLM judge, columns = human majority). **Exact agreement** is 77.5% (1550/2000). Most probability mass lies on the diagonal, especially Tie–Tie; the largest off-diagonal slice is LLM = *A* with human = *Tie* (120 cases), suggesting the judge is sometimes stricter toward the batched output than the expert consensus rather than erratic.

	Human (majority)		
LLM judge	A	B	Tie
A	360	120	30
B	10	160	0
Tie	110	180	1030

Table 3: LLM judge vs. human majority vote for the batching comparison task, pooled over Configs 1–3 ($N{=}2000$). Diagonal counts sum to 1550 (77.5% exact agreement).

The results reveal clear groupings in feature batchability:

Safely batchable (language, topics, summary, title): These generative, summarization-oriented features consistently achieve Tie+B $\geq$ 76% across configurations, because they draw on the same holistic comprehension of the article. Summary is particularly robust (91–97% Tie+B). Keywords sit at the boundary (61–76%), performing best in the two-call split (Config 2).

	Config 1 (All-in-one)				Config 2 (Two-call)				Config 3 (All exc. sent.)			
Feature	A	B	Tie	T+B%	A	B	Tie	T+B%	A	B	Tie	T+B%
language	*(100% match—skipped in all configurations)*											
topics	13	27	60	87	15	22	62	84	17	24	59	83
summary	9	0	91	91	3	0	97	97	5	0	95	95
title	22	37	41	78	17	39	44	83	23	35	42	77
keywords	39	39	22	61	24	58	18	76	37	43	20	63
entities	68	23	9	32	29	35	36	71	62	30	8	38
sent._by_ent.	65	28	7	35	57	35	8	43	*(excluded)*			

Table 2: LLM-as-Judge results for feature batching (n=100 per feature). A = single-feature better, B = batched better, Tie = equivalent. T+B% is the percentage of cases where batching is at least as good as the single-feature baseline.

Requires separation (entities, sentiment_by_entity): Entity extraction degrades dramatically when batched with all features (32% in Config 1) but improves to 71% when co-generated only with sentiment_by_entity (Config 2). Entity-level sentiment shows persistent degradation (max 43%), as the compound task of entity identification plus per-entity sentiment reasoning suffers most from shared context.

In general, generative summarization features batch well while structured extraction features require focused contexts, provides practical guidance for multi-feature agent design beyond media monitoring.

8 Related Work

Systems such as EMM (Steinberger, 2012), Event Registry (Leban et al., 2014), SUMMA (Germann et al., 2018), and Monitio (Amaral and Miranda, 2022) provide multilingual ingestion, entity extraction, and event clustering over high-volume news streams but rely on traditional NLP pipelines rather than LLM agents. MediaMind (Gunduz et al., 2025) introduced an agent-based media monitoring proof of concept on YouTube content at aiXplain; the present work extends this to a production-scale architecture with a watcher abstraction, hybrid cost-controlled matching, batched multi-feature extraction, and quantitative evaluation. Our system replaces key stages with LLM agents, enabling semantic matching and multi-feature extraction with explicit cost controls.

FrugalGPT (Chen et al., 2023) reduces LLM costs through cascades that route queries to cheaper models first. Our hybrid matching applies an analogous principle: keyword pre-filtering as a zero-cost gate before agent-based semantic classification, reducing agent invocations by $\sim$20%.

Recent LLM-based multi-agent architectures (Guo et al., 2024) such as AutoGen (Wu et al., 2024) and MetaGPT (Hong et al., 2024) demonstrate the benefits of task decomposition across specialized agents. Our system extends this with a two-tier design separating throughput-optimized pipeline agents from autonomous lifecycle agents.

Son et al. (Son et al., 2024) show that LLMs can follow multiple instructions in a single call with comparable quality. Our evaluation contributes empirical evidence on feature batchability, identifying a task-type-dependent pattern—summarization features batch well while structured extraction requires focused contexts—applicable to multi-feature agent design beyond media monitoring.

Embedding-based near-duplicate detection (Reimers and Gurevych, 2019) and HDBSCAN clustering (McInnes et al., 2017) are established techniques. Our contribution is an iterative, frequency-adaptive pipeline with composable strategies tailored to different reporting frequencies.

9 Conclusion

We presented a multi-agent media monitoring system organized around the watcher abstraction, a persistent, user-defined monitoring profile that unifies article classification, feature extraction, and report generation under a single configurable entity. The two-tier agent architecture separates throughput-optimized pipeline agents from autonomous lifecycle agents, providing a reusable pattern for continuous monitoring systems where both high-volume processing and adaptive configuration management are required.

The system's three design principles—staged processing for cost control, precomputed feature reuse, and selecting the right tool for each task (embeddings for structured operations, agents for semantic reasoning)—compound savings across

the pipeline. Our evaluation supports these choices: hybrid matching maintains F1 of 0.87 with $\sim$20% fewer agent calls, and two-call feature batching achieves a 71% reduction in feature-extraction calls with bounded quality tradeoffs. The key generalizable finding is that feature batchability is task-type-dependent: generative summarization features share comprehension context and batch well, while structured extraction features require focused contexts; a practical heuristic applicable to multi-feature agent design beyond media monitoring.

As future work, we plan to introduce human-in-the-loop feedback mechanisms that propagate user corrections back into watcher configurations and agent prompts. Additional directions include expanding source coverage beyond news articles (e.g., social media, broadcast transcripts), quantitative evaluation of the four lifecycle agents, and investigating adaptive batching strategies that adjust feature grouping based on article complexity or domain.

Limitations

At the system level, the matching and analysis agents inherit biases present in the underlying LLM's training data; geographic, political, and cultural biases can skew which articles are matched, how sentiment is scored, and how entities are characterized. Generated summaries, insights, and report narratives carry a risk of hallucinated facts or fabricated attributions, particularly for ambiguous or low-resource topics. The quality of the system's output is also bounded by the diversity and reliability of ingested news sources; underrepresented regions or languages may receive inadequate coverage.

On the evaluation side, our matching dataset covers nine watchers (96–250 labeled articles each; 1,634 instances) annotated by 3 annotators; scaling to more watchers, languages, and additional annotators would further strengthen generalizability. For batching, we aligned the GPT-4o judge with three expert annotators (N=2000; 77.5% exact match with majority labels; Table 3), but residual disagreements remain, and standard LLM-as-judge caveats (e.g., position bias, verbosity bias) still apply. The four lifecycle agents are described architecturally but not evaluated quantitatively. Our batchability findings are observed on a single LLM; different models may exhibit different sensitivities. The end-to-end latency and throughput of the system depend heavily on the serving infrastructure and LLM provider, and we leave systematic measurement under production load to future work.

Ethics Statement

Automated media monitoring systems can significantly enhance organizations' ability to track policy developments, emerging risks, and public discourse at scale. At the same time, responsible deployment is essential. Because the system relies on large language models, outputs may reflect biases present in training data, potentially influencing which articles are matched, how sentiment is characterized, or how entities are described. As with any generative system, summaries and insights may occasionally contain inaccuracies, particularly for ambiguous or low-context topics.

Classification and filtering decisions also shape the information presented to users. Misconfigurations or overly narrow keyword definitions may lead to incomplete coverage, underscoring the importance of careful watcher design and periodic review. Additionally, entity-level aggregation over time should be handled thoughtfully, particularly when monitoring involves individuals or sensitive topics.

This system is intended to support legitimate media intelligence use cases such as policy tracking, market analysis, and risk monitoring. Deployments should incorporate appropriate governance practices, including access controls, audit mechanisms, and human oversight, to ensure responsible and transparent use.

References

Carlos Amaral and Sebastião Miranda. 2022. Monitio-large scale mt for multilingual media monitoring. In *Proceedings of the 23rd Annual Conference of the European Association for Machine Translation*, pages 363–364.

Lingjiao Chen, Matei Zaharia, and James Zou. 2023. Frugalgpt: How to use large language models while reducing cost and improving performance. *arXiv preprint arXiv:2305.05176*.

Ulrich Germann, Renārs Liepins, Guntis Barzdins, Didzis Gosko, Sebastiao Miranda, and David Nogueira. 2018. The summa platform: a scalable infrastructure for multi-lingual multi-media monitoring. In *Proceedings of ACL 2018, System Demonstrations*, pages 99–104.

Ahmet Gunduz, Kamer Ali Yuksel, and Hassan Sawaf. 2025. Mediamind: Revolutionizing media monitoring using agentification. *arXiv preprint arXiv:2502.12745*.

Taicheng Guo, Xiuying Chen, Yaqi Wang, Ruidi Chang, Shichao Pei, Nitesh V Chawla, Olaf Wiest, and Xiangliang Zhang. 2024. Large language model based multi-agents: A survey of progress and challenges. *arXiv preprint arXiv:2402.01680*.

Sirui Hong, Mingchen Zhuge, Jonathan Chen, Xiawu Zheng, Yuheng Cheng, Ceyao Zhang, Jinlin Wang, Zili Wang, Steven Ka Shing Yau, Zijuan Lin, Liyang Zhou, Chenyu Ran, Lingfeng Xiao, Chenglin Wu, and Jürgen Schmidhuber. 2024. MetaGPT: Meta programming for a multi-agent collaborative framework. In *Proceedings of the Twelfth International Conference on Learning Representations (ICLR 2024)*. Oral presentation; arXiv:2308.00352.

Gregor Leban, Blaz Fortuna, Janez Brank, and Marko Grobelnik. 2014. Event registry: learning about world events from news. In *Proceedings of the 23rd international conference on World Wide Web*, pages 107–110.

Leland McInnes, John Healy, Steve Astels, and 1 others. 2017. hdbscan: Hierarchical density based clustering. *J. Open Source Softw.*, 2(11):205.

Nikolaos Panagiotou, Antonia Saravanou, and Dimitrios Gunopulos. 2021. News monitor: a framework for exploring news in real-time. *Data*, 7(1):3.

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)*, pages 3982–3992.

Guijin Son, SangWon Baek, Sangdae Nam, Ilgyun Jeong, and Seungone Kim. 2024. Multi-task inference: Can large language models follow multiple instructions at once? In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 5606–5627.

Ralf Steinberger. 2012. A survey of methods to ease the development of highly multilingual text mining applications. *Language Resources and Evaluation*, 46(2):155–176. Describes the Europe Media Monitor (EMM) family of applications.

Qingyun Wu, Gagan Bansal, Jieyu Zhang, Yiran Wu, Beibin Li, Erkang Zhu, Li Jiang, Xiaoyun Zhang, Shaokun Zhang, Jiale Liu, Ahmed Hassan Awadallah, Ryen W. White, Doug Burger, and Chi Wang. 2024. AutoGen: Enabling next-gen LLM applications via multi-agent conversation. In *ICLR 2024 Workshop on LLM Agents*. Best Paper Award; arXiv:2308.08155.