Materials, Methods and Results

Multi-Agent System for Automated Fact-Checking of YouTube Videos

Begoña Echavarren Sánchez

Tutor: Josep-Anton Mir Tutusaus

Master's Degree in Data Science
Universitat Oberta de Catalunya

PEC 3 - Implementation

December 2025

Contents

1 Materials and Methods

This section presents the comprehensive technical implementation of Factible, a multi-agent system for automated fact-checking of YouTube videos. The complete source code is publicly available at https://github.com/begoechavarren/factible. The system implements an end-to-end pipeline that processes video content through five specialized components, leveraging large language models (LLMs) for reasoning tasks while employing classical algorithms for deterministic operations. The implementation follows design-science principles [8, 13], emphasizing the creation of artifacts that extend human capabilities through systematic evaluation and iterative refinement.

1.1 System Architecture Overview

1.1.1 High-Level Architecture

Factible implements an end-to-end automated fact-checking pipeline for YouTube videos using a multi-agent architecture. Recent research on LLM agents demonstrates that multi-agent collaboration can enhance factuality and reasoning by allowing specialized agents to converse and coordinate on tasks [19]. FactAgent further shows that decomposing fact-checking into dedicated agents for input ingestion, query generation, evidence retrieval, and verdict prediction yields higher accuracy and transparency [21]. The Factible architecture follows this line of work by processing video content through five specialized, modular components that operate sequentially with three levels of internal parallelization.

The system processes a YouTube video URL through five sequential stages, each with specialized responsibilities. The pipeline begins with transcript extraction, proceeds through claim and query generation, conducts online evidence retrieval, and culminates in structured verdict synthesis. This modular design enables independent optimization of each component while maintaining clear data contracts between stages.

The five stages are:

1. **Transcriptor**: Extracts video transcripts via YouTube Transcript API, preserving timestamped segments for claim localization. The component includes automatic fallback to proxy service when rate-limited, ensuring robust transcript retrieval across different access conditions.

2. **Claim Extractor** (LLM Agent): Employs thesis-first reasoning to infer the video's central argument before extracting factual and verifiable claims. Each claim receives an importance score based on its impact on the video's thesis. Post-processing uses fuzzy string matching to locate claims within the transcript for timestamp mapping.

3. **Query Generator** (LLM Agent): Generates diverse search queries across four strategic types—direct, alternative, source-seeking, and contextual. Each query receives a priority score (1–5) based on evidence likelihood, enabling budget-conscious filtering of low-priority queries.

4. **Online Search**: Executes a four-step evidence retrieval pipeline for each query: (i) Google Search via Serper API, (ii) website reliability assessment using Media Bias/Fact Check (MBFC) data combined with domain heuristics [12], (iii) content fetching via Selenium WebDriver with JavaScript rendering support, and (iv) LLM-based evidence extraction with stance classification (supports, refutes, mixed, unclear).

5. **Output Generator** (LLM Agent): Synthesizes evidence into structured verdicts by building evidence bundles grouped by stance, generating natural language summaries with confidence levels, calculating algorithmic evidence quality scores, and mapping claims to video timestamps for interactive navigation.

Figure 1 illustrates the complete pipeline architecture with data flow and parallelization points across all five stages.

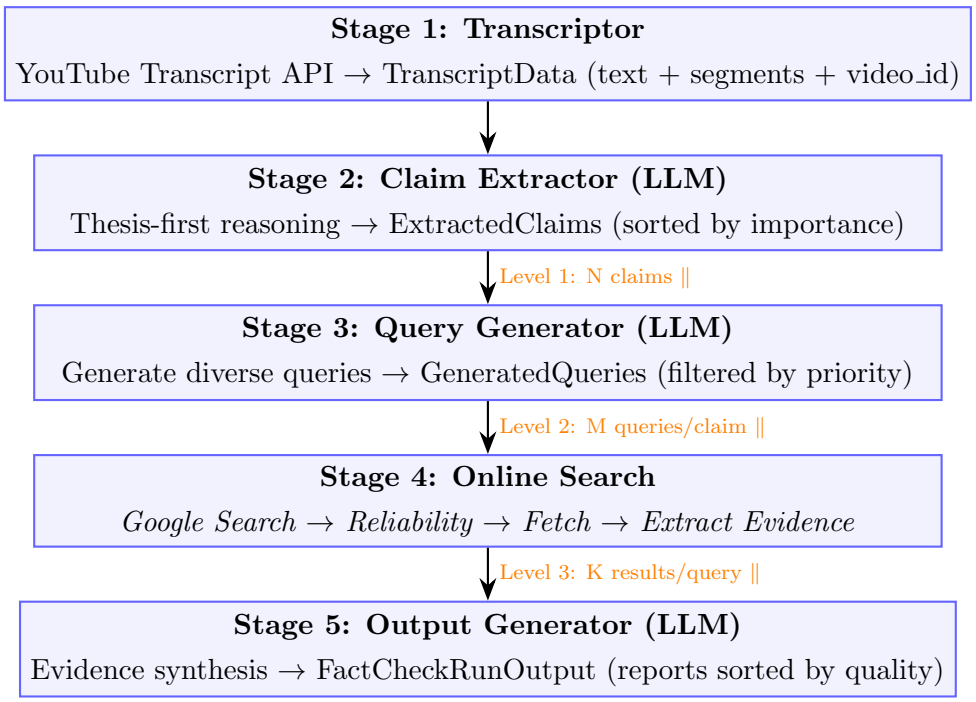


Figure 1: High-level pipeline architecture showing the five main stages and three parallelization levels. Parallelization occurs at claims (Level 1), queries per claim (Level 2), and search results per query (Level 3). Verdict generation happens within Level 1 after each claim's evidence is collected.

1.1.2 Design Principles

The system adheres to several key design principles derived from software engineering best practices and GenAI application development, aligned with the design-science paradigm in information systems research [8]:

1. **Modularity**: Each component is isolated with well-defined inputs and outputs using Pydantic schemas, enabling independent optimization, testing, and replacement.

2. **Structured Outputs**: All LLM interactions use Pydantic AI with typed output schemas, ensuring type safety, automatic validation, and consistent data structures across the pipeline.

3. **Transparency**: The full evidence chain is preserved and exposed to users—sources, reliability ratings, stances, and reasoning are all traceable from final verdict back to original source.

4. **Progressive Enhancement**: The pipeline operates with graceful degradation (e.g., fallback to snippet if scraping fails, fallback to proxy if rate-limited) rather than failing entirely.

5. **Cost-Conscious Design**: Configurable limits (`max_claims`, `max_queries`, `max_results`) prevent runaway API costs during development and production.

6. **Reproducibility**: Deterministic LLM outputs (`temperature=0.0`), structured YAML configurations, and comprehensive experiment tracking enable reproducible research.

7. **Separation of Concerns**: Classical algorithms handle tasks like reliability scoring, deduplication, and quality calculation, reserving LLM calls for tasks requiring reasoning and language understanding.

1.1.3 Component Isolation Pattern

Each component follows a consistent directory structure that promotes modularity and maintainability. This standardized organization enables independent testing of each component in isolation, seamless swapping of LLM models for experimentation, automated metrics collection via decorators, and clear interface contracts through Pydantic schemas. The typical structure includes a public exports file (`__init__.py`), main logic file with tracking decorators (`component_name.py`), and schema definitions file (`schemas.py`) containing all input and output Pydantic models. This separation of concerns facilitates parallel development and reduces coupling between pipeline stages.

1.2 Technology Stack

1.2.1 Core Technologies

Table 1 presents the core technologies employed in the implementation.

Table 1: Core technology stack

Category	Technology	Version	Purpose
Language	Python	3.12	Core implementation with type hints
LLM Framework	Pydantic AI	$\geq$1.0.0	Agent orchestration, structured outputs
Data Validation	Pydantic	$\geq$2.0.0	Schema definitions, runtime validation
Web Framework	FastAPI	$\geq$0.115.0	REST API with SSE streaming
HTTP Server	Uvicorn	$\geq$0.32.0	High-performance ASGI server
Async HTTP	httpx	$\geq$0.28.1	Async HTTP client
Web Scraping	Selenium	$\geq$4.15.2	JavaScript-rendered content extraction
YouTube	youtube-transcript-api	$\geq$1.2.2	Transcript extraction
Domain Info	python-whois	$\geq$0.8.0	Domain age lookup
CLI	Typer	$\geq$0.15.0	Experiment runner CLI
Analysis	pandas, matplotlib	-	Data analysis and visualization

1.2.2 Large Language Models

The system supports multiple LLM providers to enable comparison of cost-quality trade-offs. Table 2 shows the available models and their configurations.

Table 2: LLM providers and pricing

Provider	Model	Context	Pricing (per 1M tokens)	Use Case
OpenAI	gpt-4o-mini	128K	\$0.15 / \$0.60	Default
OpenAI	gpt-4o	128K	\$5.00 / \$15.00	High-quality
OpenAI	gpt-4-turbo	128K	\$10.00 / \$30.00	Premium
Ollama	qwen3:8b	40K	Free (local)	Budget/offline
Ollama	qwen3:4b	256K	Free (local)	Small footprint

Large language models such as GPT-4 offer multimodal capabilities and demonstrate human-level performance across diverse benchmarks [14]. Despite these advances, models still suffer from hallucinations and are constrained by limited context windows, underscoring the need for careful configuration and reliability safeguards [14]. The model management layer therefore emphasizes deterministic outputs, context-aware trimming, and tool-assisted generation.

1.2.3 External Services

The system integrates with external services for search and transcript extraction:

- **Serper API**: Google Search wrapper providing organic search results with approximately 2,500 queries per month on the free tier.
- **YouTube oEmbed API**: Video metadata retrieval without authentication.
- **Webshare Proxy**: Rate limit bypass for transcript extraction with configurable proxy locations.

1.3 Pipeline Components

1.3.1 Transcriptor

The Transcriptor component extracts YouTube video transcripts with precise timestamp information for later claim-to-video mapping.

Implementation Details The transcriptor uses the `youtube-transcript-api` library to fetch available transcripts, with preference for English (`["en", "en-US"]`). When rate-limited by YouTube, it automatically falls back to a proxy service (Webshare). Key features include:

- **Timestamped Segments**: Each segment preserves `start` time and `duration` in seconds.
- **Character Position Mapping**: Enables mapping claim text positions back to video timestamps.
- **Title Fetching**: Uses YouTube oEmbed API to retrieve video title for context.
- **Proxy Fallback**: Automatic retry through Webshare proxy when rate-limited.

Output Schema The transcriptor outputs structured data using Pydantic models for type safety and validation. Each transcript segment contains text content, start time, and duration with non-negative constraints. The complete transcript includes the full text, a list of timestamped segments, and the video identifier.

Timestamp Mapping Algorithm The timestamp mapping function enables the system to locate where in the video each claim originates. The algorithm iterates through transcript segments sequentially, maintaining a cumulative character counter. When a character position falls within a segment's range, the function returns the corresponding video timestamp (start time and duration). This mapping is crucial for

the user interface, allowing users to jump directly to the video moment where a specific claim was made.

1.3.2 Claim Extractor

The Claim Extractor identifies factual, verifiable claims from video transcripts using LLM-based extraction with thesis-relative importance ranking. This approach builds on prior work in automated claim detection: supervised models trained on annotated political debates have been used to detect check-worthy claims [5], and end-to-end systems like ClaimBuster monitor public discourse and prioritize factual statements for manual fact-checking [10]. These systems show that focusing on salient, verifiable claims improves the efficiency of fact-checking pipelines.

LLM Configuration The claim extractor uses deterministic settings for reproducibility: temperature set to 0.0 to ensure consistent outputs across multiple runs, max tokens limited to 1,200 to control response length and latency, and automatic retry logic (3 attempts) to handle transient LLM failures gracefully.

Prompt Engineering Strategy: Thesis-First Approach The claim extractor employs a novel *thesis-first approach* with multi-step reasoning designed to prioritize claims most critical to the video's central argument:

Step 1: Thesis Inference — Before listing claims, the LLM infers the video's central thesis in no more than 25 words (e.g., "Climate change alarmism is driven more by politics and media than by settled science").

Step 2: Importance Ranking with Thesis Impact Test — Claims are scored based on their impact on the video's thesis using the question: "If this claim were proven false, would the thesis collapse or materially weaken?" Table 3 presents the scoring guidelines.

Table 3: Claim importance scoring guidelines

Score Range	Description	Examples
0.85–1.0	Prescriptive/causal claims undermining thesis	Policy proposals, causal mechanisms
0.60–0.80	Quantitative/historical evidence tied to thesis	Statistics, dates, expert citations
0.30–0.55	Context/supporting background	Definitions, general facts
0.0–0.25	Peripheral/anecdotal details	Personal stories, credentials

Step 3: Relevance Guardrails — Pure credential facts are capped at 0.30 unless the thesis questions expertise; statements not affecting the thesis are capped at 0.25;

pure opinions are excluded; and paraphrases and duplicate numbers are removed.

Dynamic Instructions via Pydantic AI Dependencies The agent uses Pydantic AI's dependency injection mechanism to inject runtime constraints into the system prompt. This design pattern enables dynamic instruction generation based on runtime parameters: when a `max_claims` limit is specified, the instruction explicitly constrains the output size; otherwise, it defaults to requesting only the highest-impact claims. This approach provides flexibility for experimentation while maintaining type safety through Pydantic validation.

Post-Processing: Fuzzy Claim Localization After LLM extraction, each claim is located in the original transcript using fuzzy string matching. The algorithm normalizes text, applies a sliding window with ± 2 words around the claim length, computes similarity using `difflib.SequenceMatcher`, and requires a minimum score of 0.5 for a match. This produces `transcript_char_start`, `transcript_char_end`, and `transcript_match_score` fields for timestamp mapping.

Output Schema Extracted claims are represented as structured objects with validated fields: claim text (recommended maximum 40 words), confidence score (0.0–1.0), category (historical, scientific, statistical), importance score (0.0–1.0), and optional context. Post-processing adds transcript location metadata including character positions and fuzzy match scores. The collection of claims is sorted by importance in descending order.

Standard fact-checking datasets such as FEVER [18] and FEVEROUS [1] are used as external references to evaluate claim extraction performance. FEVER contains 185,445 claims derived from Wikipedia sentences labeled as Supported, Refuted, or NotEnoughInfo, while FEVEROUS extends this to 87,026 claims with both unstructured text and structured table evidence.

1.3.3 Query Generator

The Query Generator produces diverse, prioritized search queries optimized for evidence retrieval. Research on LLMs shows that interleaving reasoning and acting enables models to plan and perform external searches more effectively; the ReAct prompting technique encourages models to produce intermediate reasoning steps and task-specific actions, leading to improved factuality and reduced hallucination [20]. Retrieval-augmented generation methods combine parametric language models with non-parametric memory to retrieve relevant documents [9], while frameworks like RARR perform post-generation research and revision to align outputs with supporting evidence [3].

Query Type Taxonomy The system generates four types of queries with different search strategies, as shown in Table 4.

Table 4: Query type taxonomy

Type	Description	Strategy	Example
DIRECT	Exact claim phrasing	Verbatim search	"unemployment rose 15% Q3 2024"
ALTERNATIVE	Rephrased with synonyms	Semantic variation	"jobless rate increase third quarter"
SOURCE	Target authoritative sources	Source-seeking	"BLS unemployment statistics Q3"
CONTEXT	Broader context	Background search	"economic indicators fall 2024"

Priority System Queries are prioritized 1–5 based on likelihood of finding reliable, definitive information: priority 1 queries are always included, priority 2 by default, priority 3 if budget allows, and priorities 4–5 rarely or only for completeness.

Context-Aware Query Generation Beyond factual accuracy, the query generator is designed to detect misleadingly framed claims—statements that may be technically accurate but presented without essential context. The system prompt instructs the LLM to respect temporal context when choosing keywords (e.g., including relevant years or qualifiers to avoid mixing eras), and to explicitly generate queries seeking counter-arguments or opposing views. This approach helps surface evidence that may qualify, limit, or contextualize the original claim, enabling more nuanced verdict generation. For example, a claim stating "unemployment dropped 20%" might be technically accurate for a specific quarter but misleading without the broader trend; context-aware queries would seek both confirming statistics and broader economic context.

Output Schema Generated queries are structured objects containing the query text, query type (direct, alternative, source, or context), and priority (1–5, with 1 being highest). The complete output includes the original claim reference, a filtered and sorted list of queries, and a count of total queries generated before filtering.

1.3.4 Online Search

The Online Search component implements a multi-step pipeline to retrieve, assess, and extract evidence from web sources with adaptive quality filtering. Unlike the other LLM-based components, Online Search orchestrates multiple classical algorithms

alongside a single LLM call for evidence extraction. This hybrid approach balances speed, reliability, and reasoning capabilities.

The reliability assessment combines domain-level heuristics with the Media Bias/-Fact Check (MBFC) methodology, which employs a comprehensive weighted scoring system to evaluate media outlets' ideological bias and factual reliability [12]. To mitigate hallucinations and ensure evidence quality, the system draws on research like SelfCheckGPT, which detects hallucinations by comparing multiple sampled responses and ranks passages by factuality [11]. The multi-agent retrieval strategy is further informed by systems such as FactAgent [21] and LoCal [2], where decomposing, reasoning, and evaluating agents iteratively refine answers and outperform baselines.

Figure 2 illustrates the four-step Online Search pipeline executed for each query.

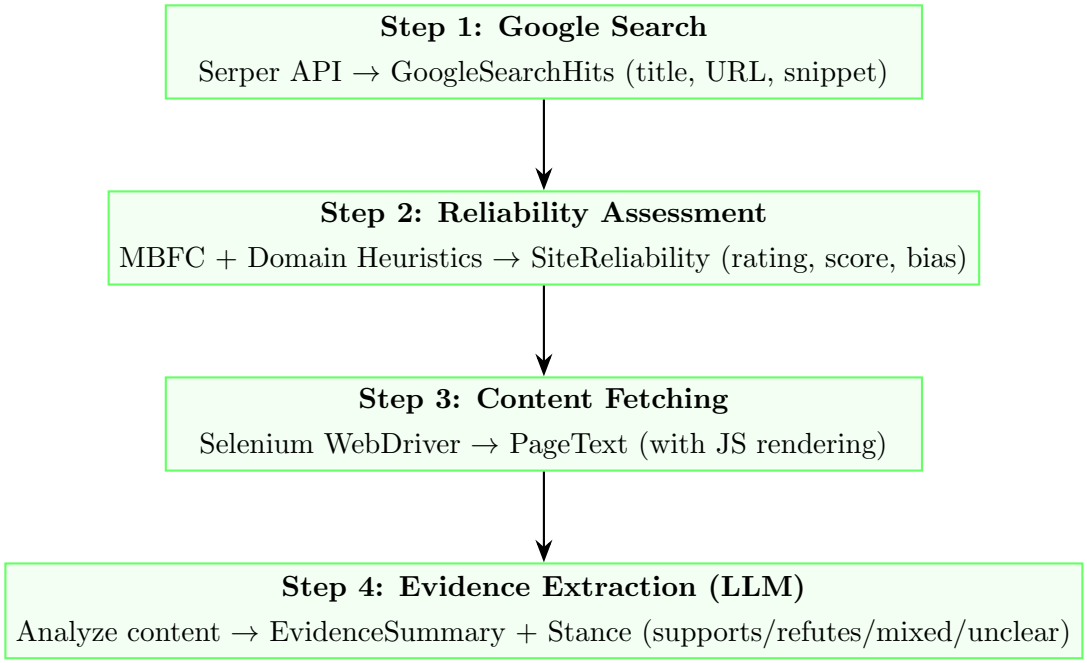


Figure 2: Online Search pipeline showing the four sequential steps executed for each search result. Steps 2–4 run in parallel across all K results per query (Level 3 parallelization).

Step 1: Google Search (Serper API) The Google search client wraps the Serper API for asynchronous search execution. The implementation uses persistent HTTP connections for performance and returns structured search hits containing title, URL, and snippet fields. Each query can retrieve up to 10 results, with the limit parameter controlling the exact number returned. The async design enables parallel query execution across multiple claims simultaneously.

Step 2: Website Reliability Assessment The reliability checker uses a multi-factor scoring system with first-match priority, combining external datasets with algorithmic heuristics:

- **Media Bias/Fact Check (MBFC) Dataset**: The system loads timestamped JSON snapshots containing nearly 10,000 news sources with credibility ratings (dataset extracted December 2025). Credibility mappings are: high $\rightarrow$ 0.85, medium $\rightarrow$ 0.60, low $\rightarrow$ 0.30, very low $\rightarrow$ 0.15.
- **TLD Reputation**: High-trust top-level domains (`.gov`, `.edu`, `.int`) receive a base score of 0.90, reflecting their institutional authority.
- **Domain Age via WHOIS**: Domains $\geq$10 years old receive a +0.10 bonus (established presence), while domains $<$1 year old receive a –0.15 penalty (recent creation may indicate lower trust).

The output includes a categorical rating (high, medium, low, unknown), numerical score (0.0–1.0), reasoning for the assessment, and political bias classification when available from MBFC data.

Step 3: Content Fetching (Selenium) Content fetching uses Selenium WebDriver in headless Chrome mode with smart wait strategies to handle JavaScript-heavy sites. The implementation employs a two-phase extraction strategy: first attempting quick extraction of paragraph elements, then waiting for JavaScript rendering if initial content is insufficient ($<$100 characters). This adaptive approach balances speed for static sites with completeness for dynamic content.

Configuration optimizations include disabling images to reduce load time, setting page load timeouts (20 seconds), and limiting wait times for dynamic content (12 seconds). Content is trimmed to a maximum of 8,000 characters to control LLM input costs while preserving sufficient context for evidence extraction. Async integration wraps blocking Selenium calls in `asyncio.to_thread()` for non-blocking operation, enabling parallel processing of multiple search results.

Step 4: Evidence Extraction (LLM) The evidence extractor analyzes retrieved content against claims using structured stance definitions:

- **SUPPORTS**: Evidence confirms or validates the claim through direct statements, semantic equivalents, or mechanism descriptions.
- **REFUTES**: Evidence contradicts or disproves the claim through counter-evidence or statements that evidence is unproven/disproven.

- **MIXED**: Both supporting and refuting elements present.
- **UNCLEAR**: Genuinely ambiguous content that discusses related topics without addressing the specific claim.

Critical prompt instructions ensure that mere discussion equals UNCLEAR, that mechanisms are recognized even without exact terminology, and that both Google snippets and page content are considered with better evidence prioritized. Importantly, the evidence extractor pays special attention to qualifiers such as "only", "never", "always", and temporal scope limitations (e.g., "since X date")—this enables detection of claims that may be technically accurate but misleadingly framed due to omitted context or overgeneralization.

Adaptive Credibility Filtering The search orchestrator implements adaptive credibility filtering as a key innovation for ensuring evidence quality. The algorithm operates in three phases:

1. **Initial Batch**: Fetch 2× the desired limit to provide filtering headroom
2. **Quality Check**: If >50% of results are unreliable, fetch an additional batch to increase the pool of high-quality sources
3. **Intelligent Filtering**: Sort all results by reliability score and select the top reliable sources, with a minimum guarantee ensuring at least some results are returned even if reliability is universally low

Additional filtering mechanisms include stance filtering (removing unclear results if >50% have definitive stances) and URL deduplication using hash sets to prevent duplicate sources across different queries for the same claim.

1.3.5 Output Generator

The Output Generator synthesizes evidence into coherent verdicts with confidence levels, quality scoring, and timestamp mapping.

Two-Step Process The Output Generator employs a hybrid approach combining algorithmic evidence organization with LLM-based synthesis:

Step 1: Build Evidence Bundle (Algorithmic) — The system groups evidence by stance (supports, refutes, mixed, unclear), deduplicates sources by URL, and sorts within each group by reliability rating (high first), then numerical score, with alphabetic tie-breaking for consistency. This deterministic organization ensures reproducible output ordering.

Step 2: Generate Verdict (LLM) — Organized evidence is formatted into a structured prompt containing stance labels, source counts, reliability ratings, and evidence summaries. The system prompt instructs the LLM to synthesize concise verdicts, naming sources explicitly only when clarifying contrasting perspectives or when evidence directly conflicts. This reduces verbosity while maintaining attribution transparency.

Evidence Quality Score (Algorithmic) The quality score is calculated algorithmically without LLM involvement for consistency and speed. The scoring formula combines three components with different weights: a base score of 0.3 for having any evidence, an actionable stance bonus of up to 0.3 (scaled by the number of supports/refutes/mixed sources, saturating at 3 sources), and a reliability bonus of up to 0.4 (scaled by the number of high/medium reliability sources, saturating at 3 sources). This design prioritizes both actionable stances and source reliability, with the maximum achievable score of 1.0 indicating high-quality, decisive evidence from multiple reliable sources.

Table 5 summarizes the quality score components.

Table 5: Evidence quality score breakdown

Component	**Weight**	**Criteria**
Base	0.3	Having any evidence
Actionable	0.3	Up to 3 supports/refutes/mixed sources
Reliability	0.4	Up to 3 high/medium reliability sources
Maximum	**1.0**	

Integrated Verdict Generation (within Level 1) Verdicts are generated immediately after each claim's evidence collection completes, within the same parallel execution context as the claim processing. This design choice eliminates the latency overhead of waiting for all claims to finish evidence collection before beginning verdict synthesis. Each claim's verdict generation executes as soon as its evidence is ready, allowing early-finishing claims to produce results while slower claims continue processing. After all parallel claim tasks complete, the reports are sorted by evidence quality score (descending) to prioritize high-confidence verdicts in the user interface.

Output Schema The fact-check report is a structured object containing all information for user presentation: claim metadata (text, confidence, category), verdict assessment (overall stance, confidence level, summary), evidence organization (grouped by stance with source summaries), quality metrics (total source count, evidence quality

score), and optional timestamp references for video navigation. Verdict confidence is constrained to low, medium, or high levels.

1.4 LLM Configuration and Model Management

1.4.1 Model Abstraction Layer

The system uses an enum-based model configuration with Pydantic validation for type-safe model management. Each model is defined through a `ModelConfig` schema containing provider name (OpenAI or Ollama), model identifier, input and output pricing per million tokens, and context window size. The `ModelChoice` enumeration defines available models as named constants (e.g., `OPENAI_GPT4O_MINI`, `OLLAMA_QWEN3_8B`), each associated with its configuration. This abstraction enables centralized model definitions, compile-time checking of model names, automatic price tracking for cost estimation, and easy addition of new models through enum extension.

1.4.2 Model Instantiation with Caching

Ollama model instances are cached using Python's `@lru_cache` decorator to enable connection reuse across multiple agent invocations. Without caching, each agent run would create a new provider connection, incurring initialization overhead. The cache is unbounded (`maxsize=None`), ensuring that once a model is instantiated, subsequent requests reuse the existing connection. This optimization is particularly important for Ollama models where the provider connection establishes communication with the local inference server at `http://127.0.0.1:11434/v1`.

1.4.3 Per-Component Model Configuration

A centralized configuration file defines which model each component uses, enabling easy model swapping for experiments. By default, all components use `OPENAI_GPT4O_MINI` for consistency, but this can be overridden on a per-component basis. For example, the Evidence Extractor could be downgraded to a local Ollama model to reduce costs while keeping the Claim Extractor on GPT-4o-mini for quality. This granular control enables component-specific optimization, A/B testing of model choices, and precise cost-quality trade-off analysis.

1.4.4 Common Model Settings

All components use `temperature=0.0` for deterministic outputs, enabling reproducibility, consistency in structured outputs, and meaningful A/B comparisons. Table 6 shows the token limits per component.

Table 6: Component model settings

Component	Temperature	Max Tokens	Rationale
Claim Extractor	0.0	1,200	Deterministic, structured claims
Query Generator	0.0	600	Concise queries, no explanations
Evidence Extractor	0.0	1,100	Summary + key quote
Output Generator	0.0	900	Concise verdict synthesis

1.5 Latency Optimization Strategies

The system implements multiple latency optimization strategies following established principles for LLM applications [15]. Research into LLM latency shows that prompt size, completion length, and model size are the dominant factors in inference time; reducing input and output token counts directly lowers compute and memory overhead [4, 15]. Additional techniques such as defining clear output boundaries, setting token limits, and adjusting sampling temperature can reduce generated tokens, and caching prompts allows reuse of computations for identical prefixes [15]. Choosing smaller model weights yields faster inference speeds. These optimizations can be grouped into seven core principles: processing tokens faster, generating fewer tokens, using fewer input tokens, making fewer requests, parallelizing operations, reducing perceived wait time, and using classical methods where LLMs are not required [15].

1.5.1 Process Tokens Faster: Model Selection

The default model (gpt-4o-mini) is selected for its balanced performance across speed, cost-effectiveness, and large context window (128K tokens). This model provides sufficient reasoning capabilities for fact-checking tasks while maintaining low latency and competitive pricing ($0.15 per million input tokens, $0.60 per million output tokens). For budget-conscious deployments or offline operation, local Ollama models (qwen3:8b, qwen3:4b) offer zero-cost inference at the expense of potential quality degradation. The modular model abstraction layer enables easy comparison of these trade-offs through experiment configuration.

1.5.2 Generate Fewer Tokens: Output Constraints

Each component has carefully tuned `max_tokens` limits to minimize generation latency and API costs without sacrificing information quality. Claims are limited to approximately 40 words (sufficient for most factual assertions), context descriptions to 20 words (brief background), and evidence summaries to 1–2 sentences (key findings only). These constraints are enforced through explicit prompt instructions and vali-

dated against output token limits. By preventing verbose outputs, the system reduces both generation time and downstream processing costs for subsequent pipeline stages.

1.5.3 Use Fewer Input Tokens: Content Trimming

Input token counts are minimized through aggressive content trimming strategies. Web content is trimmed to 6,000–8,000 characters before being passed to the Evidence Extractor, removing excessive context while retaining the most relevant portions (typically the first several paragraphs of an article). Evidence prompts include only Google snippets and extracted page text, explicitly excluding raw HTML, JavaScript, CSS, and other non-content elements that would inflate token counts without improving extraction quality. This targeted trimming reduces LLM input costs by an order of magnitude compared to naive full-page submission.

1.5.4 Make Fewer Requests: Combined Operations

The pipeline minimizes LLM API calls by combining operations into single requests wherever possible. Each component makes exactly one LLM call per input unit (one call for claim extraction, one per claim for query generation, one per search result for evidence extraction, and one per claim for verdict synthesis), with no multi-turn conversations that would multiply request counts. The Query Generator produces all queries for a claim in a single batch call rather than generating queries iteratively. Similarly, the Output Generator synthesizes verdicts with all available evidence in a single call. This design reduces API overhead, improves latency, and simplifies cost tracking.

1.5.5 Parallelize: 3-Level Async Architecture

The system implements three levels of nested parallelization to maximize throughput while maintaining dependency ordering. This architecture enables the pipeline to process multiple claims, queries, and search results simultaneously, dramatically reducing total execution time compared to sequential processing.

The parallelization hierarchy operates as follows:

- **Level 1 (Claims)**: After extracting N claims from the transcript, all claims are processed in parallel. Each claim independently proceeds through query generation, evidence search, and verdict generation. Critically, each claim's verdict is generated immediately after its evidence collection completes, rather than waiting for all claims to finish—this optimization reduces perceived latency by producing results progressively.

- **Level 2 (Queries per Claim)**: Within each claim's processing, the Query Generator produces M queries. These queries are executed in parallel, enabling simultaneous search across different query formulations (direct, alternative, source-seeking, contextual).
- **Level 3 (Search Results per Query)**: Within each query's execution, the Online Search component retrieves K results from Google. The four-step pipeline (reliability assessment, content fetching, and evidence extraction) runs in parallel for all K results, with each result processed independently.

This design achieves maximum theoretical parallelization of $N \times M \times K$ operations during the search phase, bounded only by system resources and API rate limits. The nesting structure means that at peak execution, the system may be processing dozens of parallel operations across all three levels simultaneously. Blocking I/O operations (Selenium WebDriver for content fetching, WHOIS for domain age lookup) are wrapped in `asyncio.to_thread()` to avoid blocking the event loop, ensuring that CPU-bound and I/O-bound operations can execute concurrently.

1.5.6 Real-Time Streaming

Server-Sent Events (SSE) provide progressive updates as the pipeline executes, improving perceived responsiveness for users. The streaming endpoint emits progress updates at key milestones, ranging from 5% (transcript extraction initiated) through 100% (fact-checking complete). Importantly, extracted claims are streamed to the user interface immediately after the Claim Extractor completes, allowing users to preview claims and begin reviewing the video's assertions before the time-intensive search phase completes. This progressive disclosure pattern reduces perceived latency and enables users to provide early feedback or cancellation if the extracted claims are not aligned with their expectations.

1.5.7 Classical Methods for Non-Reasoning Tasks

Table 7 shows operations handled by classical algorithms rather than LLMs.

Table 7: Operations using classical methods

Operation	**Method**	**Rationale**
Reliability scoring	Rule-based + MBFC lookup	Faster, deterministic, no API cost
Claim localization	Fuzzy string matching	No LLM needed for text search
Evidence quality score	Algorithmic calculation	Consistent, fast, reproducible
URL deduplication	Hash set	O(1) lookup
Stance filtering	Threshold-based	Simple percentage check

1.6 Data Schemas and Structured Outputs

1.6.1 Pydantic AI Integration

All LLM agents use Pydantic models as their `output_type`, providing comprehensive type safety throughout the pipeline. This integration ensures automatic JSON parsing and validation of LLM outputs, graceful error handling with automatic retries on validation failures, full IDE support with autocomplete for all schema fields, and consistent serialization via `model_dump()` for logging and persistence. The agent configuration includes the model selection, output schema type, runtime dependencies type, model settings (temperature, token limits), system prompt, and retry count. This declarative configuration style reduces boilerplate while maintaining explicit control over agent behavior.

1.6.2 Schema Design Principles

The schema design follows five guiding principles to balance flexibility, safety, and maintainability. First, schemas prefer flat structures over deeply nested objects to simplify validation and reduce complexity. Second, optional fields use Python's union type syntax (`str | None`) for clarity. Third, `Literal` types constrain string fields to predefined values (e.g., stance must be "supports", "refutes", "mixed", or "unclear"). Fourth, all fields include descriptive names and `Field` descriptions for documentation and IDE hints. Fifth, numeric fields include validation constraints (e.g., `Field(ge=0.0, le=1.0)` for scores) to catch invalid data at runtime. These principles ensure robust data contracts across component boundaries.

1.7 Experimentation and Evaluation Framework

Evaluating LLM-based fact-checking systems presents unique challenges: outputs are non-deterministic, external dependencies (web search) introduce variability, and traditional benchmarks risk overfitting [17]. To address these challenges, a three-component experimentation framework was developed to capture complete execution traces, support batch experimentation, and compute performance metrics.

1.7.1 Framework Architecture

The framework follows a linear data flow through three stages:

1. **Experiment Runner**: Executes the fact-checking pipeline on configured videos, invoking the tracking module for each run.

2. **Tracking Module**: Captures all execution data—inputs, outputs, LLM calls, timing, and costs—saving structured artifacts for later analysis.

3. **Evaluator**: Computes performance metrics including comparisons against ground truth annotations, system efficiency measurements, and source quality assessments.

This separation of concerns enables independent iteration on each component while maintaining a consistent data contract between stages.

1.7.2 Tracking Module

The tracking module implements a singleton pattern with context manager support, enabling any pipeline component to log data without explicit parameter passing. When initialized, the tracker creates a timestamped run directory and registers itself as the global tracker. The context manager protocol ensures automatic saving on exit.

Each run generates structured artifacts containing: run configuration and parameters, complete records of all LLM calls with prompts, responses, latency and cost, final extracted claims and fact-check verdicts, aggregated timing and cost metrics, and the original video transcript for reference.

LLM call tracking is achieved via a decorator that instruments the Pydantic AI agent methods, automatically recording component name, model, timestamp, latency, token counts, and calculated cost for every inference call.

1.7.3 Experiment Runner

The experiment runner enables batch execution through YAML configuration files that define video corpora and parameter sweeps. Videos are specified with metadata and tags for filtering, while experiments define parameter variations that automatically expand into multiple runs (e.g., testing `max_claims` values of 1, 3, 5, 7, and 10 generates five separate experiment runs).

A command-line interface provides commands for running experiments, filtering by experiment name or video ID, and previewing configurations before execution.

1.7.4 Evaluator

The evaluator computes performance metrics using modular components for each evaluation dimension:

- **Claim extraction metrics**: Precision@k, Recall, F1, and MAP computed using semantic similarity matching between extracted and ground truth claims.
- **Verdict accuracy**: Comparison of system stances against ground truth labels.
- **Evidence retrieval metrics**: Success rate and source reliability distribution based on MBFC credibility ratings.

- **System efficiency**: Latency and cost aggregation across all pipeline components.
- **LLM-as-judge evaluation** (optional): Qualitative assessment of claim relevance, evidence quality, and verdict coherence using LLM-based evaluation [16].

Evaluations execute in parallel across multiple videos for efficiency. Results include per-video reports and aggregate statistics (means, standard deviations, distributions) across all evaluated videos.

This infrastructure enabled an iterative development cycle: run experiments, evaluate metrics, identify issues, adjust parameters, and repeat—producing the results reported in Section 2.

1.8 API Layer and Real-Time Streaming

1.8.1 FastAPI Setup

The API uses FastAPI with CORS middleware configured for local frontend development, supporting common development server ports. API routes are organized under a versioned prefix to enable future API evolution without breaking existing clients.

1.8.2 Streaming Endpoint with SSE

The streaming endpoint (`POST /api/v1/fact-check/stream`) returns a `StreamingResponse` with `text/event-stream` media type. A callback-based progress handler collects updates into an asyncio queue, which are then yielded as SSE events.

1.8.3 Progress Events

Table 8 shows the progress events emitted during pipeline execution.

Table 8: SSE progress events

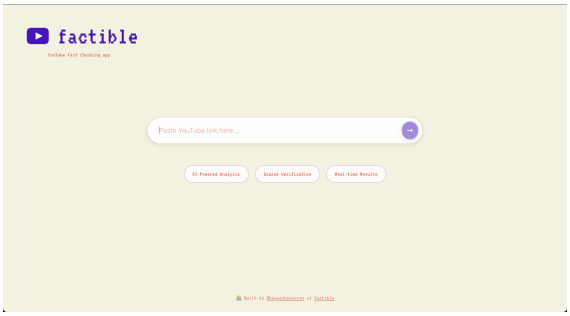
Stage	Progress	Event Name	Data Payload
1	5%	transcript_extraction	–
2	15%	transcript_complete	transcript_length
3	20%	claim_extraction	–
4	35%	claims_extracted	claims[], total_claims
5	90%	generating_report	–
6	100%	complete	result (full output)
Error	100%	error	error message

1.8.4 Request/Response Schemas

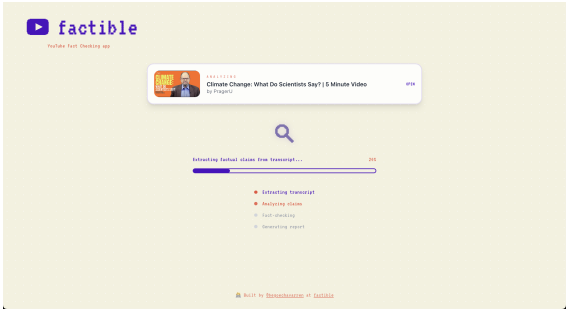
Request schemas validate YouTube video URLs and pipeline parameters (claim limits, query limits, results per query) with appropriate range constraints. Progress update schemas structure the SSE events with step identification, descriptive messages, progress percentages (0–100), and optional data payloads for intermediate results.

1.9 User Interface

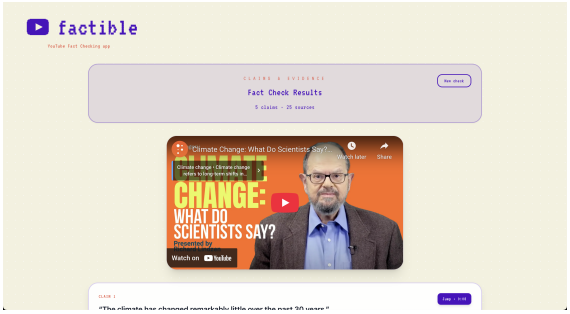
A web-based frontend provides an accessible interface for end users to interact with the fact-checking pipeline. The interface is built with React and communicates with the backend via the streaming API endpoint. Figure 3 presents the four main interface states.



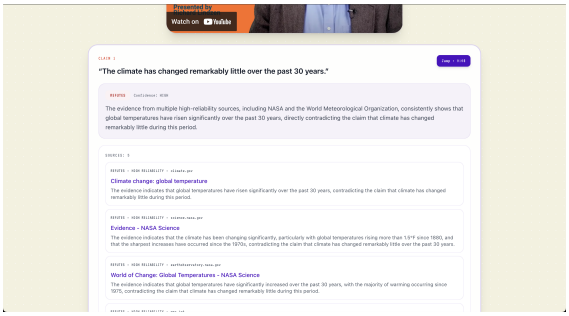
(a) Landing page with YouTube URL input



(b) Real-time processing with progress updates



(c) Results overview with embedded video player



(d) Detailed claim view with verdict and sources

Figure 3: User interface screenshots showing the fact-checking workflow: (a) users paste a YouTube URL, (b) real-time progress is displayed during analysis, (c) results are presented alongside the embedded video, and (d) each claim shows its verdict, confidence level, and supporting evidence with source reliability ratings.

The interface emphasizes transparency by displaying source reliability ratings, confidence levels, and direct links to evidence sources. Users can click "Jump" buttons to navigate directly to the video timestamp where each claim was made, enabling quick verification of the original context.

1.10 Code Quality and Engineering Practices

1.10.1 Type Safety

The codebase enforces strong type safety through multiple mechanisms. All functions use Python 3.12 type annotations with modern union syntax (`str | None`) and generic types (`list[T]`), enabling static analysis and IDE assistance. Pydantic models provide runtime validation for all data structures crossing component boundaries, catching type mismatches and constraint violations at execution time. Additionally, static type checking via mypy with strict mode configuration runs in continuous integration, preventing type errors from reaching production. This layered approach combines the benefits of gradual typing with runtime safety nets.

1.10.2 Logging

Module-level loggers provide structured logging throughout the pipeline execution. Each component initializes a logger using Python's standard logging module, configured with consistent formatting for timestamp, level, and message. Log messages use semantic levels (INFO for normal progress, WARNING for recoverable issues, ERROR for failures) and include contextual information such as claim indices, query counts, and reliability scores. The logging output supports both real-time monitoring during development and post-hoc analysis from experiment run files.

1.10.3 Error Handling Patterns

Table 9 summarizes error handling strategies.

Table 9: Error handling patterns

Error Type	Handling	Fallback
No transcript	Return empty claims	Continue with empty pipeline
LLM failure	Retry 3x	Return empty/unclear
Search API failure	Log and continue	Empty results for query
Scraping failure	Fall back to snippet	Use Google snippet
Reliability failure	Default to "unknown"	Conservative rating
Verdict failure	Error message in summary	Unclear stance

1.10.4 Configuration Management

Configuration is managed through multiple layers optimized for different use cases. Sensitive credentials (API keys for Serper, OpenAI, Webshare) are stored in environment variables loaded from a `.env` file, following security best practices by keeping

secrets out of version control. Experiment parameters (max claims, queries per claim, results per query, video corpus) are defined in YAML files for human readability and version control tracking. Model settings (temperature, token limits, retry counts) are specified as Python constants for type safety and inline documentation. Finally, API server configuration (CORS origins, port, host) uses Pydantic Settings with automatic environment variable override support. This layered approach balances security, flexibility, and maintainability.

1.10.5 Pre-commit Hooks

Code quality is enforced automatically through pre-commit hooks that run before each git commit. The ruff linter performs fast Python linting with automatic fixes for common issues (unused imports, trailing whitespace, line length violations) and consistent code formatting following PEP 8 style guidelines. The mypy static type checker validates type annotations across the entire codebase in strict mode, catching potential type errors before runtime. Additional standard hooks validate YAML and TOML file syntax, preventing configuration errors. These automated checks ensure code quality standards are maintained consistently across all contributions without requiring manual review for style issues.

1.11 Design Patterns and Software Engineering

1.11.1 Patterns Summary

Table 10 summarizes the design patterns employed.

Table 10: Design patterns used in the implementation

Pattern	**Usage**	**Benefit**
Singleton	ExperimentTracker	Global access during pipeline
Context Manager	Tracker, Timer, Fetcher	Resource cleanup, timing
Decorator	@track_pydantic	Cross-cutting concerns
Monkey-Patching	Agent monitoring	Non-invasive instrumentation
Factory	get_model()	Model instantiation
Strategy	Adaptive filtering	Runtime algorithm selection
Observer	Progress callbacks	Decoupled progress reporting
Builder	Evidence bundle	Complex object construction

1.11.2 Async Patterns

The implementation employs four asynchronous programming patterns to maximize concurrency. First, parallel independent tasks use `asyncio.gather()` to execute multiple coroutines concurrently and collect their results. Second, thread pools handle blocking I/O operations (Selenium, WHOIS) via `asyncio.to_thread()`, preventing them from blocking the event loop. Third, queue-based producer/consumer patterns enable streaming results from background workers to the API endpoint. Fourth, background task spawning via `asyncio.create_task()` allows fire-and-forget operations that don't block the main execution flow. These patterns combine to create a highly concurrent pipeline that efficiently utilizes system resources.

1.11.3 Key Engineering Decisions

Several architectural decisions significantly shaped the implementation. Pydantic AI was chosen over LangChain for its simpler API surface and native support for type-safe structured outputs through Pydantic models. Selenium was selected over lightweight HTTP libraries like requests specifically to handle JavaScript-rendered content that requires browser execution. The MBFC dataset provides authoritative reliability ratings based on established fact-checking methodology, offering more credible assessments than simple domain heuristics alone. The `asyncio.to_thread` pattern enables non-blocking integration of synchronous libraries (Selenium, WHOIS) within the async pipeline. Temperature 0.0 ensures reproducible experiments by eliminating sampling randomness. Finally, token estimation using character count divided by 4 provides a computationally cheap approximation for cost tracking without requiring exact tokenization.

2 Results

This section presents the experimental evaluation of Factible across 30 YouTube videos spanning diverse topics including health, science, politics, and climate. The evaluation framework follows established practices from claim detection and fact-checking research [5, 18], combining ground truth comparison with LLM-as-judge quality assessments. All experiments used `max_claims=5` as the primary operating point, selected after exploring the precision-recall tradeoff across multiple configurations while balancing cost and latency constraints.

2.1 Experimental Setup

2.1.1 Evaluation Dataset

The evaluation corpus consists of 30 YouTube videos manually annotated with ground truth claims. This sample size is comparable to evaluation scales used in end-to-end fact-checking systems; for example, ClaimBuster evaluated their system on 25 presidential debates [7]. Videos were selected across three thematic categories—climate, health, and political/social issues—with 10 videos per category to ensure balanced representation. Within each category, videos were equally split between factual content (5 videos) and misinformation (5 videos), allowing evaluation of the system's performance across different truth orientations. Videos range from educational science content to political commentary, representing diverse domains and claim densities. Table 11 summarizes the dataset characteristics.

Table 11: Evaluation dataset statistics

Metric	**Value**
Total videos	30
Ground truth claims per video (mean)	16.8
Ground truth claims per video (range)	9–35
Total ground truth claims	503
System claims extracted per video	5

2.1.2 Ground Truth Annotation

Ground truth annotations were created following a structured protocol. For each video, all factual, verifiable claims from the transcript were manually annotated. Each claim was annotated with an importance score (0.0–1.0) reflecting its centrality to the video's main argument, and a verdict label indicating the expected verification outcome (SUPPORTS, REFUTES, MIXED, or UNCLEAR). The annotation process followed guidelines from ClaimBuster's check-worthiness criteria [5], prioritizing claims that are specific, verifiable, and consequential. The complete evaluation dataset, including all 30 annotated videos with their ground truth claims, importance ratings, and expected verdicts, is provided in Appendix A.

2.1.3 Evaluation Metrics

The evaluation employs metrics at two levels: claim extraction quality and verdict accuracy.

Claim Extraction Metrics:

- **Precision@k**: Proportion of extracted claims matching any ground truth claim, using semantic similarity matching with a threshold of 0.7. This metric follows the standard information retrieval formulation used in claim detection systems [6].
- **Recall@k**: Proportion of ground truth claims matched by the top-k extracted claims [6].
- **F1 Score**: Harmonic mean of precision and recall.
- **Mean Average Precision (MAP)**: Ranking quality metric from information retrieval, measuring whether important claims appear early in the extraction order [6].
- **Recall@Important**: Recall specifically for high-importance claims (importance ≥ 0.80).
- **Importance-Weighted Coverage**: Percentage of total ground truth importance mass captured by matched claims.

Verdict Accuracy Metrics:

- **Stance Accuracy**: Classification accuracy for the four-class stance problem (SUPPORTS, REFUTES, MIXED, UNCLEAR), measuring the percentage of verdicts matching ground truth stance.

System Efficiency Metrics:

- **Latency**: End-to-end processing time per video.
- **Evidence Retrieval Rate**: Proportion of queries successfully retrieving evidence.
- **Source Reliability**: Distribution of source reliability ratings across retrieved evidence.

2.1.4 Component Evaluation Scope

The evaluation focuses on components where the system makes reasoning decisions that can be compared against ground truth. Specifically, the evaluation covers **Claim Extraction** (precision, recall, importance ranking) and **Verdict Generation** (stance accuracy, explanation quality) as these components employ LLM-based reasoning that can produce varying results.

Components relying on external APIs—**Transcript Extraction** (YouTube Transcript API) and **Online Search** (Serper/Google)—are not evaluated directly, as their

performance depends on third-party services rather than the system's design. However, their effectiveness is implicitly covered by the end-to-end evaluation: if transcript extraction fails, no claims can be extracted; if search fails, verdict accuracy degrades. The **Query Generator** component is assessed indirectly through evidence retrieval success rates, as effective queries should yield relevant evidence.

This evaluation strategy enables focused assessment of the system's core reasoning capabilities while acknowledging that external dependencies affect overall performance through the end-to-end metrics.

2.2 Precision-Recall Tradeoff Analysis

Before presenting detailed results, this section analyzes the precision-recall tradeoff to justify the choice of `max_claims=5` as the primary configuration. The system was evaluated across six configurations with `max_claims` $\in \{1, 3, 5, 7, 10, 15\}$.

Table 12: Precision-recall tradeoff across different `max_claims` configurations

max_claims	Precision	Recall	F1	MAP
1	0.800	0.052	0.098	0.800
3	0.822	0.159	0.264	0.897
5	**0.813**	**0.262**	**0.390**	**0.870**
7	0.795	0.359	0.486	0.862
10	0.769	0.471	0.573	0.854
15	0.719	0.570	0.623	0.865

Figure 4 illustrates the precision-recall tradeoff. As `max_claims` increases, recall improves from 5.2% to 57.0%, while precision decreases from 82.2% to 71.9%. The slight precision increase from $k = 1$ (80.0%) to $k = 3$ (82.2%) indicates that the system benefits from extracting multiple high-confidence claims rather than being forced to select exactly one. Beyond $k = 5$, the classic precision-recall tradeoff becomes pronounced.

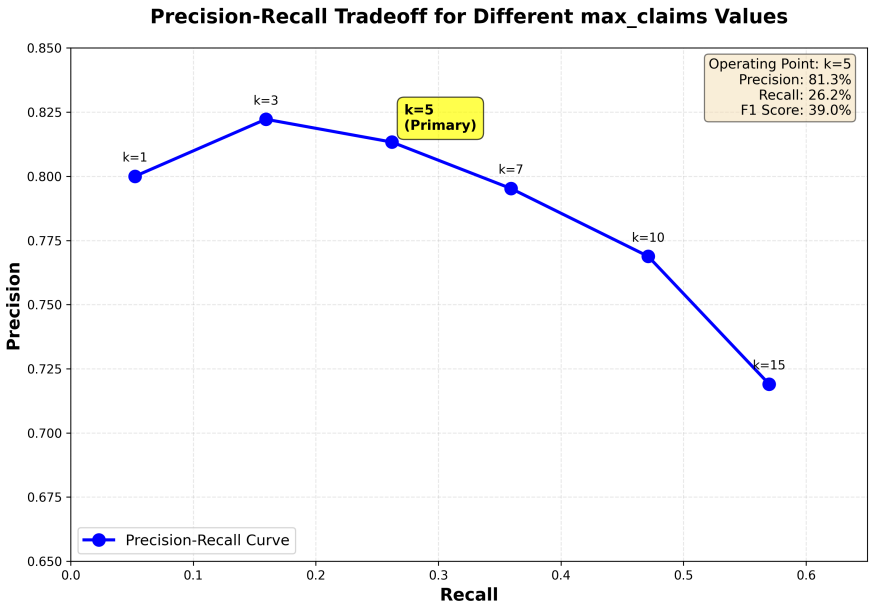


Figure 4: Precision-recall curve for different `max_claims` values. The selected operating point ($k = 5$) achieves 81.3% precision at 26.2% recall, balancing claim quality with coverage.

The configuration `max_claims=5` was selected as the primary operating point because it maintains high precision (81.3%) while achieving reasonable recall (26.2%), achieves a strong MAP score (0.870) indicating good ranking quality, and provides favorable cost and latency characteristics. Beyond `max_claims=5`, processing time and API costs increase linearly with claim count, while precision degrades. For a fact-checking application where user trust depends on accuracy and responsiveness, balancing precision, cost, and latency is critical—presenting 5 high-quality claims efficiently is more valuable than presenting 15 claims with higher false positive rates, increased costs, and longer wait times.

2.3 Claim Extraction Performance

Table 13 presents the claim extraction results at the primary configuration (`max_claims=5`).

Table 13: Claim extraction performance (n=30 videos, `max_claims`=5)

Metric	Mean	Std Dev
Precision@5	0.813	0.171
Recall	0.262	0.088
F1 Score	0.390	0.111
Mean Average Precision (MAP)	0.870	0.146
Recall@Important ($\geq$0.80)	0.395	0.178
Importance-Weighted Coverage	0.292	0.096
Importance MAE	0.126	0.060

2.3.1 Interpretation of Results

The system achieves 81.3% precision, meaning that approximately 4 out of 5 extracted claims on average match ground truth claims. This high precision indicates that the claim extractor successfully identifies legitimate factual claims rather than extracting irrelevant or fabricated statements.

The overall recall of 26.2% reflects the constraint of extracting only 5 claims from videos averaging 16.8 ground truth claims. However, the Recall@Important metric (39.5%) demonstrates that the system prioritizes high-importance claims—capturing nearly 40% of the most critical claims while only extracting approximately 30% of the total claim set. This selective extraction behavior aligns with the design goal of thesis-first reasoning, where claims are ranked by their impact on the video's central argument.

The MAP score of 0.870 indicates strong ranking quality: important claims consistently appear early in the extraction order. This metric, borrowed from ClaimBuster's evaluation framework [5], validates that the importance scoring mechanism effectively prioritizes claims.

The importance MAE of 0.126 (on a 0–1 scale) shows that system-assigned importance scores closely approximate human judgments, with typical errors of approximately one importance tier (e.g., scoring a claim 0.7 when ground truth is 0.85).

2.3.2 Contextualizing Recall

The 26.2% recall, while appearing low in isolation, must be interpreted in context. Given that videos contain an average of 16.8 checkable claims and the system extracts 5, a theoretical maximum recall of approximately 30% exists under this constraint. The achieved recall of 26.2% therefore represents strong performance relative to the configuration limit.

Furthermore, the importance-weighted coverage of 29.2% indicates that the system captures nearly one-third of the total "importance mass" of ground truth claims. For practical fact-checking applications where user attention is limited, presenting 5 high-quality, important claims provides more value than exhaustive but overwhelming coverage.

2.4 Verdict Generation Performance

Table 14 presents the verdict accuracy results.

Table 14: Verdict generation performance (n=30 videos)

Metric	**Mean**	**Std Dev**
Stance Accuracy	0.733	0.334

2.4.1 Accuracy Distribution Analysis

The standard deviation (0.334) in verdict accuracy warrants investigation. Analysis of per-video results reveals a distribution skewed toward high accuracy:

- **21 videos (70.0%)** achieved high accuracy ($\geq$75%)—the majority of extracted claims were correctly classified.
- **5 videos (16.7%)** achieved medium accuracy (25–74%)—partial verdict correctness.
- **4 videos (13.3%)** achieved low accuracy (<25%)—most claims were incorrectly classified.

Evidence Retrieval and Verdict Quality With an evidence retrieval success rate of 94.7%, the system consistently finds relevant sources for most claims. Table 15 illustrates how verdict accuracy varies across the dataset.

Table 15: Verdict accuracy distribution with evidence retrieval rates

Video Topic	Retrieval Rate	Verdict Acc.	Avg Sources
High accuracy (≥75%) — 21 videos			
Brain Benefits of Exercise (TED)	100%	100%	2.5
Climate Change (NatGeo)	100%	100%	2.3
Fossil Fuels	100%	100%	2.7
UK Election Results	100%	75%	2.8
Medium/Low accuracy (<75%) — 9 videos			
Gender Wage Gap	100%	50%	2.7
FBI & January 6th	100%	50%	2.3
Inflation Explainer	60%	25%	1.8
Immigration Statistics	100%	0%	2.9

Notably, successful evidence retrieval does not guarantee high verdict accuracy. Some politically contentious topics achieve 100% retrieval but lower verdict accuracy, suggesting that the challenge lies not in finding evidence but in correctly synthesizing conflicting sources or matching the ground truth annotator's interpretation.

Factors Affecting Verdict Accuracy Analysis of low-accuracy videos reveals several contributing factors:

- **Contested claims**: Topics with legitimate disagreement (e.g., wage gap interpretations, immigration statistics) may have evidence supporting multiple stances, making definitive verdicts challenging.
- **Ground truth subjectivity**: Some claims involve nuanced interpretations where reasonable annotators might disagree on the correct stance.
- **Evidence-claim mismatch**: Retrieved evidence may address related but not identical claims, leading to verdict errors.

These findings suggest that further improvements require enhanced evidence synthesis and more sophisticated handling of contested claims, rather than simply improving retrieval success.

2.4.2 Comparison to Random Baseline

The stance accuracy of 73.3% substantially exceeds a random baseline. For a four-class classification problem (SUPPORTS, REFUTES, MIXED, UNCLEAR), random

guessing would achieve approximately 25% accuracy. The system's 73.3% accuracy represents a 2.93× improvement over random, demonstrating meaningful verification capability. Moreover, unlike random classification, the system provides evidence-backed explanations that enable users to evaluate the verdict's reasoning.

2.5 Evidence Retrieval Performance

2.5.1 Retrieval Success

Table 16 summarizes evidence retrieval performance.

Table 16: Evidence retrieval performance (n=30 videos)

Metric	Value
Evidence retrieval success rate	94.7%
Average sources per query	1.52
Average evidence items per claim	2.41

The evidence retrieval success rate of 94.7% indicates that the vast majority of search queries return usable evidence. When evidence is retrieved, claims receive an average of 2.41 evidence items, providing multiple perspectives for verdict synthesis.

2.5.2 Source Reliability Distribution

A critical aspect of fact-checking is source quality. Table 17 presents the distribution of source reliability ratings across all retrieved evidence.

Table 17: Source reliability distribution (n=724 total sources)

Reliability Rating	Count	Percentage
High	605	83.6%
Medium	110	15.2%
Low	0	0.0%
Unknown	9	1.2%

The overwhelming majority of retrieved sources (83.6%) receive high reliability ratings from the Media Bias/Fact Check-based assessment system [12]. No sources received low reliability ratings, and only 1.2% were classified as unknown (typically due to missing MBFC data for niche domains). This distribution validates the effectiveness of the source credibility filtering, which prioritizes established, factual reporting sources.

2.6 System Efficiency

2.6.1 Processing Latency

Table 18 presents processing time statistics.

Table 18: System latency (n=30 videos)

Metric	Value
Mean latency	129.6 seconds
Standard deviation	101.8 seconds
Minimum latency	36.5 seconds
Maximum latency	643.7 seconds
Total processing time (30 videos)	64.8 minutes

The mean processing time of 129.6 seconds per video enables near-interactive use for individual videos. The high variance (standard deviation 101.8s) reflects multiple contributing factors:

- **Video length**: Longer transcripts require more LLM tokens for claim extraction, increasing inference time.
- **Pipeline parameters**: The configuration parameters `max_claims`, `max_queries`, and `max_results_per_query` directly multiply the number of downstream operations. With the evaluation configuration (max_claims = 5, max_queries = 3, max_results = 3), each video triggers up to $5 \times 3 \times 3 = 45$ evidence extraction operations.
- **Evidence retrieval success**: Videos with failed evidence retrieval complete faster (fewer web requests), while videos requiring multiple successful searches experience longer latencies.
- **Web scraping variability**: JavaScript-heavy sites require longer Selenium wait times, and some domains respond slower than others.

2.6.2 Cost Analysis

All experiments used GPT-4o-mini for LLM inference at current pricing ($0.15 per million input tokens, $0.60 per million output tokens). Across the 30-video evaluation corpus, the average cost per video was $0.003, with individual videos ranging from $0.0008 to $0.0164 depending on transcript length, claim complexity, and evidence retrieval needs. The total cost for processing all 30 videos was $0.09, demonstrating the practical affordability of the system for individual users. This cost-effectiveness

contrasts with concerns about LLM deployment costs noted in prior work [21], showing that fact-checking systems can achieve meaningful accuracy at minimal expense when using appropriately-sized models.

2.7 Qualitative Analysis

2.7.1 Successful Extraction Examples

Table 19 presents examples of successful claim extractions demonstrating semantic matching between ground truth and system-extracted claims.

Table 19: Examples of successful claim extraction with semantic matching

Ground Truth Claim	**System-Extracted Claim**	**Imp.**
A single workout immediately increases levels of neurotransmitters like dopamine, serotonin, and noradrenaline	A single workout increases levels of neurotransmitters like dopamine, serotonin, and noradrenaline	0.9
HIV infects one of the immune cells that is central to the body's response to pathogens—the helper T-cell	HIV infects helper T-cells, which are central to the immune response	0.95
Scientists at UCT have uncovered garlic's cancer fighting properties	Scientists at UCT uncovered garlic's cancer-fighting properties	0.95

These examples demonstrate that the system successfully extracts claims while allowing minor paraphrasing and condensation. The semantic similarity matching correctly identifies these as equivalent claims despite surface-level textual differences.

2.7.2 Error Analysis: Verdict Failures

Analysis of verdict errors reveals systematic patterns. With high evidence retrieval success (94.7%), the primary failure modes involve stance misclassification and handling nuanced claims where conflicting evidence requires domain expertise to synthesize correctly. Table 20 categorizes the primary error types.

Table 20: Verdict error categories

Error Type	Description
Evidence retrieval failure	No evidence retrieved; system defaults to UNCLEAR
Stance misclassification	Evidence retrieved but stance incorrectly assessed (e.g., MIXED classified as REFUTES)
Nuanced claims	Claims requiring domain expertise to evaluate mixed evidence

Error analysis reveals that evidence retrieval success does not guarantee verdict accuracy. Two of the four lowest-accuracy videos achieved 100% evidence retrieval but 0% verdict accuracy, indicating that the challenge lies in evidence synthesis rather than retrieval. These cases involve politically contested claims where ground truth stances require nuanced interpretation of conflicting sources.

2.7.3 Analysis of Low-Accuracy Cases

Only 4 videos (13.3%) achieved less than 25% verdict accuracy. Detailed analysis reveals their characteristics:

Table 21: Low-accuracy video analysis (<25% verdict accuracy)

Video Topic	Retrieval	Accuracy	Likely Cause
Trump's National Emergency	60%	0%	Political claims with contested interpretations
Immigration Statistics	100%	0%	Conflicting sources on contested statistics
Juice vs. Whole Fruit	20%	20%	Limited evidence retrieval
Sleep & Teenage Brain	100%	20%	Nuanced scientific claims

Key observations:

- **High retrieval does not guarantee accuracy:** Two videos achieved 100% retrieval but 0–20% accuracy, confirming that evidence synthesis is the bottleneck for contested claims.
- **No single category dominates:** Low-accuracy videos span both political (2) and health (2) topics, and include both factual content (3) and misinformation

(1).

- **Contested claims are hardest:** The common thread is claims where reasonable sources disagree or where ground truth requires nuanced interpretation.

Successful categories: Videos on scientific explanations (e.g., climate science, exercise benefits), health misinformation debunking (e.g., fluoride claims, detox myths), and conspiracy content (e.g., chemtrails, geoengineering) achieved high accuracy, demonstrating the system's effectiveness when evidence clearly supports or refutes claims.

2.8 Considerations for Generative AI Systems

It is important to contextualize these results within the unique characteristics of generative AI systems. Unlike traditional machine learning models with deterministic outputs, LLM-based systems introduce inherent variability that affects evaluation interpretation.

2.8.1 Non-Determinism in LLM Systems

Despite configuring all LLM calls with `temperature=0.0` to minimize output variability, complete determinism is not guaranteed. Even with zero temperature, LLM outputs may vary across runs due to:

- **Floating-point precision**: GPU computation introduces subtle numerical variations that can cascade through token selection.
- **Model updates**: Cloud-hosted models (e.g., GPT-4o-mini) may be silently updated by providers, affecting outputs over time.
- **Batching effects**: Different batch sizes or concurrent requests may influence internal state.

This inherent non-determinism means that exact reproduction of results is challenging, though setting temperature to zero substantially reduces variability compared to default settings.

2.8.2 External Dependencies and Temporal Sensitivity

Beyond LLM variability, the system's reliance on external web search introduces additional sources of result variability:

- **Search result volatility**: Web search results change over time as new content is indexed and rankings evolve.

- **Content availability**: Websites may become unavailable, paywalled, or block automated access.
- **Rate limiting**: Search APIs may throttle requests, causing some queries to fail during high-load evaluation runs.

These factors contribute to verdict accuracy variance: search results may differ between runs, and a claim that retrieved limited evidence in one execution might find more sources in another.

2.8.3 Implications for Metric Interpretation

Unlike traditional classification tasks where metrics are stable given fixed test data, LLM-based systems produce metrics with inherent variance. When evaluating generative AI systems, researchers should consider:

- **Expected variability**: Metrics may vary by several percentage points across identical evaluation runs.
- **Qualitative validation**: Beyond aggregate metrics, examining individual outputs provides crucial insight into system behavior.
- **Temporal context**: Results reflect system performance at a specific point in time with then-current search results and model versions.

The strategies employed in this work to maximize reproducibility—temperature=0.0 for all LLM calls, fixed random seeds, comprehensive logging, and experiment versioning—represent current best practices for LLM evaluation but cannot eliminate all sources of variability.

2.9 Summary of Key Findings

The experimental evaluation demonstrates that Factible achieves reliable fact-checking performance across diverse video content:

1. **High-precision claim extraction:** 81.3% precision with strong importance ranking (MAP 0.870), meaning users can trust that presented claims are legitimate, high-priority factual statements.
2. **Robust evidence retrieval:** 94.7% success rate with 83.6% of sources from high-reliability origins, demonstrating effective query generation and source filtering.

3. **Consistent verdict accuracy:** 73.3% overall accuracy, with 70% of videos (21/30) achieving ≥75% accuracy. This represents a 2.93× improvement over random baselines.

4. **Identified challenges:** The 4 low-accuracy videos (13.3%) involve politically contested claims with legitimate disagreement, where evidence synthesis rather than retrieval poses the challenge.

5. **Practical efficiency:** 129.6 seconds mean latency at $0.003 per video enables cost-effective, near-interactive use.

These results position Factible as a reliable tool for preliminary fact-checking of YouTube content. The system performs well on scientific, health, and clearly verifiable claims, while contested political topics with conflicting evidence sources represent the primary remaining challenge for future work.

3 Future Work

While the current implementation demonstrates the viability of automated fact-checking for YouTube videos, several directions warrant further investigation to enhance the system's capabilities and broaden its applicability.

3.1 Multilingual Support

The current system operates exclusively with English-language content, limiting its applicability in a globally connected information ecosystem. Extending support to additional languages would require: (1) multilingual transcript extraction, as YouTube provides auto-generated transcripts in multiple languages; (2) multilingual claim extraction, leveraging multilingual LLMs such as GPT-4 or open-source alternatives like mBART and XLM-RoBERTa; (3) cross-lingual evidence retrieval, enabling the system to find evidence in languages different from the claim's source language; and (4) multilingual verdict synthesis, generating explanations in the user's preferred language. Languages with high misinformation prevalence (Spanish, Portuguese, Hindi, Arabic) should be prioritized for maximum impact.

3.2 Large Language Model Comparison

This work used GPT-4o-mini as the primary LLM across all components. A systematic comparison of different LLMs would provide valuable insights for deployment decisions:

- **Commercial models**: GPT-4, GPT-4-turbo, Claude 3 (Opus, Sonnet, Haiku), Gemini Pro

- **Open-source models**: LLaMA 3, Mistral, Qwen, DeepSeek
- **Trade-off analysis**: Cost vs. accuracy vs. latency across different model sizes and providers
- **Component-specific optimization**: Different models may be optimal for different pipeline stages (e.g., smaller models for query generation, larger models for verdict synthesis)

Such comparisons would inform cost-effective deployment strategies and identify which components benefit most from more capable models.

3.3 Prompt Engineering and Management

The current system uses fixed prompts for each component. A more sophisticated prompt management approach could improve both performance and maintainability:

- **Prompt library**: Develop a versioned repository of prompts with documented performance characteristics, enabling systematic comparison and A/B testing
- **Prompt optimization**: Apply techniques such as automatic prompt optimization (APO) or DSPy to systematically improve prompts based on evaluation metrics
- **Few-shot example curation**: Identify optimal examples for few-shot prompting based on claim type, topic domain, and difficulty
- **Dynamic prompt selection**: Select prompts based on claim characteristics (scientific vs. political claims may benefit from different verification strategies)

3.4 Enhanced Source Reliability Assessment

The current reliability assessment relies primarily on the Media Bias/Fact Check (MBFC) dataset and domain heuristics. However, this approach has limitations when handling sources from well-established publishers that may still exhibit bias or inaccuracies on specific topics:

- **Publisher reputation nuances**: Established publishers (e.g., Frontiers, Nature subsidiary journals) may have varying quality standards across different publications or sections. A more granular assessment could consider journal impact factors, retraction rates, and topic-specific reliability
- **Claim-specific source evaluation**: A source reliable for general news may be less authoritative for specialized scientific claims; incorporating domain expertise signals would improve evidence quality

- **Temporal reliability**: Source reliability may change over time; incorporating historical accuracy tracking could improve assessments
- **Cross-referencing validation**: When multiple sources provide conflicting information, systematic cross-referencing protocols could help identify the most reliable interpretation

3.5 Production Deployment

Transitioning from research prototype to production system requires addressing several engineering challenges:

- **Cloud deployment**: Deploy the system on AWS, Google Cloud, or Azure with auto-scaling capabilities to handle variable load
- **User authentication and rate limiting**: Implement user accounts with usage quotas to manage API costs and prevent abuse
- **Caching and optimization**: Cache transcript extractions, reliability assessments, and common search results to reduce latency and costs
- **Monitoring and observability**: Implement comprehensive logging, error tracking, and performance monitoring for production operations
- **Public API**: Expose the fact-checking capabilities via a documented REST API for integration with third-party applications, browser extensions, or other platforms

Making the system publicly accessible would enable real-world validation and feedback collection, informing further improvements based on actual usage patterns.

3.6 Extended Evaluation

The current evaluation, while comparable in scale to prior fact-checking systems [7], would benefit from expansion along two dimensions:

- **Larger evaluation corpus**: Scaling from 30 to 100+ videos would enable more robust statistical analysis, identification of edge cases, and evaluation across a broader range of topics and content styles. A larger corpus would also allow stratified analysis by video characteristics (length, topic, claim density) and more reliable confidence intervals for reported metrics.

- **Parameter grid exploration**: The current evaluation used a single configuration (`max_claims`=5, `max_queries`=3, `max_results`=3). Systematic evaluation across the full parameter space would characterize the cost-accuracy-latency tradeoffs more comprehensively. Key configurations to explore include higher claim limits (10, 15, 20) for recall-focused applications, varied query counts to understand evidence saturation, and different result limits to assess diminishing returns in evidence collection.

Such extended evaluation would provide deployment guidelines for different use cases (e.g., high-precision quick checks vs. thorough investigations) and quantify the marginal value of additional computational resources.

A Ground Truth Annotation Dataset

This appendix provides details on the ground truth annotation dataset used for system evaluation. The complete dataset comprises 30 YouTube videos with 503 annotated claims across three thematic categories.

A.1 Video Corpus Summary

Table 22 presents the evaluation video corpus organized by category and content type.

Table 22: Evaluation video corpus by category and content type

Video Title	Category	Content Type	Claims
Fossil Fuels: The Greenest Energy	Climate	Misinformation	18
The Great Texas Freeze of 2021	Climate	Misinformation	15
Climate Change: What Do Scientists Say?	Climate	Misinformation	21
Is There Really a Climate Emergency?	Climate	Misinformation	19
Proof: Worldwide Massive Flooding is All Manmade	Climate	Misinformation	25
Causes and Effects of Climate Change (NatGeo)	Climate	Factual	12
Why Does Climate Change Matter	Climate	Factual	9
Extreme Weather	Climate	Factual	14
The Life Cycle of a Plastic Bottle	Climate	Factual	11

Continued on next page

Table 22 – continued from previous page

Video Title	Category	Content Type	Claims
What are Greenhouse Gases?	Climate	Factual	13
Fluoridated Water Lowers IQ (Harvard Study)	Health	Misinformation	16
Living with HIV: How Women Were Infected	Health	Misinformation	18
Garlic Cancer	Health	Misinformation	14
3 Detox Juices	Health	Misinformation	12
The Magical 3 Day Juice Fast	Health	Misinformation	15
What Happens When You Exercise Regularly	Health	Factual	17
The Brain-Changing Benefits of Exercise	Health	Factual	15
Immunology Wars: The Battle with HIV	Health	Factual	11
Juice vs. Whole Fruit: Which is Healthier?	Health	Factual	13
What Lack of Sleep Does to the Teenage Brain	Health	Factual	14
Egg Price Warning Comes True	Politics	Misinformation	19
Proof of Election Fraud in 2020	Politics	Misinformation	35
FBI Orchestrated Jan 6th	Politics	Misinformation	22
There is No Gender Wage Gap	Politics	Misinformation	17
A Nation of Immigrants	Politics	Misinformation	16
National Trust Sues Trump Admin	Politics	Factual	20
What Is Democracy (BBC)	Politics	Factual	9
What is Inflation?	Politics	Factual	10
UK Election Results Explained	Politics	Factual	18
Trump's Historic National Emergency	Politics	Factual	21

A.2 Annotation Schema

Each ground truth claim was annotated with the following attributes:

- **claim_text**: The verbatim or paraphrased factual assertion from the video transcript

- **is_checkable**: Boolean indicating whether the claim is verifiable (all included claims are checkable)
- **importance**: Numeric score (0.0–1.0) reflecting the claim's centrality to the video's main argument:
  - 0.85–1.0: Thesis-critical claims that would undermine the video's argument if proven false
  - 0.60–0.80: Important supporting evidence tied to the main thesis
  - 0.30–0.55: Contextual or background information
  - 0.0–0.25: Peripheral or anecdotal details
- **verdict.overall_stance**: Expected verification outcome (SUPPORTS, REFUTES, MIXED, UNCLEAR)
- **verdict.reasoning**: Brief justification for the assigned stance

A.3 Sample Annotation

A typical annotated video (e.g., "The Brain-Changing Benefits of Exercise" TED talk) contains the video identifier, title, and a list of claims with their attributes. Example claims include statements about neurotransmitter increases from exercise (importance: 0.9, verdict: SUPPORTS), neurogenesis in the hippocampus (importance: 0.95, verdict: MIXED due to human-animal research differences), and exercise recommendations (importance: 0.8, verdict: SUPPORTS based on CDC/WHO guidelines). The complete ground truth dataset in machine-readable YAML format is available in the project repository.

A.4 Annotation Guidelines

The annotation process followed these guidelines adapted from ClaimBuster's check-worthiness criteria [6]:

1. **Claim identification**: Include only factual, verifiable statements. Exclude opinions, predictions, and rhetorical questions.
2. **Importance scoring**: Apply the "thesis impact test"—if this claim were proven false, would the video's main argument collapse or weaken significantly?
3. **Verdict assignment**: Base verdicts on current scientific consensus and authoritative sources (peer-reviewed literature, government agencies, established fact-checkers).

4. **MIXED vs. UNCLEAR**: Use MIXED when evidence exists for both sides; use UNCLEAR when insufficient evidence exists to make a determination.

5. **Consistency**: Apply the same standards across all videos regardless of the video's overall stance or topic.

The complete ground truth dataset in machine-readable YAML format is available in the project repository at `experiments/data/ground_truth/`.

References

[1] Rami Aly, Zhijiang Guo, Michael Schlichtkrull, James Thorne, Andreas Vlachos, Christos Christodoulopoulos, Oana Cocarascu, and Arpit Mittal. Feverous: Fact extraction and verification over unstructured and structured information. *arXiv preprint arXiv:2106.05707*, 2021. URL https://arxiv.org/abs/2106.05707.

[2] Yifan Chen et al. Local: Logical and causal fact-checking with llm based multi-agents. *OpenReview*, 2024. URL https://openreview.net/pdf/9ddca198ed6f1db6d975787e879eced0a2b0d342.pdf.

[3] Luyu Gao, Zhuyun Dai, Panupong Pasupat, Anthony Chen, Arun Tejasvi Chaganty, Yicheng Fan, Vincent Y. Zhao, Ni Lao, Hongrae Lee, Da-Cheng Juan, et al. Rarr: Researching and revising what language models say, using language models. *arXiv preprint arXiv:2210.08726*, 2022. URL https://arxiv.org/pdf/2210.08726.pdf.

[4] Graphsignal. Llm api latency optimization explained, 2024. URL https://graphsignal.com/blog/llm-api-latency-optimization-explained/. Accessed: December 2025.

[5] Naeemul Hassan, Bill Adair, James T. Hamilton, Chengkai Li, Mark Tremayne, Jun Yang, and Cong Yu. Detecting check-worthy factual claims in presidential debates. In *Proceedings of the 24th ACM International Conference on Information and Knowledge Management (CIKM)*, pages 1835–1838, 2015. URL https://ranger.uta.edu/~cli/pubs/2015/claimbuster-cikm15-hassan.pdf.

[6] Naeemul Hassan, Gensheng Zhang, Fatma Arslan, Josue Caber, Damian Muthukrishnan, Baoyu Shu, Junghoo Kim, Chengkai Li, and Mark Tremayne. Toward automated fact-checking: Detecting check-worthy factual claims by claimbuster. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 1803–1812. ACM, 2017. doi: 10.1145/3097983.3098131. Industry standard for claim detection evaluation metrics including Precision@k, Recall@k, and MAP.

[7] Naeemul Hassan, Gensheng Zhang, Fatma Arslan, Josue Caber, Damian Muthukrishnan, Baoyu Shu, Junghoo Kim, Chengkai Li, and Mark Tremayne. Claimbuster: The first-ever end-to-end fact-checking system. *Proceedings of the VLDB Endowment*, 10(12):1945–1948, 2017. Evaluated on 25 presidential debates from the 2016 U.S. election cycle.

[8] Alan R. Hevner, Salvatore T. March, Jinsoo Park, and Sudha Ram. Design science in information systems research. *MIS Quarterly*, 28(1):75–105, 2004. URL https://wise.vub.ac.be/sites/default/files/thesis_info/design_science.pdf.

[9] Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, et al. Retrieval-augmented generation for knowledge-intensive nlp tasks. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 33, pages 9459–9474, 2020. URL https://nlp.cs.ucl.ac.uk/publications/2020-05-retrieval-augmented-generation-for-knowledge-intensive-nlp-tasks/.

[10] Chengkai Li et al. A platform for live and on-demand monitoring of public discourse. *Proceedings of the VLDB Endowment*, 10(12):1945–1948, 2017. URL https://vldb.org/pvldb/vol10/p1945-li.pdf.

[11] Potsawee Manakul, Adian Liusie, and Mark J. F. Gales. Selfcheckgpt: Zero-resource black-box hallucination detection for generative large language models. *arXiv preprint arXiv:2303.08896*, 2023. URL https://arxiv.org/abs/2303.08896.

[12] Media Bias/Fact Check. Methodology - media bias/fact check, 2024. URL https://mediabiasfactcheck.com/methodology/. Accessed: December 2025.

[13] Briony J. Oates. *Researching Information Systems and Computing*. SAGE Publications, 2006.

[14] OpenAI. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*, 2024. URL https://arxiv.org/pdf/2303.08774.pdf.

[15] OpenAI. Latency optimization - openai platform documentation, 2024. URL https://platform.openai.com/docs/guides/latency-optimization. Accessed: December 2025.

[16] Sebastian Raschka. Understanding the 4 main approaches to llm evaluation (from scratch). *Ahead of AI*, 2025. URL https://magazine.sebastianraschka.com/p/llm-evaluation-4-approaches.

[17] Sebastian Ruder. The evolving landscape of llm evaluation. *NLP Newsletter*, 2025. URL https://www.ruder.io/the-evolving-landscape-of-llm-evaluation/.

[18] James Thorne, Andreas Vlachos, Christos Christodoulopoulos, and Arpit Mittal. Fever: A large-scale dataset for fact extraction and verification. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL-HLT)*, pages 809–819, 2018. URL https://aclanthology.org/N18-1074/.

[19] Qingyun Wu, Gagan Bansal, Jieyu Zhang, Yiran Wu, Shaokun Zhang, Erkang Zhu, Beibin Li, Li Jiang, Xiaoyun Zhang, and Chi Wang. Autogen: Enabling next-gen llm applications via multi-agent conversation. *arXiv preprint arXiv:2308.08155*, 2023. URL https://arxiv.org/abs/2308.08155.

[20] Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models. *arXiv preprint arXiv:2210.03629*, 2023. URL https://arxiv.org/abs/2210.03629.

[21] Xuan Zhang, Wei Wei, Yuxiao Wen, Yang Shi, and Bowen Zou. Factagent: Towards agentic multi-hop fact-checking via large language models. *arXiv preprint arXiv:2506.17878*, 2025. URL https://arxiv.org/abs/2506.17878. Available at: https://github.com/HySonLab/FactAgent.