VerbaTask: Multi-Agent AI for Meeting Action Tracking

VerbaTask turns raw meeting transcripts into structured, tracked action items automatically. It was built on Elasticsearch Agent Builder and runs a pipeline of four specialized agents at the moment a transcript is uploaded. It extracts the tasks, summarizes discussions, validates deadlines and answers questions across all your meetings.

The Problem

Most meetings end with notes that are hard to keep track of. Project management tools are great for tracking tasks, but someone still has to manually create every ticket, assign owners, and set deadlines after the meeting. That process is slow, inconsistent, and unnecessarily manual. In a world where AI can automate complex workflows, manually transcribing action items from meetings is a problem worth solving. That's where VerbaTask comes in.

The Solution

Paste or upload any meeting transcript. VerbaTask's agent pipeline automatically:

- Extracts every action item with owner, due date, and risk level
- Resolves relative dates like "tomorrow" or "next Friday" into absolute dates
- Summarizes the meeting with key decisions and roadblocks.
- Validates completeness and flags overdue items
- Stores everything in Elasticsearch making the data ready to query, track, and act on

Demo:

Process a transcript -> agents run automatically -> dashboard updates in real time

1. Paste or upload a .txt meeting transcript
2. Hit **Run All Agents**. The full pipeline executes sequentially
3. Switch to the **Dashboard** to see extracted tasks with owners, due dates, and risk levels

4. Mark tasks complete, update due dates, or change risk levels. All these values are autosaved to Elasticsearch indexes.
5. Click Revalidate to refresh overdue statuses and get an updated validation report
6. Ask Insight questions like “Who has the most overdue tasks?” or “Which meeting had the most action items?”

Architecture (check out verbatask_architecture.drawio.png in the docs folder)

Components:

Indexes:

- **meeting_messages:** stores every parsed message from the transcript (meeting ID, speaker and text)
- **action_items:** stores action items extracted by the agent (meeting ID, task, owner, team, due_date, status, risk level, created at)
- **meeting_summaries:** stores the meeting summary generated by the Meeting Summary Agent
- **meeting_validations:** stores the validation report generated by the Overdue Item Validator Agent.

Tools:

VerbaTask runs a FASTMCP server (tools.py) that exposes two tools to the Kibana agents. The server runs locally and is exposed to Kibana via ngrok.

- **create_action_item:** Called by the Action Item Extraction Agent. Creates a new action item in the action_items index.
- **update_action_item:** Called by the Overdue Item Validator Agent. Updates the status of an existing action item.
- **search_meeting_messages:** Called by the Action Item Extraction Agent and the Meeting Summary Agent. Fetches all the transcript messages for a given meeting ID from the meeting_messages index. It is defined directly in the ElasticSearch platform.
- **Elasticsearch's platform.core.search tool:** Called by the Insights Agent. Answers natural language questions across all meetings using ES|QL.

Agents:

VerbaTask uses four agents hosted on Elasticsearch Agent Builder Platform. They run sequentially every time a transcript is processed. Instructions for creating each agent in Kibana can be found in the agents/ folder.

1. Action Item Extraction Agent

The agent calls the search_meeting_messages tool to fetch all transcript messages from the meeting_messages index for the given meeting_id.

It then reads through those messages to identify every action item and calls create_action_item for each one it finds.

For each one it calls the create_action_item MCP tool to write a structured document to the action_items index with:

- Task description
- Owner
- Due date (relative dates like “tomorrow” resolved to absolute YYYY-MM-DD)
- Risk level
- Status (default: Open)
- Unique action_id
- Team
- Created at timestamp

2. Meeting Summary Agent

Generates a concise summary of the meeting covering key decisions, blockers, and outcomes. Saved to the meeting_summaries index and displayed on the dashboard per meeting.

3. Overdue Item Validator Agent

Reviews all action items for the meeting and flags any action items where due date has strictly passed as Overdue using the update_action_item MCP tool. Saves a full validation report to meeting_validations index. Can be re-run on demand using the revalidate button on the dashboard.

4. Insights Agent

Answers natural language questions across all meetings using ES|QL and semantic search via Elasticsearch's platform.core.search tool. Examples:

- "Who has the most overdue tasks?"
- "Which high-risk items are still open?"
- "Summarize what happened across all meetings"

Dashboard Features

- **Metrics:** Displays the counts of Meetings, Open, Overdue, Completed, and Completion Rate
- **Meeting selector**: Allows us to switch between meetings with a progress bar showing task completion
- **Task list**: Displays every action item with owner, due date, risk, and status
- **Date and Risk Level pickers:** Allows us to update due dates and risk levels directly from the dashboard, auto-saved to Elasticsearch
- **Completion checkboxes**: Allows us to mark tasks done with strikethrough styling
- **Revalidate Button:** Regenerates the validation report and updates the status.
- **Meeting Summary**: Displays a collapsible summary from the Meeting Summary Agent
- **Validation Report**: Displays a collapsible report from the Overdue Item Validator Agent

Tech Stack

Agent Orchestration: Elasticsearch Agent Builder

MCP server: Python+ FastMCP

Backend: Python

Frontend: Streamlit

LLM: Most likely Anthropic Claude Sonnet 4.5. Depends upon the Agent Setup.

Local Tunnel: ngrok

Project Structure

```
verbatask/
|–––– src/
|     |---- app.py                   (streamlit frontend)
|     |----  transcript_parser.py      (Parses transcripts)
|     |----  transcript_ingest.py      (Indexes messages into Elasticsearch)
|     |----  extract_action_items.py   (Calls Action Item Extraction Agent)
|     |---- meeting_summary_agent.py  (Calls Meeting  Summary Agent)
|     |----  validation_agent.py          (Calls Overdue Item Validator Agent)
|     |----  insights_agent.py        (Calls Insights Agent)
|     |---- tools.py                     (MCP tools(FastMCP server)
|     |----  setup_indices.py            (Creates all ES indices on first run)
| ---- sample_data                  (contains the synthetic data that can be used to test the pipeline)
|---- raw_data                       (stores the uploaded raw transcript here(.txt))
|----parsed_data                    (stores the parsed transcript here (.json))
|---- .env                       (Environment variables- needs to be added before running)
| ---- requirements.txt  (Install before running)
|---- README.md
```

Setup

1. Clone the repo

```
git clone https://github.com/mahitayadla/Verbatask.git
cd verbatask
```

2. Install dependencies

```
pip install -r requirements.txt
```

3. Configure environment variables

Fill in your own credentials in the .env file placeholders before running the app.

4. Create Elasticsearch indices

```
python setup_indices.py
```

This will create all the indexes in ES.

5. **Create Agents**

Create the four agents in ElasticSearch Agent Builder using the custom instructions present in agents/ folder.

6.Create search_meeting_messages tool

Create search_meeting_messages tool in ElasticSearch Agent Builder using the custom instructions present in tools/ folder.

7. Install ngrok and expose the MCP server

Download and install ngrok (or any other local tunnel service) from https://ngrok.com

Then run: python src/tools.py

In another terminal expose it publicly:

```
ngrok http 8000
```

Copy the ngrok URL and add it as the MCP server URL in your Kibana Agent Builder tool settings.

Ngrok_url/mcp

(ex: https://252e-2600-4040-b62f-1400-xxxx-xxxx-d6d-cca6.ngrok-free.app/mcp)

8. Run the app in another terminal

```
streamlit run src/app.py
```

Sample Data

To test the pipeline without needing a real meeting, synthetic transcripts were generated using LLMs (CHATGPT and Claude) and the following prompt:

“Write a realistic workplace meeting transcript in WebVTT format with 8 discussing [topic]. Keep the conversation natural, the way a real meeting would flow. Action items should come up organically through the dialogue, not labeled or called out explicitly. Medium length, around 2-3 minutes.”

10 sample transcripts covering different meeting topics are available in the ‘sample_data/’ folder and can be used to test the full pipeline.

Transcript Format

VerbaTask supports WEBVTT format. Common format used by meeting platforms like Zoom. (Can use any format by making changes to transcript_parser.py)

```
WEBVTT

1
00:00:01.000 --> 00:00:03.000
Alice: We need to finish the report by Friday.

2
00:00:04.000 --> 00:00:06.000
Bob: I’ll have it ready by Thursday morning.
```

Plain text in Speaker: message format also works.

How it works:

Transcript Parsing & Ingestion

When a user pastes or uploads a transcript and clicks **Process Transcript**, the app first runs **transcript_parser.py**.

The parser reads the **WEBVTT** text line by line and uses a regex pattern to extract the speaker and message text. Instead of keeping it as raw text, it converts it to a structured message object with speaker and text fields.

These messages are passed into **transcript_ingest.py** which indexes all of them into the **meeting_messages** ES index.

Every transcript is grouped using a meeting_id generated from the current timestamp (formatted like M_YYYYMMDD_HHMMSS). That way, each meeting stays isolated and easy to query later.

Action Item Extraction Agent

Once the transcript is indexed, **app.py** calls **extract_action_items.py**.

This sends a POST request to the Kibana Agent Builder and includes the meeting_id and today's date in the prompt. Including today's date allows the agent to convert relative deadlines like "tomorrow" or "next Friday" into real YYYY-MM-DD dates.

The **Action Item Extraction Agent** first fetches all transcript messages using the custom **search_meeting_messages** tool. It reads through the full conversation and identifies any commitments or tasks.

For every action item it finds, it calls the custom **create_action_item** MCP tool (defined in tools.py). That tool creates a document in the **action_items** index with fields like:

- action_id
- meeting_id
- task
- owner
- team

- due_date
- risk_level
- status (defaulted to Open)

Meeting Summarization

The **meeting_summary_agent.py** script calls a separate custom Meeting Summary Agent using the same meeting_id and the **search_meeting_messages** tool.

The agent pulls the full transcript from Elasticsearch and generates a structured summary covering:

- Key decisions
- Main discussion points
- Blockers

The summary is saved in the **meeting_summaries** index using the meeting_id as the document ID.

Validation

The **validation_agent.py** script checks for overdue items.

It sends today's date in the prompt and explicitly instructs the agent to mark items as **Overdue only if the due date is strictly before today**. The agent fetches all open items from the **action_items** index, compares their due dates, and calls custom **update_action_item** MCP tool for anything past due.

update_action_item uses Elasticsearch's update_by_query to locate the item by action_id and change its status.

The final validation report is saved to the meeting_validations index so it can be viewed later.

Dashboard:

The dashboard in app.py fetches all action items using get_all_action_items() and displays them in a layout showing:

- Done checkbox
- Task

- Owner
- Due Date
- Risk
- Status

Each row updates in real time.

When a checkbox is toggled, update_status() immediately writes the new status (Completed or Open) to Elasticsearch and triggers st.rerun() so the UI refreshes instantly.

Risk level and due date edits use inline widgets that auto-save as soon as they're changed.

Overdue Status Management

The Revalidate Meeting button does two things:

1. It runs a direct update_by_query from Python to mark any open items with due dates strictly before today as Overdue.
2. It re-runs the Validation Agent to generate a fresh validation report reflecting the current state of all tasks.

The new report replaces the previous one in the meeting_validations index, so there's always a single up-to-date validation summary per meeting.

Metrics & Progress

The metrics row at the top calculates totals in real time:

- Meetings
- Open
- Overdue
- Completed
- Completion Rate

The progress bar is specific to the selected meeting and is calculated from that meeting's filtered task list.

Insights

The **Ask Insights** page sends the user's natural language question to the Insights Agent.

The agent uses Elasticsearch's native platform.core.search tool to run ES|QL or semantic queries across all indices and returns a structured response.

Common questions are available as buttons that auto-fill the input field.

To avoid unnecessary re-queries, answers are stored in st.session_state, so they persist across reruns.

References:

- **Elastic AI Agent Builder — Context Engineering Introduction**
  https://www.elastic.co/search-labs/blog/elastic-ai-agent-builder-context-engineering-introduction
  (Explains Elastic's context engineering principles)
- **Elastic Agent Builder Documentation**
  https://www.elastic.co/docs/explore-analyze/ai-features/elastic-agent-builder
  (Official docs detailing how to build, configure, and integrate AI agents with Elastic tools and data.)
- **ChatGPT (OpenAI)**
  Used to generate synthetic meeting transcripts.
- **Claude (Anthropic)**
  Used to generate additional synthetic transcripts.
- **FastMCP Documentation**
  (Documentation and examples for defining and structuring MCP tools used by agents.)
- **Streamlit Documentation**
  https://docs.streamlit.io
  (Documentation referencing UI components and layout patterns used in the dashboard.)
- **ngrok Documentation**
  https://ngrok.com/docs
  (Referenced for securely exposing local development servers for agent-to-tool communication.)

- **Update by query API — Elasticsearch Documentation**
  https://www.elastic.co/docs/reference/elasticsearch/rest-apis/update-by-query-api
  (Official reference for the update_by_query API used in tools.py to update action item statuses)