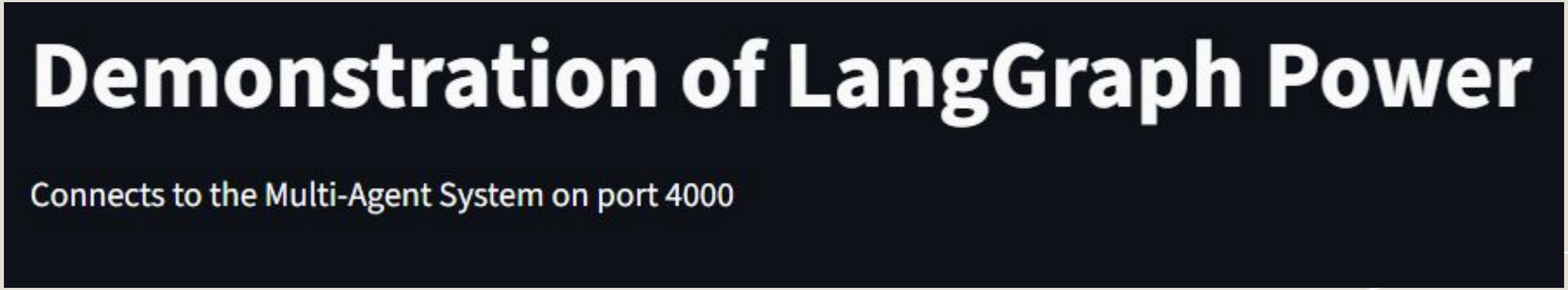
Agentic Email Automation … with LangGraph

Presented by: Gaurav Pandey, Nick Chandler, Anirudh Sarda, Navnish Pandey, Yann L'Hotelier

Contents

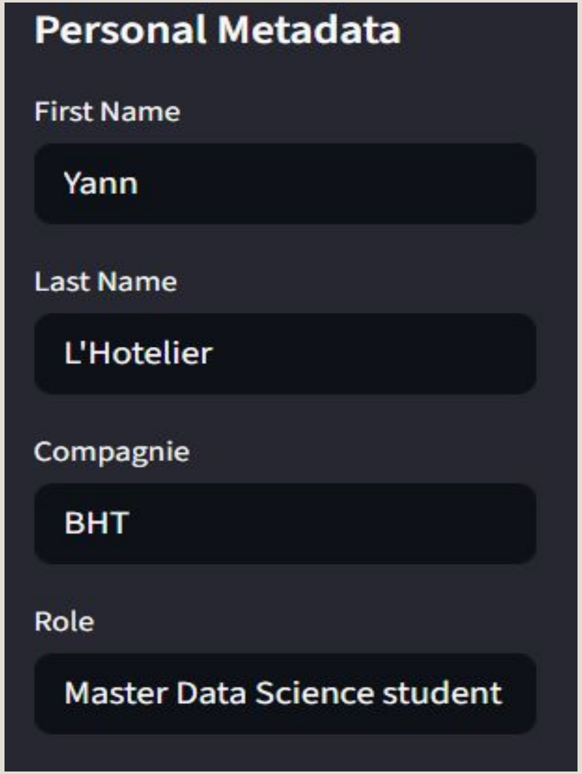
Demo!



Demonstration of LangGraph Power

Connects to the Multi-Agent System on port 4000

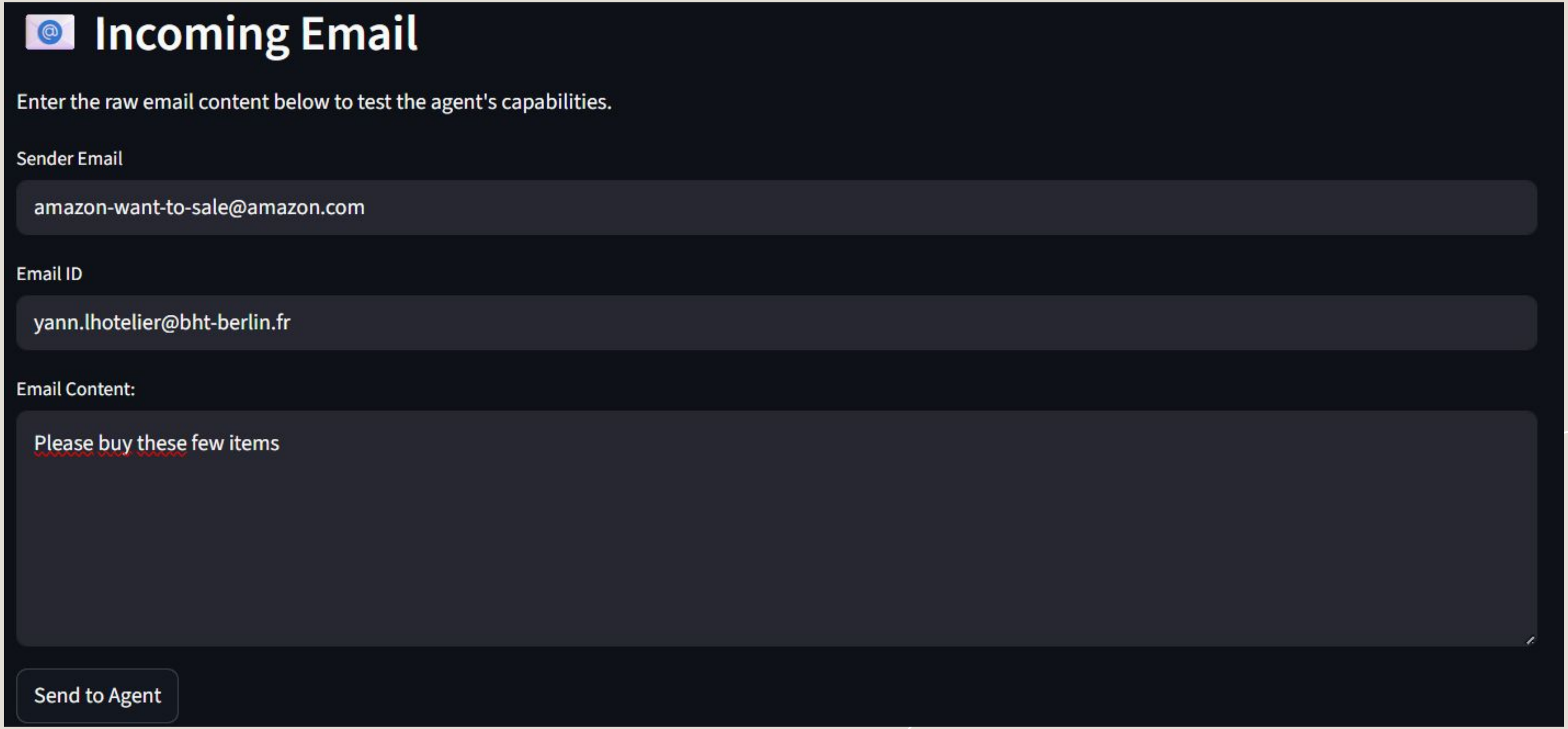
Enter your data



Personal Metadata

First Name	Yann
Last Name	L'Hotelier
Compagnie	BHT
Role	Master Data Science student

Give an email to review



Incoming Email

Enter the raw email content below to test the agent's capabilities.

Sender Email

amazon-want-to-sale@amazon.com

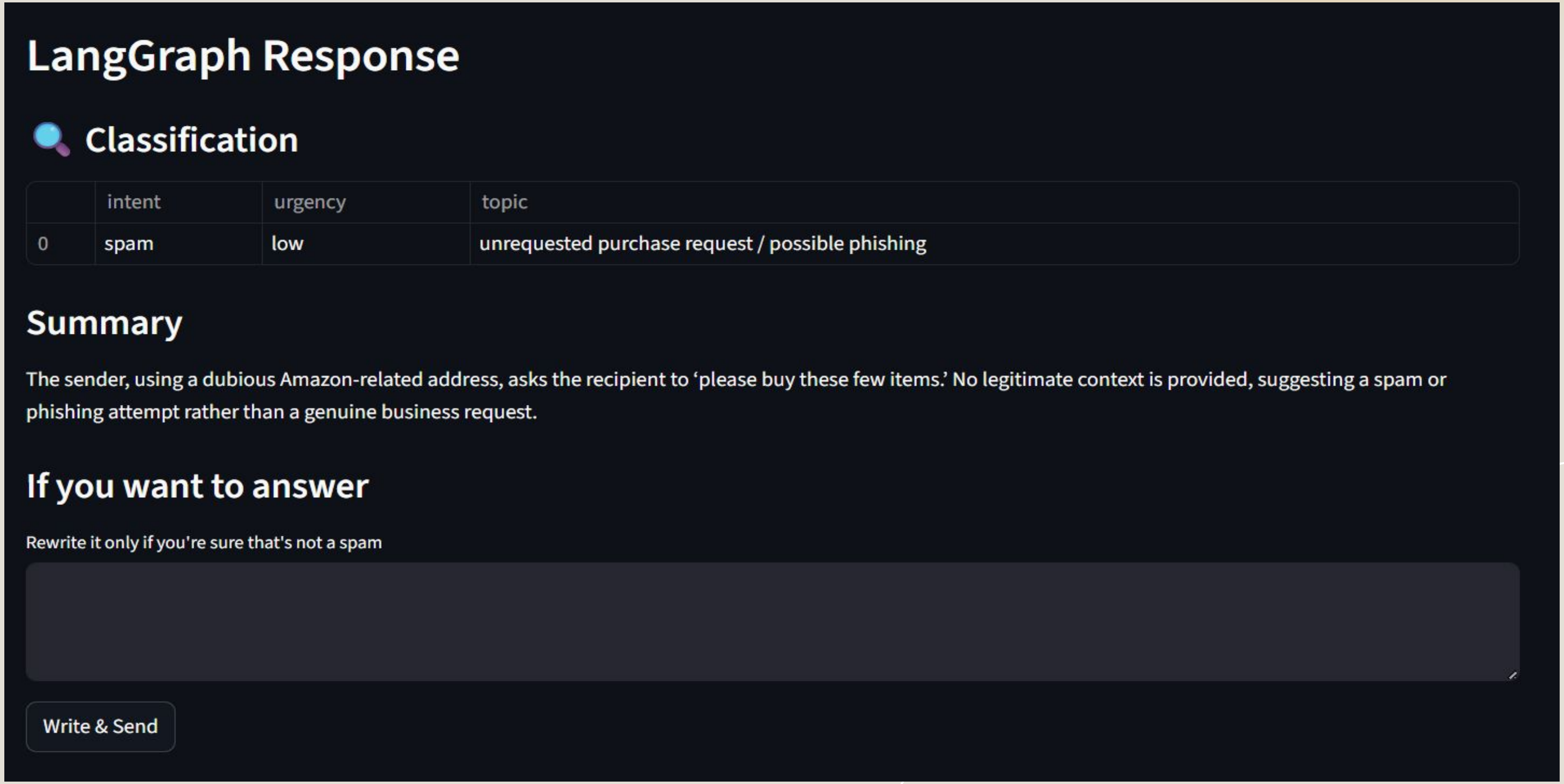
Email ID

yann.lhotelier@bht-berlin.fr

Email Content:

Please buy these few items

Examine results



LangGraph Response

🔍 Classification

	intent	urgency	topic
0	spam	low	unrequested purchase request / possible phishing

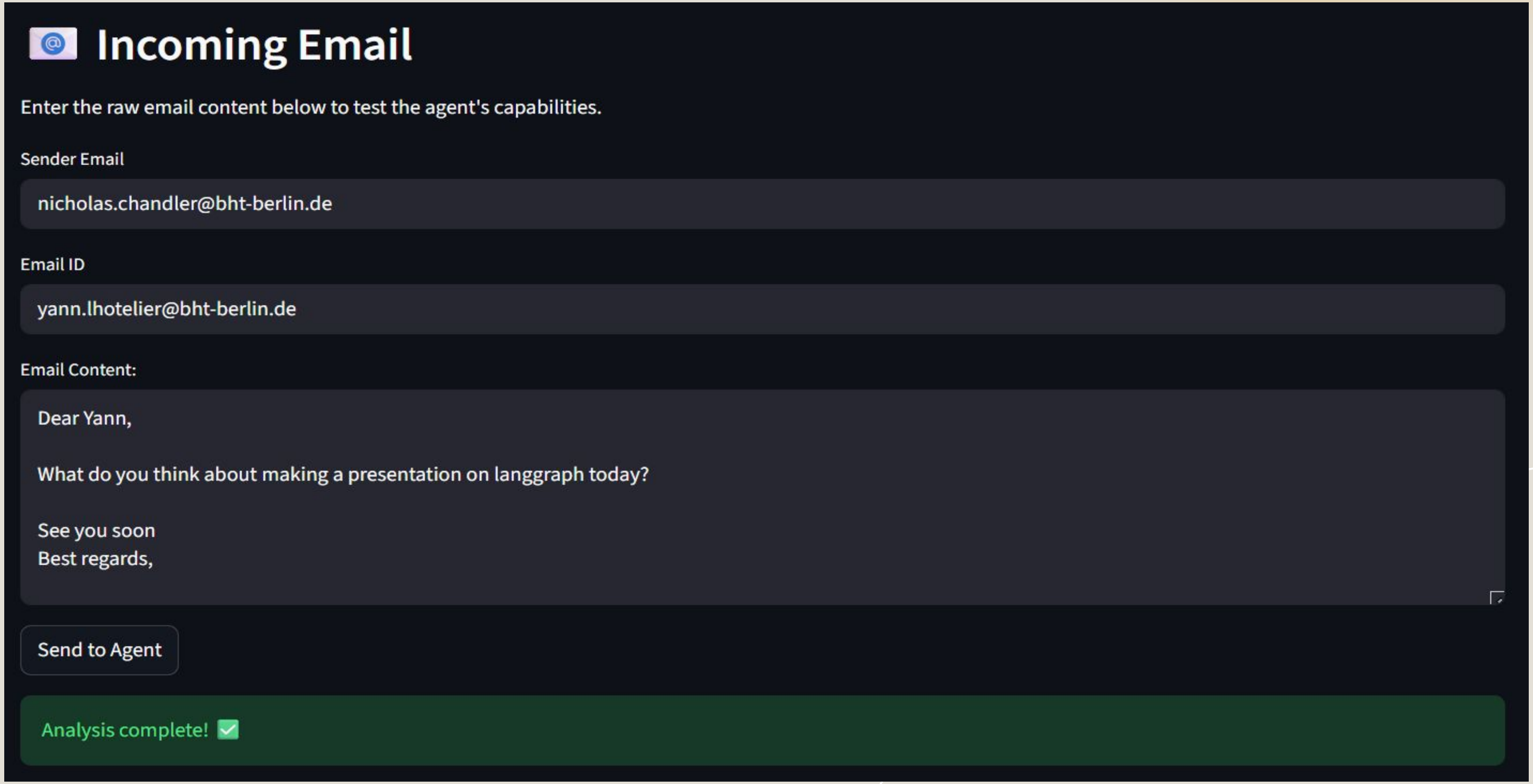
Summary

The sender, using a dubious Amazon-related address, asks the recipient to ‘please buy these few items.’ No legitimate context is provided, suggesting a spam or phishing attempt rather than a genuine business request.

If you want to answer

Rewrite it only if you're sure that's not a spam

With non-spam email



Incoming Email

Enter the raw email content below to test the agent's capabilities.

Sender Email

nicholas.chandler@bht-berlin.de

Email ID

yann.lhotelier@bht-berlin.de

Email Content:

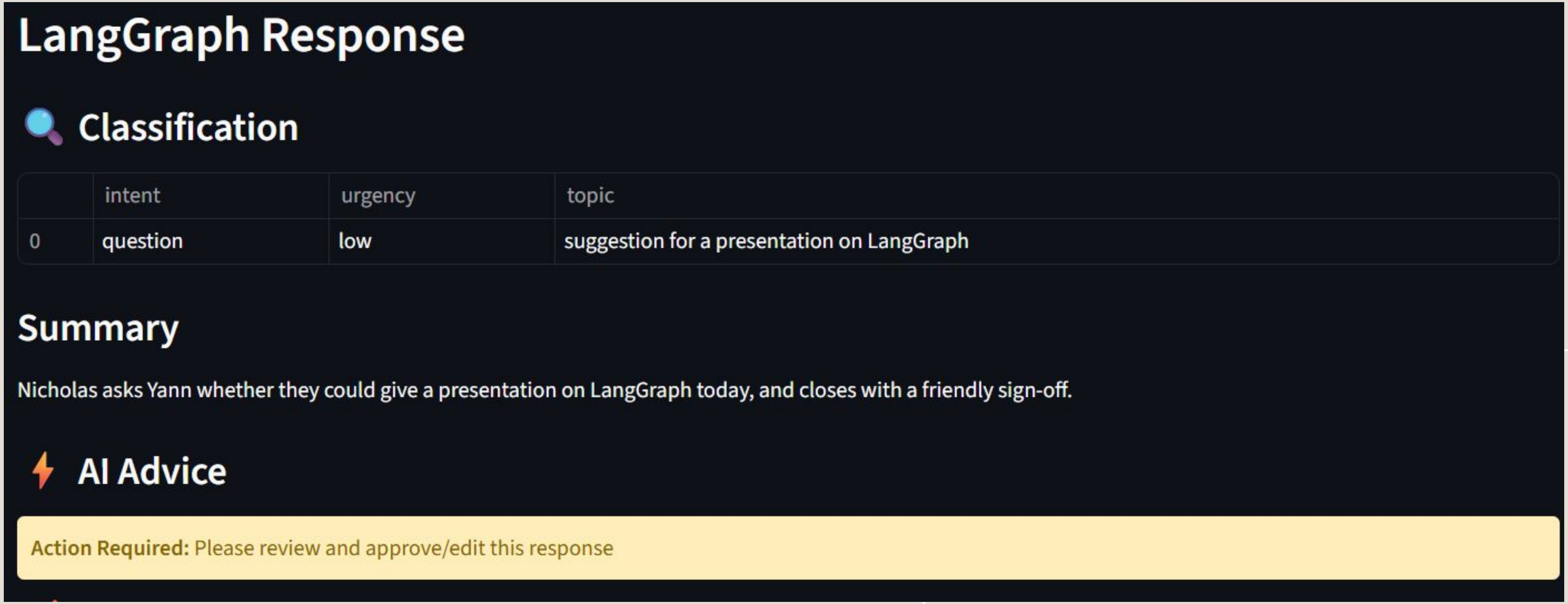
Dear Yann,

What do you think about making a presentation on langgraph today?

See you soon
Best regards,

Analysis complete! ✅

With non-spam email



LangGraph Response

🔍 Classification

	intent	urgency	topic
0	question	low	suggestion for a presentation on LangGraph

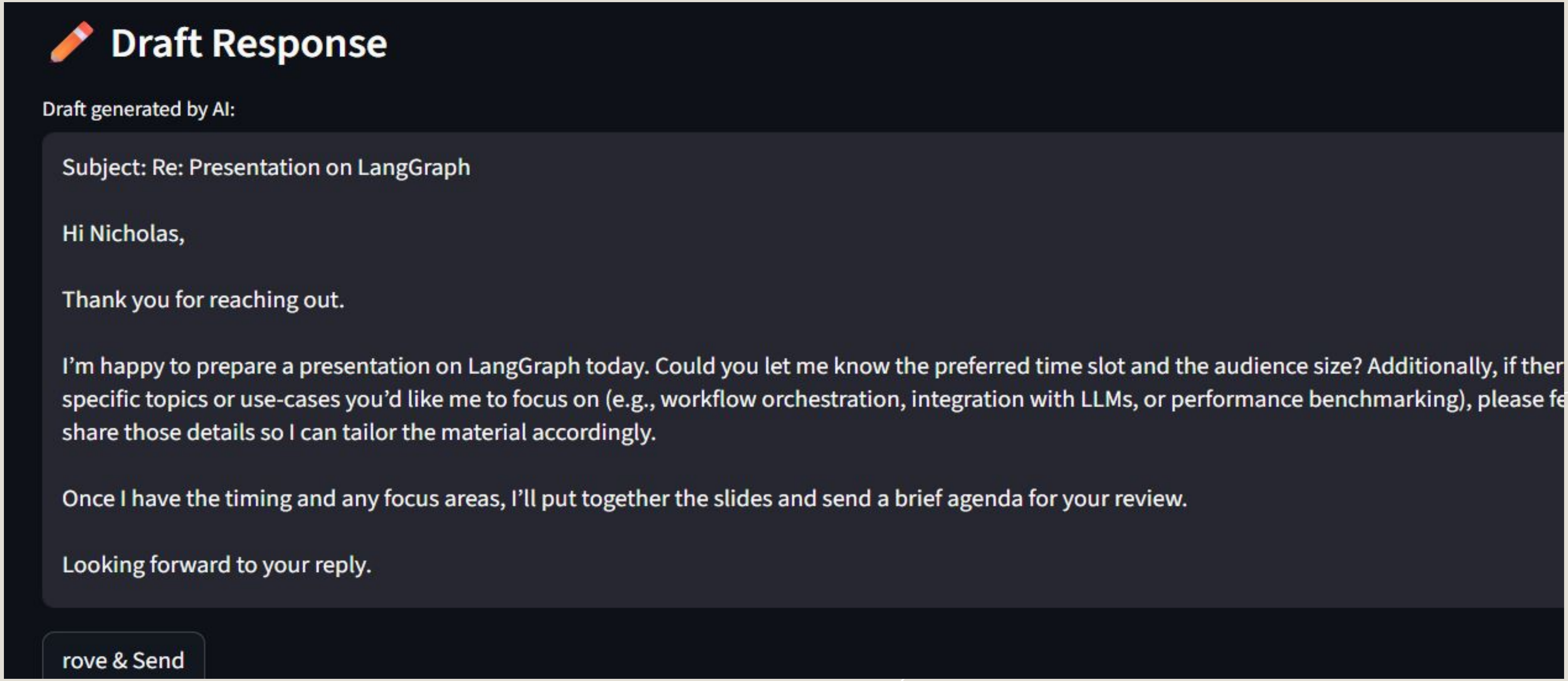
Summary

Nicholas asks Yann whether they could give a presentation on LangGraph today, and closes with a friendly sign-off.

⚡ AI Advice

Action Required: Please review and approve/edit this response

With non-spam email



Draft Response

Draft generated by AI:

Subject: Re: Presentation on LangGraph

Hi Nicholas,

Thank you for reaching out.

I'm happy to prepare a presentation on LangGraph today. Could you let me know the preferred time slot and the audience size? Additionally, if ther
specific topics or use-cases you'd like me to focus on (e.g., workflow orchestration, integration with LLMs, or performance benchmarking), please fe
share those details so I can tailor the material accordingly.

Once I have the timing and any focus areas, I'll put together the slides and send a brief agenda for your review.

Looking forward to your reply.

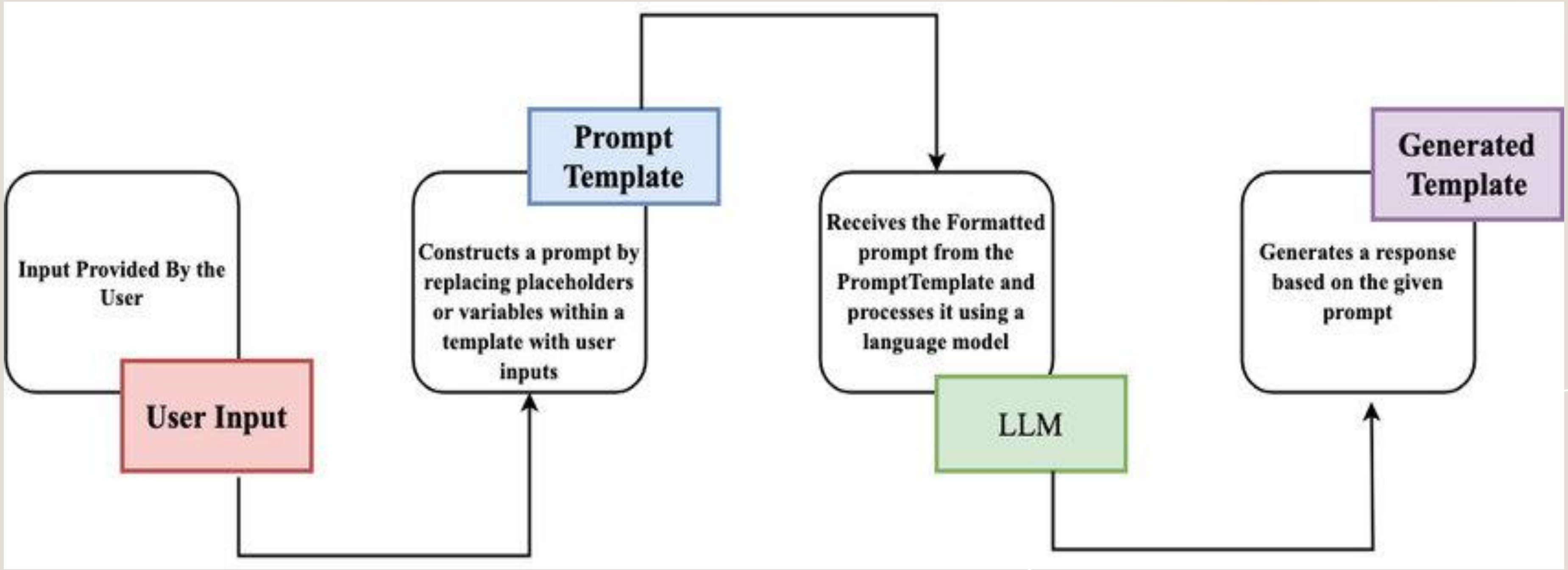
LangChain & LangGraph

The Foundations of Our System

What is LangChain?

- Framework for building LLM-powered applications
- Connects prompts, tools, models, and memory
- Ideal for linear and sequential workflows
- Limited control over complex, branching logic

LangChain Workflow


Input Provided By the User
User Input
Prompt Template
Constructs a prompt by replacing placeholders or variables within a template with user inputs
Receives the Formatted prompt from the PromptTemplate and processes it using a language model
LLM
Generated Template
Generates a response based on the given prompt

What is LangGraph?

- Built on top of LangChain
- Represents workflows as explicit graphs
- Nodes = well-defined actions
- Edges = controlled execution flow
- Enables stateful, production-ready agents

Why LangGraph for Our System?

Email processing is non-linear:

- Spam vs Human Review vs Machine Processing
- Priority-based routing
- Feedback and retry loops

LangGraph enables:

- Explicit control over decision paths
- Safe, debuggable automation
- Multi-agent and human-in-the-loop workflows
- Production-grade reliability

Thinking in LangGraph

Step 1 : Start from the real process (not the LLM)

LangGraph agents are designed by modeling real workflows, not open-ended LLM reasoning loops.

- Identify the business process first

- Important question is “What actually happens step by step?”

Example (email automation):
Read -> Classify -> Route -> Draft -> Review -> Send

Step 2 : Break the workflow into discrete nodes

Each step in the process becomes a node.

Node principles:

- One node = one responsibility
- Nodes are just Python functions
- Nodes can make routing decisions

Examples:

- ➢ Read Email, Classify Intent, Search Documentation, Send Reply

Node Types

Different kinds of work need different nodes. LangGraph distinguishes what kind of work each node does

- ➢ LLM nodes -> reasoning, classification, text generation
- ➢ Data nodes -> database or document retrieval
- ➢ Action nodes -> sending emails, creating tickets
- ➢ Human nodes -> approval, edits, escalation

Step 3 : Design the Workflow and Decision Paths

LangGraph uses graphs, not hidden decision-making.

- Nodes declare where they can go next
- Transitions are designed upfront
- No “LLM decides everything” black box

❑ This makes execution safer, easier to debug and production-ready

Step 4 : Design State

State is shared memory, not chat history. All nodes read and write to a shared state object.

State design rules:

- Store raw data only
- No formatted prompts
- No generated explanations

❏ State includes: Original inputs (emails, IDs), Search results, Draft responses

Step 5 : Handle errors

Different errors are handled differently:

- Transient errors -> automatic retries
- LLM-recoverable errors -> loop back with context
- User-fixable errors -> pause for input
- Unexpected errors -> bubble up for debugging

Human Review Is Built into the Workflow

Human input is built into the graph, not bolted on.

- interrupt() pauses execution
- State is checkpointed
- Workflow resumes exactly where it stopped

❏ Used when high urgency, complex issues, quality or compliance matters

Step 6 : Wire it together

Only essential edges are defined.

- Routing happens inside nodes
- Nodes return both State updates and next destination

❏ This keeps the graph simple and predictable.

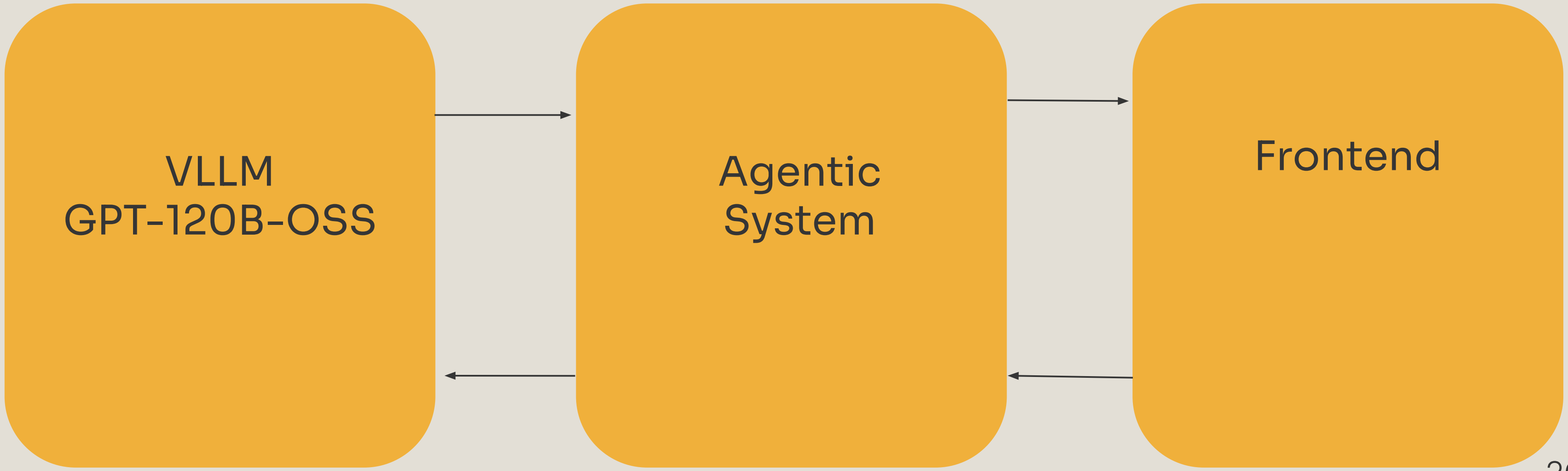
How LangGraph Differs from Other Agent Models

	Classic Agents	LangGraph Agents
State handling	Information embedded in prompts	Shared structured state object
Control flow	LLM decides next step dynamically	Flow explicitly defined in a graph
Human involvement	Added manually if needed	Built-in human review steps
Planning	Implicit inside the model	Designed ahead of time
Debugging	Hard to trace decisions	Easy, step-by-step tracing

How we built our demo

Nick & Yann

System Architecture



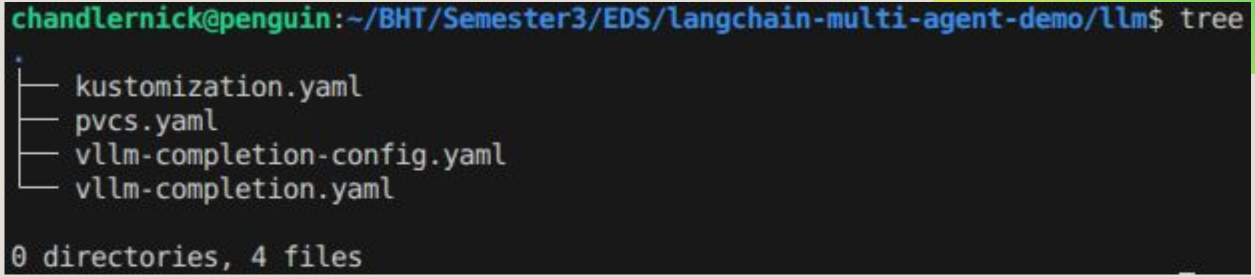
Connections Summary

- The VLLM component runs on an A100 80GB
- VLLM served on port 8000 to communicate with the agentic system
- Agentic system run via uvicorn on port 4000 to talk to front end
- Front end runs as a Streamlit service

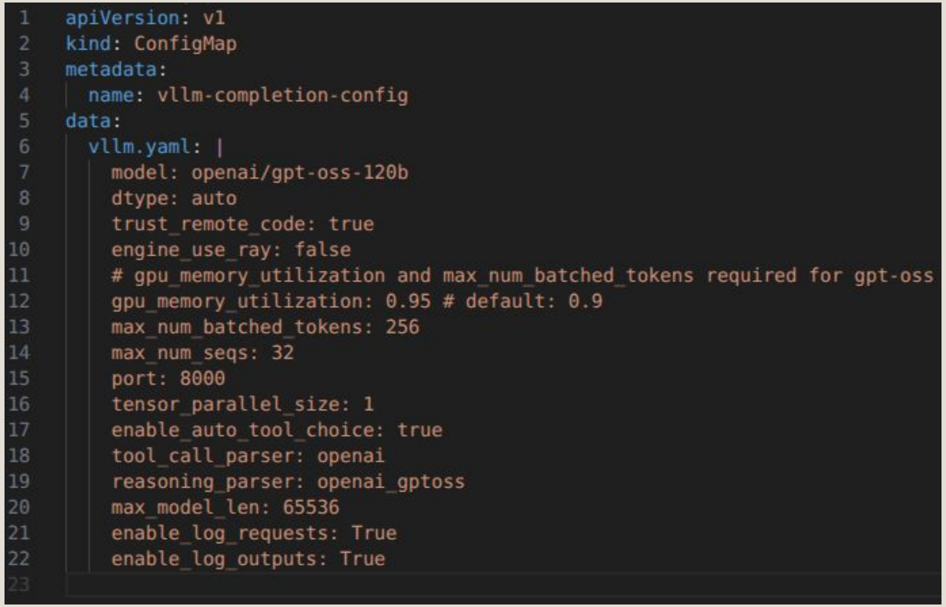
VLLM

- Optimized LLM Serving
- Memory-Efficient KV-Cache
- Paged Attention
- Open-Source
- Runs with a simple ‘kubectl apply -f vllm.yaml’ command

VLLM



```
chandlernick@penguin:~/BHT/Semester3/EDS/langchain-multi-agent-demo/llm$ tree
.
├── kustomization.yaml
├── pvcs.yaml
├── vllm-completion-config.yaml
└── vllm-completion.yaml

0 directories, 4 files
```



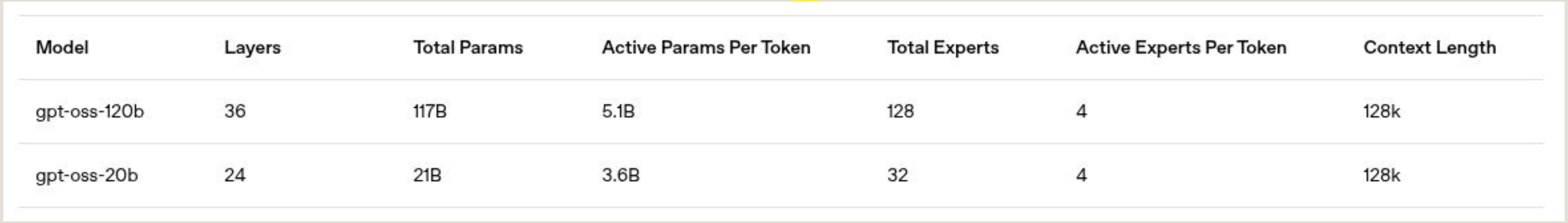
```
apiVersion: v1
kind: ConfigMap
metadata:
  name: vllm-completion-config
data:
  vllm.yaml: |
    model: openai/gpt-oss-120b
    dtype: auto
    trust_remote_code: true
    engine_use_ray: false
    # gpu_memory_utilization and max_num_batched_tokens required for gpt-oss
    gpu_memory_utilization: 0.95 # default: 0.9
    max_num_batched_tokens: 256
    max_num_seqs: 32
    port: 8000
    tensor_parallel_size: 1
    enable_auto_tool_choice: true
    tool_call_parser: openai
    reasoning_parser: openai_gptoss
    max_model_len: 65536
    enable_log_requests: True
    enable_log_outputs: True

```

- This directory contains the infrastructure for the system.
- The config is given here with parameters specifying aspects about the LLM.
- To get an LLM like this, you need only to deploy these yaml files.
- This was from digi-llm.

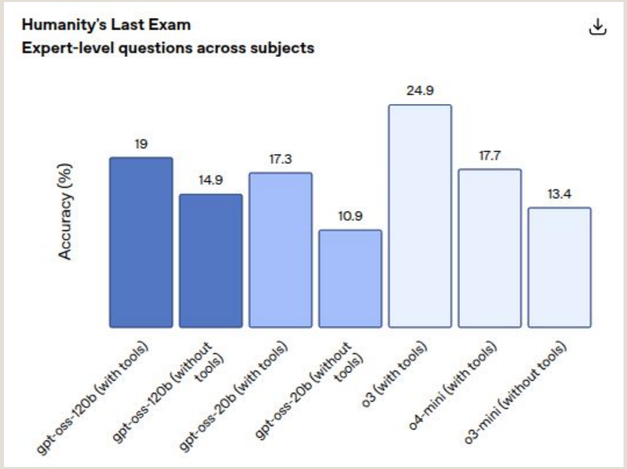
GPT-OSS-120B

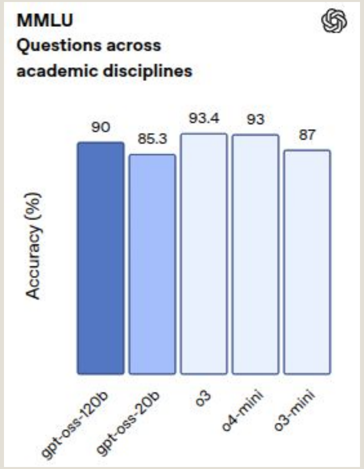
- ~120B Parameters (Quantized)
- Hybrid Attention
- Open-Weight Reasoning Model from OpenAI
- Works for multiple languages: German, French, Hindi, English

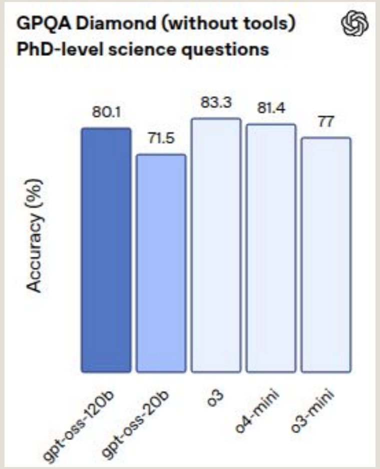
GPT-OSS-120B

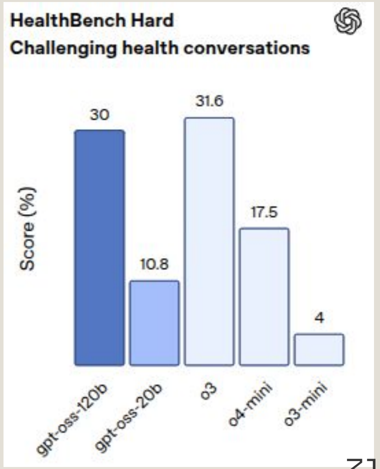


Model	Layers	Total Params	Active Params Per Token	Total Experts	Active Experts Per Token	Context Length
gpt-oss-120b	36	117B	5.1B	128	4	128k
gpt-oss-20b	24	21B	3.6B	32	4	128k



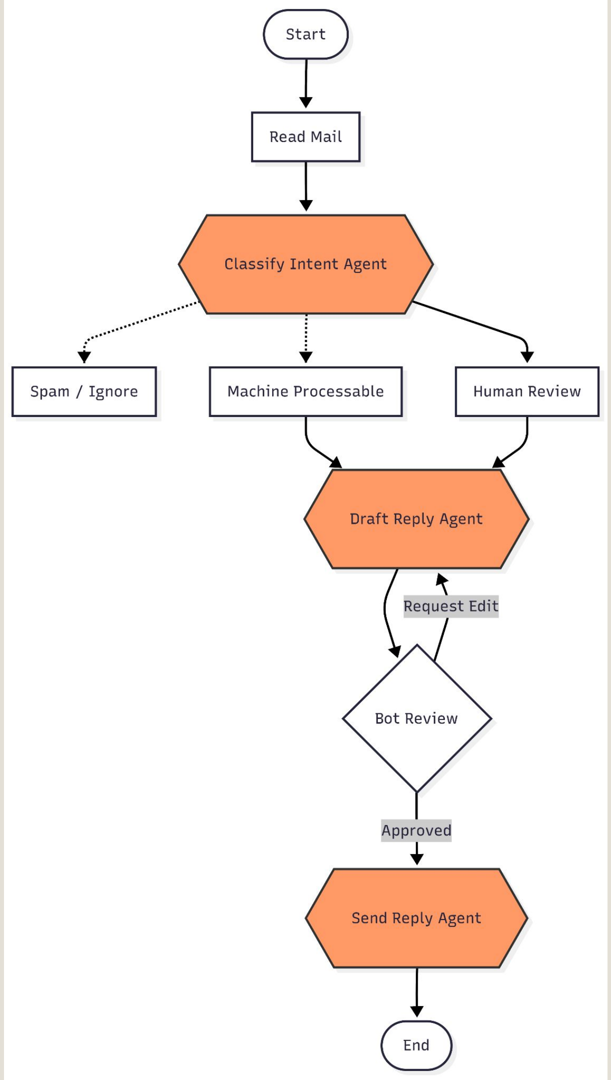






Agentic System

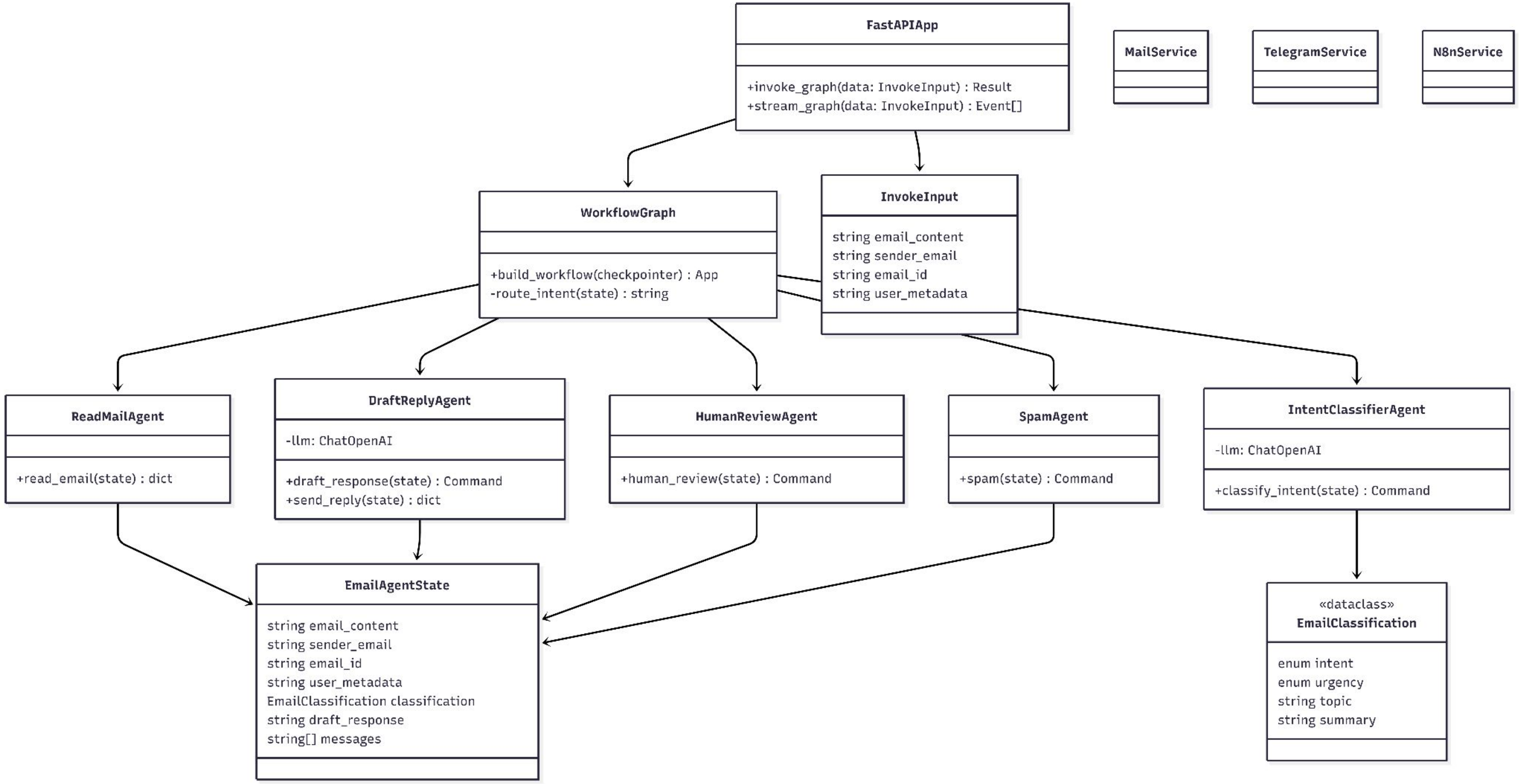
- Built using LangGraph
- Consists of several nodes with distinct functionalities
- Provides interface between the frontend and the VLLM
- Core of the project
- A detailed overview is given in the next slides



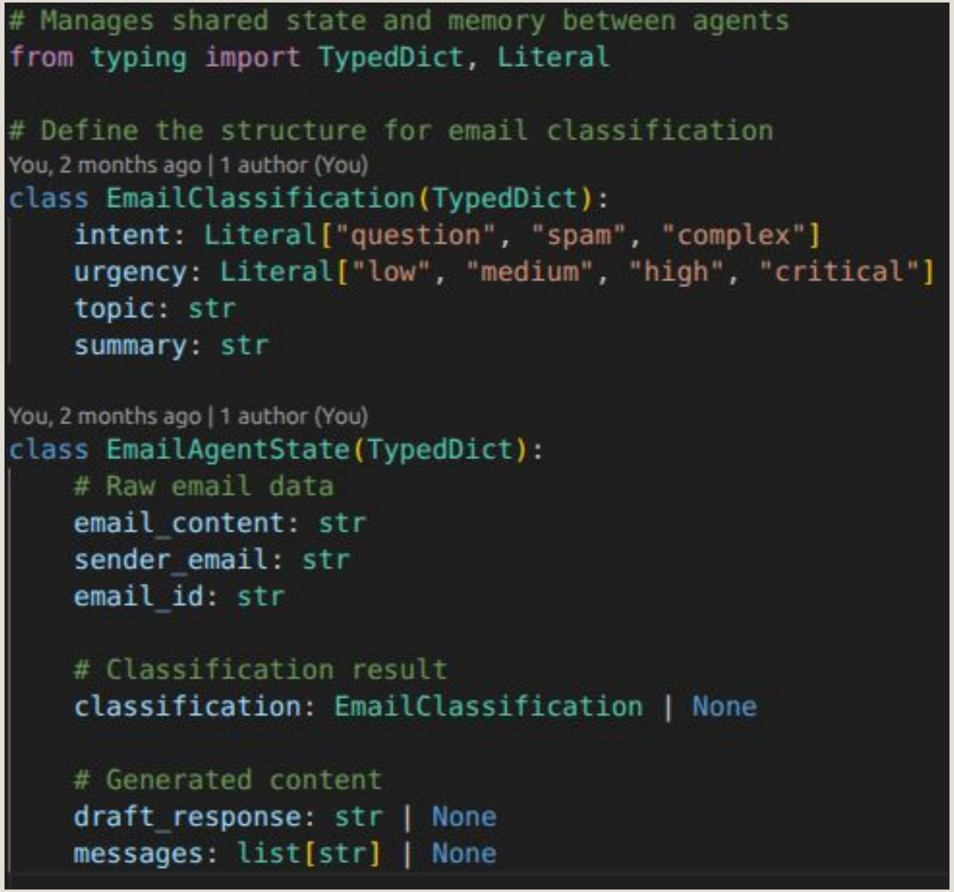
Agentic System

- Emails have their intent classified
- A response email is drafted
- Human-in-the-loop allows a person to re-draft
- Finally, the email can be sent

Agentic System

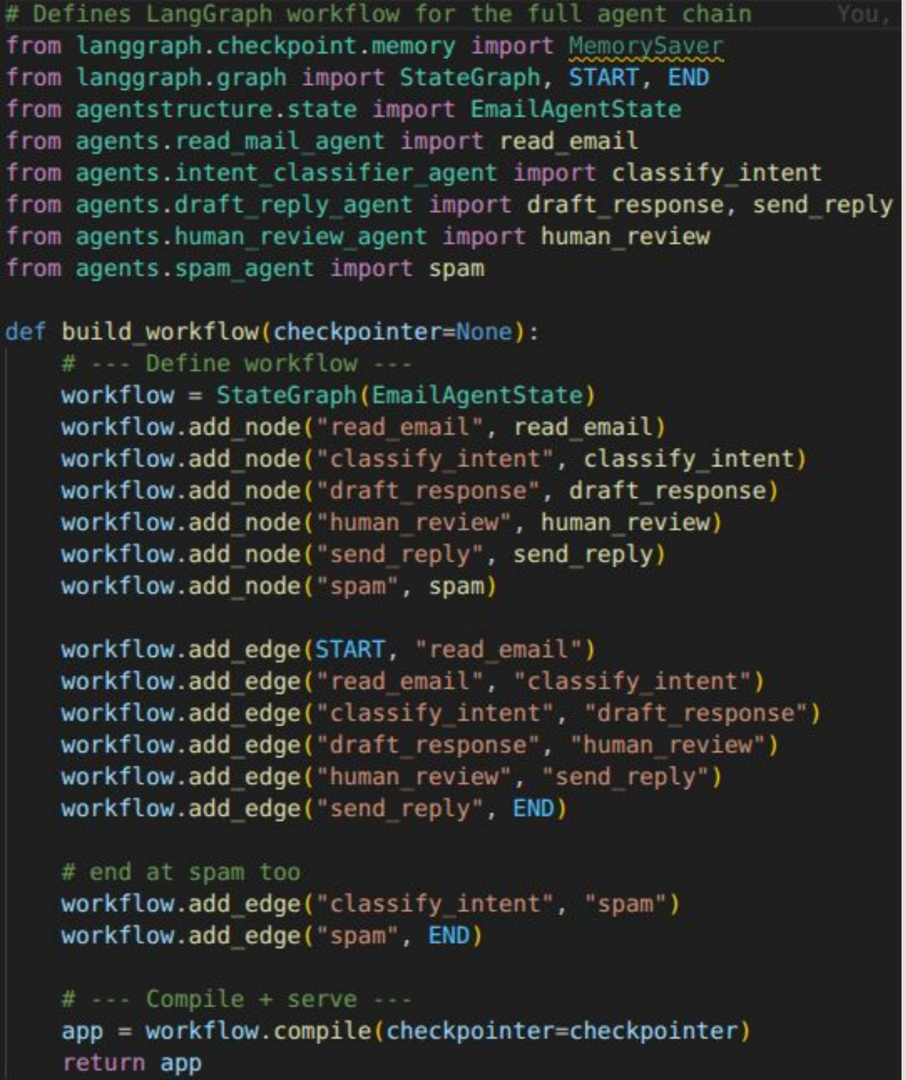


Agentic System - State



```
# Manages shared state and memory between agents
from typing import TypedDict, Literal

# Define the structure for email classification
You, 2 months ago | 1 author (You)
class EmailClassification(TypedDict):
    intent: Literal["question", "spam", "complex"]
    urgency: Literal["low", "medium", "high", "critical"]
    topic: str
    summary: str

You, 2 months ago | 1 author (You)
class EmailAgentState(TypedDict):
    # Raw email data
    email_content: str
    sender_email: str
    email_id: str

    # Classification result
    classification: EmailClassification | None

    # Generated content
    draft_response: str | None
    messages: list[str] | None
```

- Fundamentally, the system passes around state
- This is a wrapper around a TypedDict
- Information relevant to the system is stored and modified via state

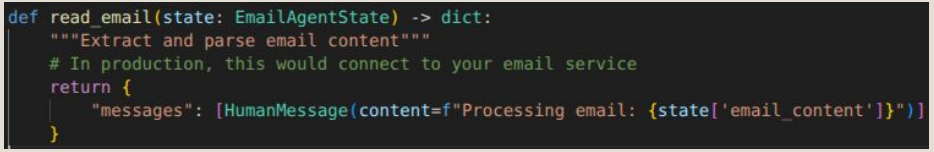


```
# Defines LangGraph workflow for the full agent chain                    You,
from langgraph.checkpoint.memory import MemorySaver
from langgraph.graph import StateGraph, START, END
from agentstructure.state import EmailAgentState
from agents.read_mail_agent import read_email
from agents.intent_classifier_agent import classify_intent
from agents.draft_reply_agent import draft_response, send_reply
from agents.human_review_agent import human_review
from agents.spam_agent import spam

def build_workflow(checkpointer=None):
    # --- Define workflow ---
    workflow = StateGraph(EmailAgentState)
    workflow.add_node("read_email", read_email)
    workflow.add_node("classify_intent", classify_intent)
    workflow.add_node("draft_response", draft_response)
    workflow.add_node("human_review", human_review)
    workflow.add_node("send_reply", send_reply)
    workflow.add_node("spam", spam)

    workflow.add_edge(START, "read_email")
    workflow.add_edge("read_email", "classify_intent")
    workflow.add_edge("classify_intent", "draft_response")
    workflow.add_edge("draft_response", "human_review")
    workflow.add_edge("human_review", "send_reply")
    workflow.add_edge("send_reply", END)

    # end at spam too
    workflow.add_edge("classify_intent", "spam")
    workflow.add_edge("spam", END)

    # --- Compile + serve ---
    app = workflow.compile(checkpointer=checkpointer)
    return app
```

Agentic System - Graph

- After state is defined, we define edges and nodes
- **Nodes** are discrete actions
- **Edges** are where state can flow through the graph
- The code shown is the graph API of LangGraph
- There is also a functional API

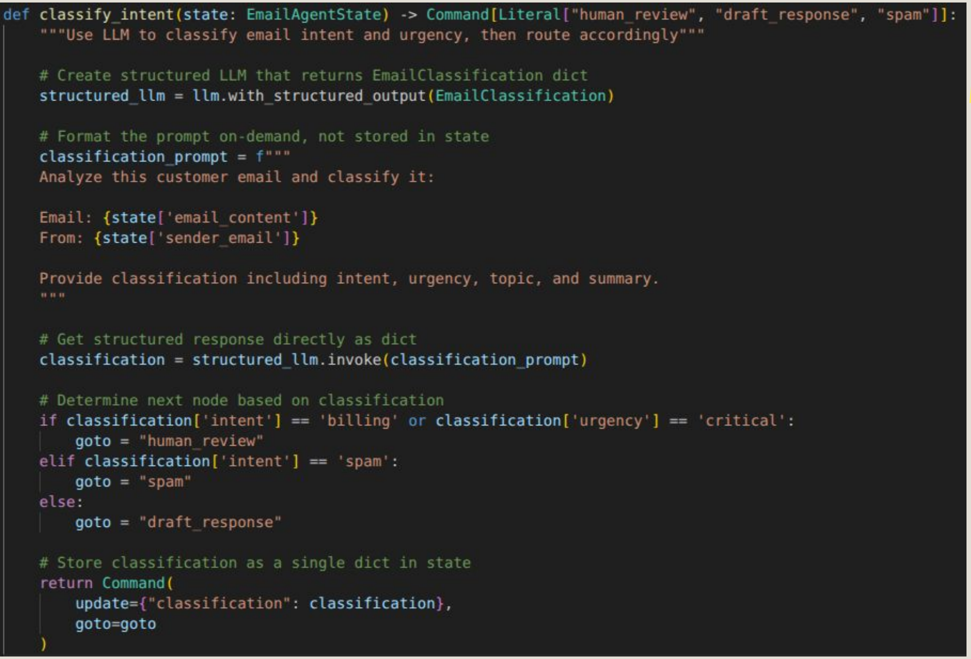
Agentic System - Read Mail



```python
def read_email(state: EmailAgentState) -> dict:
    """Extract and parse email content"""
    # In production, this would connect to your email service
    return {
        "messages": [HumanMessage(content=f"Processing email: {state['email_content']}")]
    }
```

- This is the read email node
- It shows how state will be updated
- Shows how the email is initially processed

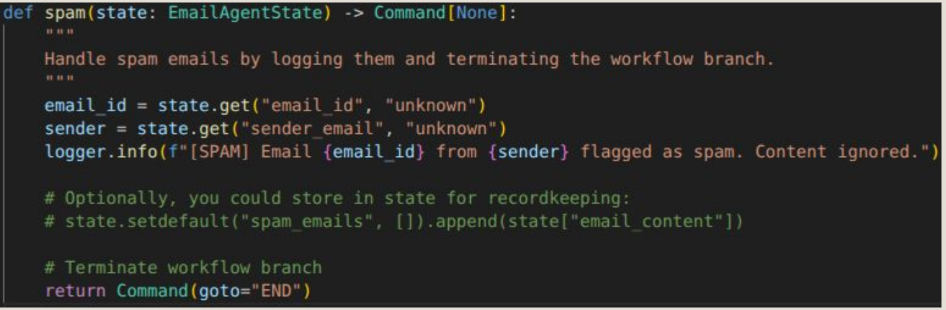
Agentic System - Classify Intent



```python
def classify_intent(state: EmailAgentState) -> Command[Literal["human_review", "draft_response", "spam"]]:
    """Use LLM to classify email intent and urgency, then route accordingly"""

    # Create structured LLM that returns EmailClassification dict
    structured_llm = llm.with_structured_output(EmailClassification)

    # Format the prompt on-demand, not stored in state
    classification_prompt = f"""
    Analyze this customer email and classify it:

    Email: {state['email_content']}
    From: {state['sender_email']}

    Provide classification including intent, urgency, topic, and summary.
    """

    # Get structured response directly as dict
    classification = structured_llm.invoke(classification_prompt)

    # Determine next node based on classification
    if classification['intent'] == 'billing' or classification['urgency'] == 'critical':
        goto = "human_review"
    elif classification['intent'] == 'spam':
        goto = "spam"
    else:
        goto = "draft_response"

    # Store classification as a single dict in state
    return Command(
        update={"classification": classification},
        goto=goto
    )
```

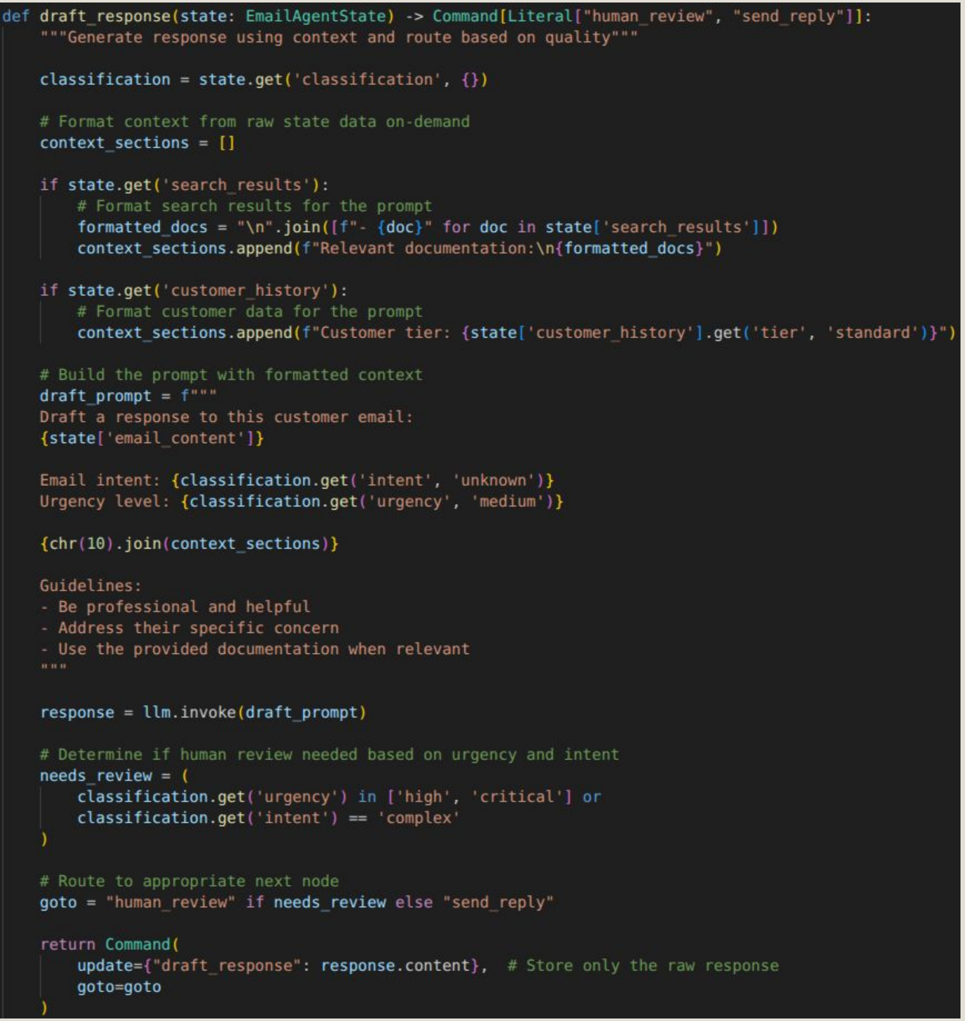
- Classifies email's intent and urgency
- Basic prompt engineering done to get the classification
- Structured output used to ensure the classifications exist
- After intent and urgency are decided, routing based on the classification happens
- State is updated

Agentic System - Spam



```
def spam(state: EmailAgentState) -> Command[None]:
    """
    Handle spam emails by logging them and terminating the workflow branch.
    """
    email_id = state.get("email_id", "unknown")
    sender = state.get("sender_email", "unknown")
    logger.info(f"[SPAM] Email {email_id} from {sender} flagged as spam. Content ignored.")

    # Optionally, you could store in state for recordkeeping:
    # state.setdefault("spam_emails", []).append(state["email_content"])

    # Terminate workflow branch
    return Command(goto="END")
```

- If the mail was classified as spam we end the graph
- In future work or a business context, these could be stored or sent to cybersecurity
- Here we just log them

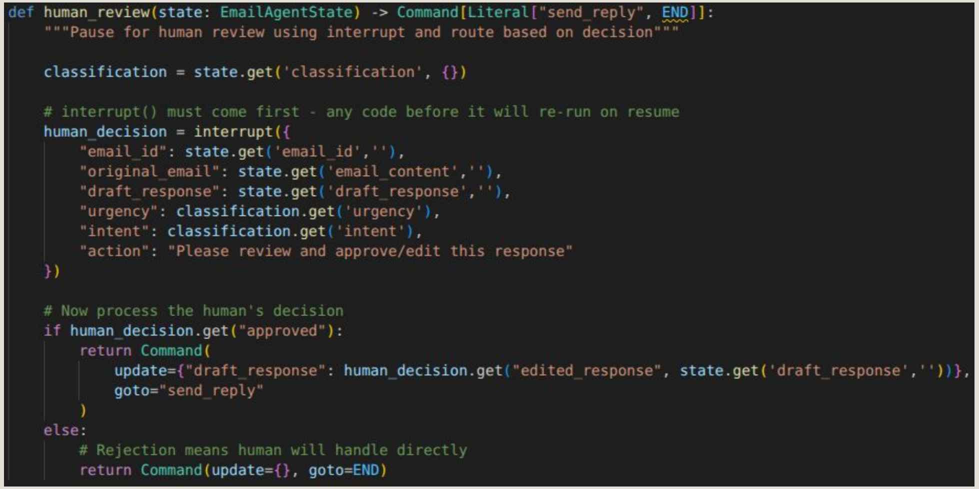


```python
def draft_response(state: EmailAgentState) -> Command[Literal["human_review", "send_reply"]]:
    """Generate response using context and route based on quality"""

    classification = state.get('classification', {})

    # Format context from raw state data on-demand
    context_sections = []

    if state.get('search_results'):
        # Format search results for the prompt
        formatted_docs = "\n".join([f"- {doc}" for doc in state['search_results']])
        context_sections.append(f"Relevant documentation:\n{formatted_docs}")

    if state.get('customer_history'):
        # Format customer data for the prompt
        context_sections.append(f"Customer tier: {state['customer_history'].get('tier', 'standard')}")

    # Build the prompt with formatted context
    draft_prompt = f"""
    Draft a response to this customer email:
    {state['email_content']}

    Email intent: {classification.get('intent', 'unknown')}
    Urgency level: {classification.get('urgency', 'medium')}

    {chr(10).join(context_sections)}

    Guidelines:
    - Be professional and helpful
    - Address their specific concern
    - Use the provided documentation when relevant
    """

    response = llm.invoke(draft_prompt)

    # Determine if human review needed based on urgency and intent
    needs_review = (
        classification.get('urgency') in ['high', 'critical'] or
        classification.get('intent') == 'complex'
    )

    # Route to appropriate next node
    goto = "human_review" if needs_review else "send_reply"

    return Command(
        update={"draft_response": response.content},  # Store only the raw response
        goto=goto
    )
```

Agentic System - Draft Response

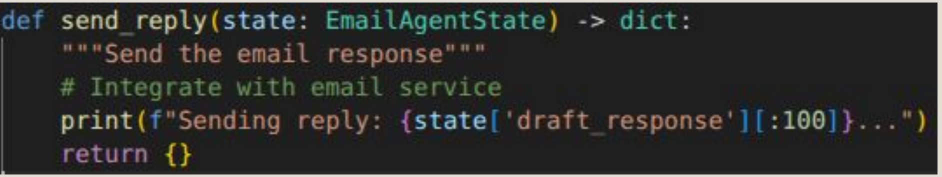
- Drafts a response based on state
- Initially modeled from the LangGraph website demo where they had customer service emails
- After response is drafted, classification from previous step is used to determine the review status
- The response is sent to the send reply or human review

Agentic System - Human Review



```python
def human_review(state: EmailAgentState) -> Command[Literal["send_reply", END]]:
    """Pause for human review using interrupt and route based on decision"""

    classification = state.get('classification', {})

    # interrupt() must come first - any code before it will re-run on resume
    human_decision = interrupt({
        "email_id": state.get('email_id',''),
        "original_email": state.get('email_content',''),
        "draft_response": state.get('draft_response',''),
        "urgency": classification.get('urgency'),
        "intent": classification.get('intent'),
        "action": "Please review and approve/edit this response"
    })

    # Now process the human's decision
    if human_decision.get("approved"):
        return Command(
            update={"draft_response": human_decision.get("edited_response", state.get('draft_response',''))},
            goto="send_reply"
        )
    else:
        # Rejection means human will handle directly
        return Command(update={}, goto=END)
```

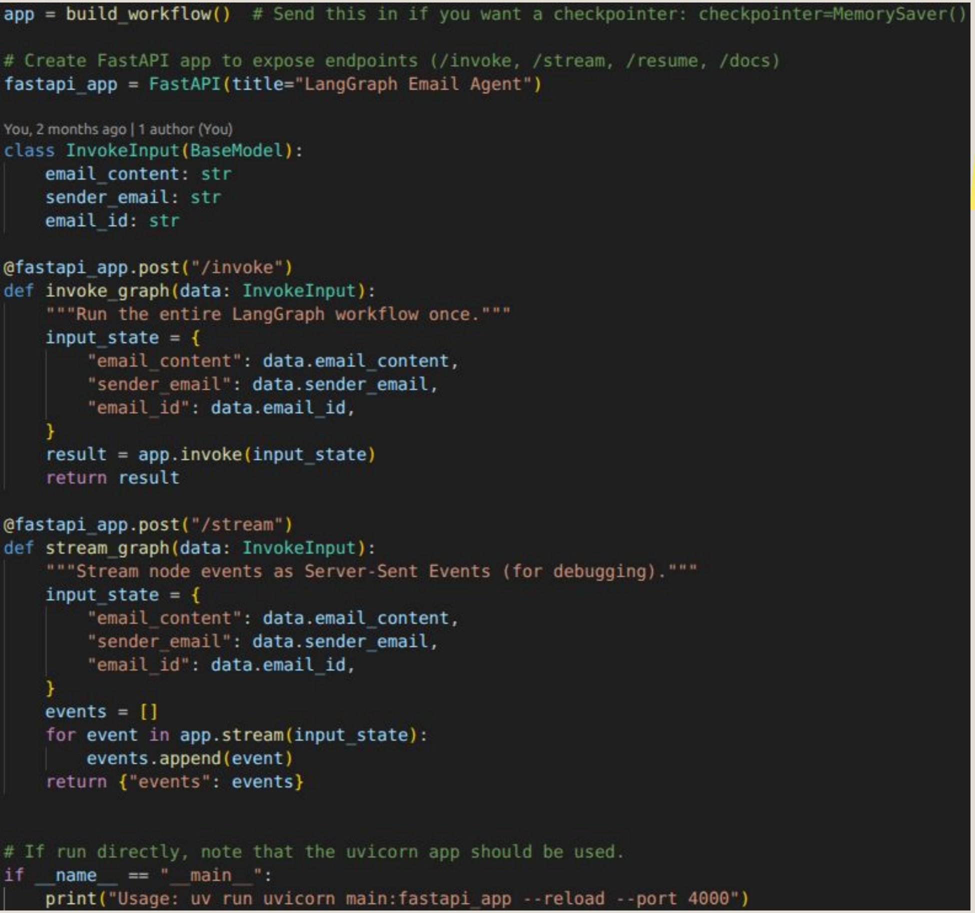
- The human review node allows the Human-in-the-loop functionality of LangGraph to be utilized
- Human decides to give the green light or to stop the process
- This lightens the load of the humans tasked with answering emails



```
def send_reply(state: EmailAgentState) -> dict:
    """Send the email response"""
    # Integrate with email service
    print(f"Sending reply: {state['draft_response'][:100]}...")
    return {}
```

Agentic System - Send Reply

- The final send reply function sends the response.
- In our case, it no-ops since we haven't connected the demo to an email service.
- This would be future work.



```python
app = build_workflow()  # Send this in if you want a checkpointer: checkpointer=MemorySaver()

# Create FastAPI app to expose endpoints (/invoke, /stream, /resume, /docs)
fastapi_app = FastAPI(title="LangGraph Email Agent")

You, 2 months ago | 1 author (You)
class InvokeInput(BaseModel):
    email_content: str
    sender_email: str
    email_id: str

@fastapi_app.post("/invoke")
def invoke_graph(data: InvokeInput):
    """Run the entire LangGraph workflow once."""
    input_state = {
        "email_content": data.email_content,
        "sender_email": data.sender_email,
        "email_id": data.email_id,
    }
    result = app.invoke(input_state)
    return result

@fastapi_app.post("/stream")
def stream_graph(data: InvokeInput):
    """Stream node events as Server-Sent Events (for debugging)."""
    input_state = {
        "email_content": data.email_content,
        "sender_email": data.sender_email,
        "email_id": data.email_id,
    }
    events = []
    for event in app.stream(input_state):
        events.append(event)
    return {"events": events}


# If run directly, note that the uvicorn app should be used.
if __name__ == "__main__":
    print("Usage: uv run uvicorn main:fastapi_app --reload --port 4000")
```

Agentic System - Bringing it Together

- Exposes a FastAPI to allow the front end to talk to the LangGraph System
- This puts everything together and is where any front-end would go to talk to it
- In practice, you could hook this up to an email service but we demo it with streamlit

Frontend

- Built a Streamlit app
- Why Streamlit?
  - Rapid prototyping of AI applications.
  - Directly integrates with the Python backend logic.

Streamlit

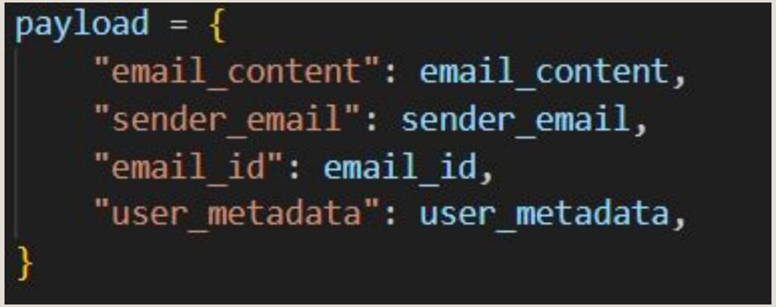
Frontend

The purpose was :

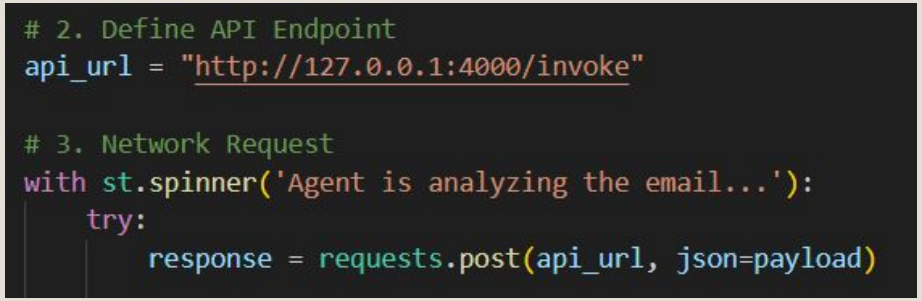
- Allows for input of an email
- Returns a draft email, a processing/priority classification
- Could be replaced with a service such as n8n

Frontend

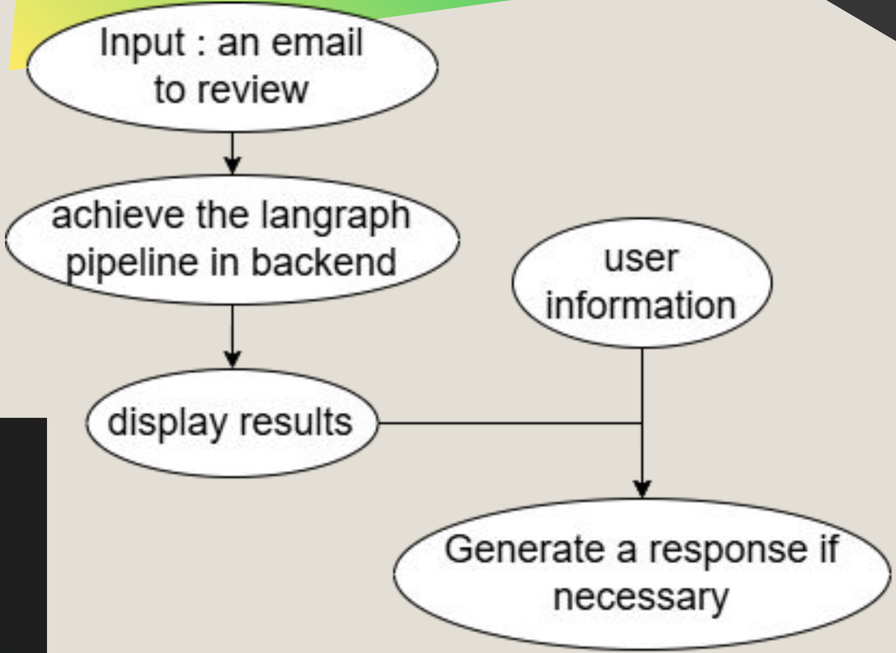
How this application work ?



```
payload = {
    "email_content": email_content,
    "sender_email": sender_email,
    "email_id": email_id,
    "user_metadata": user_metadata,
}
```

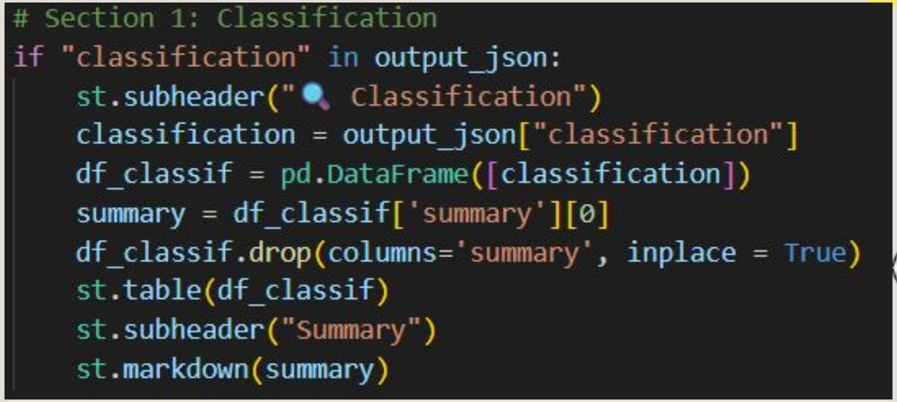


```
# 2. Define API Endpoint
api_url = "http://127.0.0.1:4000/invoke"

# 3. Network Request
with st.spinner('Agent is analyzing the email...'):
    try:
        response = requests.post(api_url, json=payload)
```

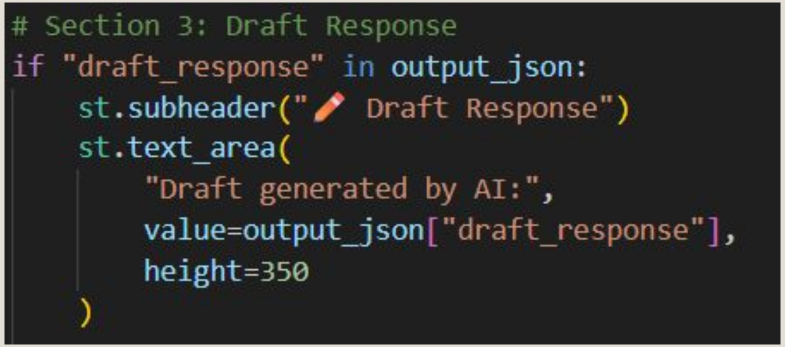

Input : an email to review
achieve the langraph pipeline in backend
user information
display results
Generate a response if necessary

Frontend

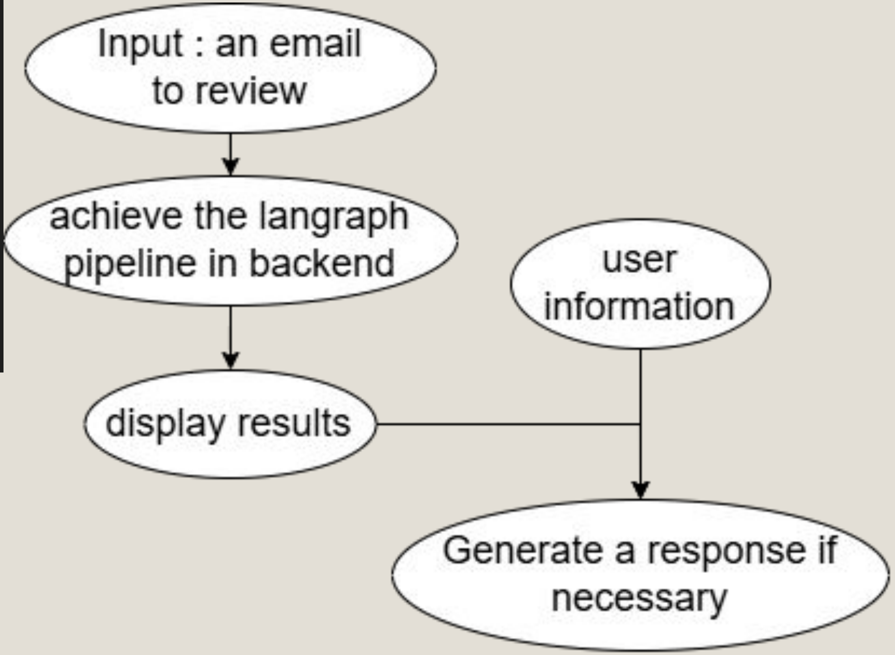
How this application work ?



```
# Section 1: Classification
if "classification" in output_json:
    st.subheader("🔍 Classification")
    classification = output_json["classification"]
    df_classif = pd.DataFrame([classification])
    summary = df_classif['summary'][0]
    df_classif.drop(columns='summary', inplace = True)
    st.table(df_classif)
    st.subheader("Summary")
    st.markdown(summary)
```

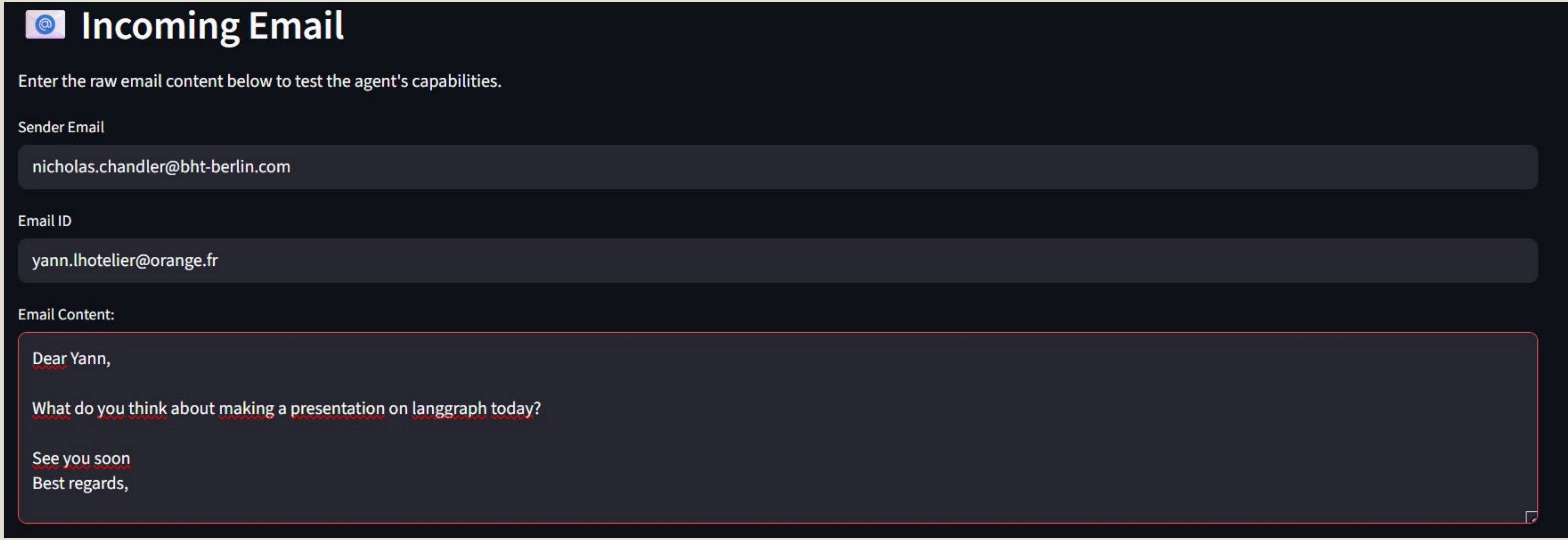


```
# Section 3: Draft Response
if "draft_response" in output_json:
    st.subheader("✏️ Draft Response")
    st.text_area(
        "Draft generated by AI:",
        value=output_json["draft_response"],
        height=350
    )
```


Input : an email to review
achieve the langraph pipeline in backend
display results
user information
Generate a response if necessary

Frontend

Why this is working ?



Incoming Email

Enter the raw email content below to test the agent's capabilities.

Sender Email

nicholas.chandler@bht-berlin.com

Email ID

yann.lhotelier@orange.fr

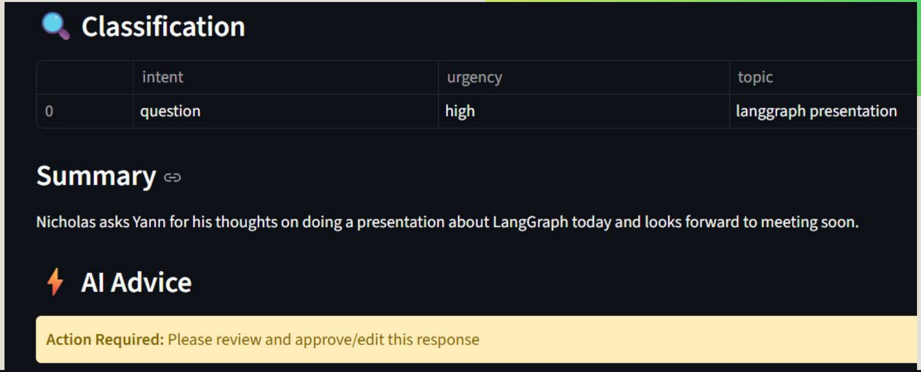
Email Content:

Dear Yann,

What do you think about making a presentation on langgraph today?

See you soon
Best regards,

Frontend



Classification

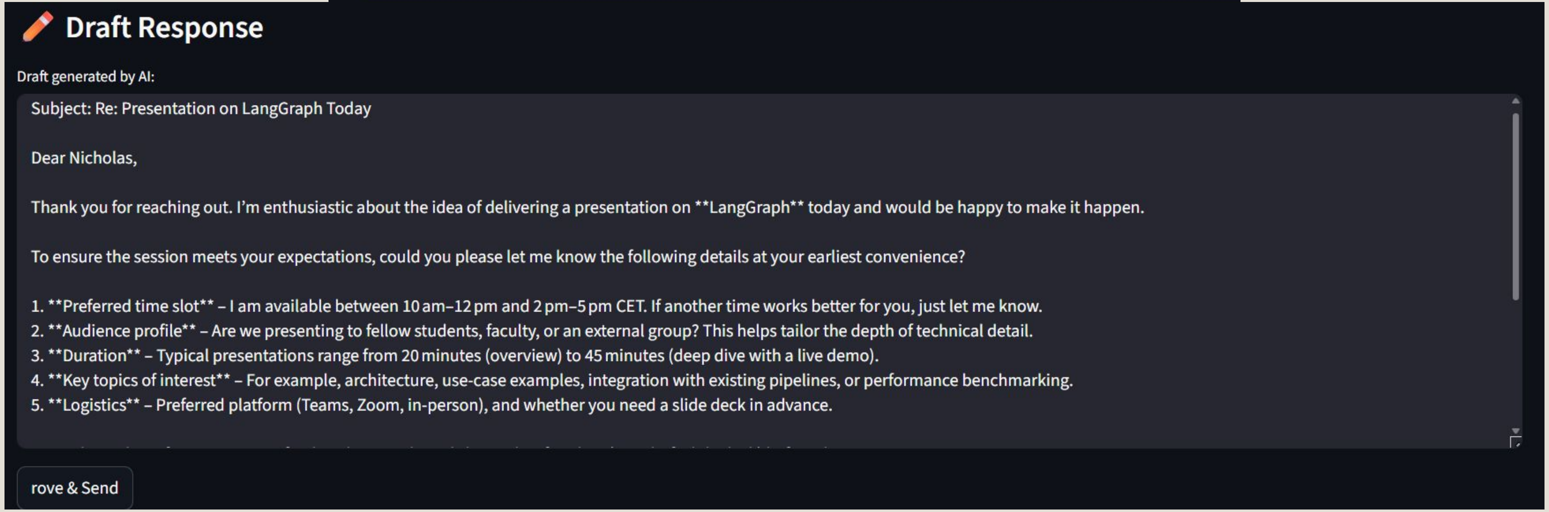
	intent	urgency	topic
0	question	high	langgraph presentation

Summary

Nicholas asks Yann for his thoughts on doing a presentation about LangGraph today and looks forward to meeting soon.

AI Advice

Action Required: Please review and approve/edit this response



Draft Response

Draft generated by AI:

Subject: Re: Presentation on LangGraph Today

Dear Nicholas,

Thank you for reaching out. I'm enthusiastic about the idea of delivering a presentation on **LangGraph** today and would be happy to make it happen.

To ensure the session meets your expectations, could you please let me know the following details at your earliest convenience?

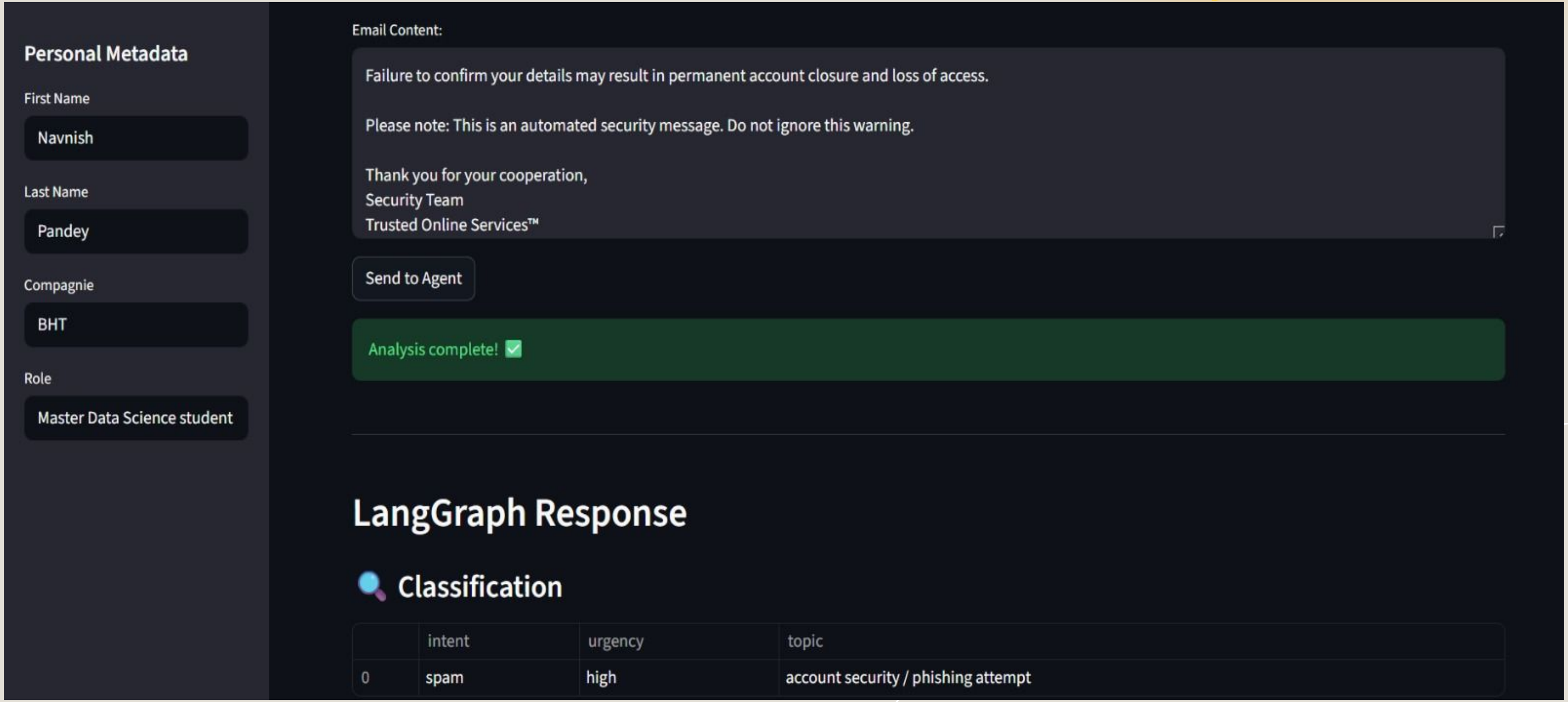
1. **Preferred time slot** – I am available between 10 am–12 pm and 2 pm–5 pm CET. If another time works better for you, just let me know.
2. **Audience profile** – Are we presenting to fellow students, faculty, or an external group? This helps tailor the depth of technical detail.
3. **Duration** – Typical presentations range from 20 minutes (overview) to 45 minutes (deep dive with a live demo).
4. **Key topics of interest** – For example, architecture, use-case examples, integration with existing pipelines, or performance benchmarking.
5. **Logistics** – Preferred platform (Teams, Zoom, in-person), and whether you need a slide deck in advance.

Error Analysis

Dataset

- 74 emails in English
    - 30 Spam emails
        - No priority category
        - E.g. promotions, phising
    - 28 Human Review emails
        - Priority category:
            - Low
            - Medium
            - High
        - E.g. billing system error
    - 16 Machine processable emails
        - Priority category (as above)
        - E.g. questions about information in documentation

Example Spam Email



Personal Metadata

First Name	Navnish
Last Name	Pandey
Compagnie	BHT
Role	Master Data Science student

Email Content:

```
Failure to confirm your details may result in permanent account closure and loss of access.

Please note: This is an automated security message. Do not ignore this warning.

Thank you for your cooperation,
Security Team
Trusted Online Services™
```

Send to Agent

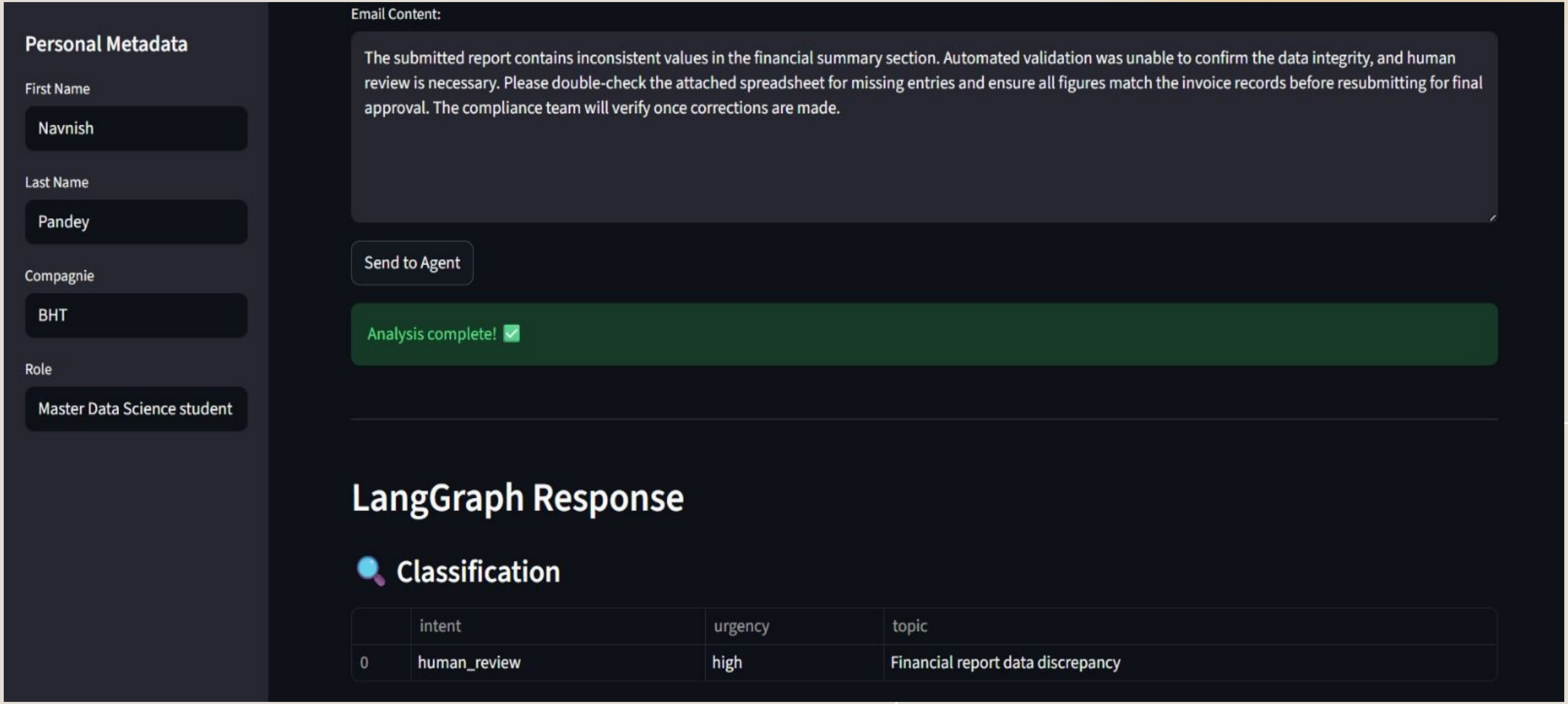
Analysis complete! ✅

LangGraph Response

🔍 Classification

	intent	urgency	topic
0	spam	high	account security / phishing attempt

Example Human Review Email



Personal Metadata

First Name	Navnish
Last Name	Pandey
Compagnie	BHT
Role	Master Data Science student

Email Content:

The submitted report contains inconsistent values in the financial summary section. Automated validation was unable to confirm the data integrity, and human review is necessary. Please double-check the attached spreadsheet for missing entries and ensure all figures match the invoice records before resubmitting for final approval. The compliance team will verify once corrections are made.

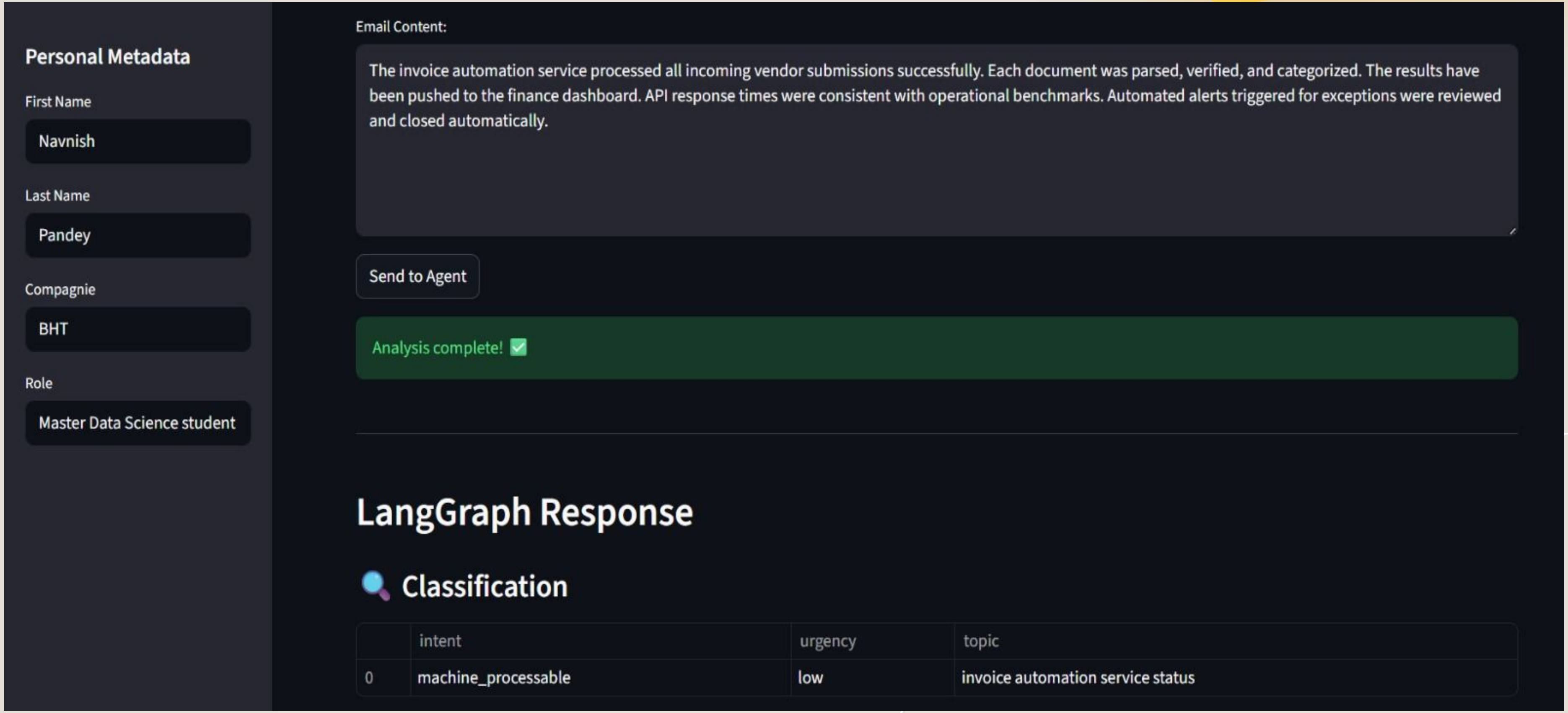
Analysis complete! ✅

LangGraph Response

🔍 Classification

	intent	urgency	topic
0	human_review	high	Financial report data discrepancy

Example Machine Processable Email



Personal Metadata

First Name	Navnish
Last Name	Pandey
Compagnie	BHT
Role	Master Data Science student

Email Content:

The invoice automation service processed all incoming vendor submissions successfully. Each document was parsed, verified, and categorized. The results have been pushed to the finance dashboard. API response times were consistent with operational benchmarks. Automated alerts triggered for exceptions were reviewed and closed automatically.

Analysis complete! ✅

LangGraph Response

🔍 Classification

	intent	urgency	topic
0	machine_processable	low	invoice automation service status

Quantitative Error Analysis

- The classifier node of our system classified the processing step and priority of each email
- We report:
  - Accuracy: Number of correct predictions / total predictions
  - Precision (macro): TP / (TP+FP)
  - Recall (macro): TP / (TP+FN)
  - F1 Score (macro): 2 · (Precision · Recall) / (Precision + Recall)

Quantitative Error Analysis - Processing Step

- Accuracy: 94.59%
- Precision (macro): 72.58%
- Recall (macro): 69.48%
- F1 Score (macro): 70.72%

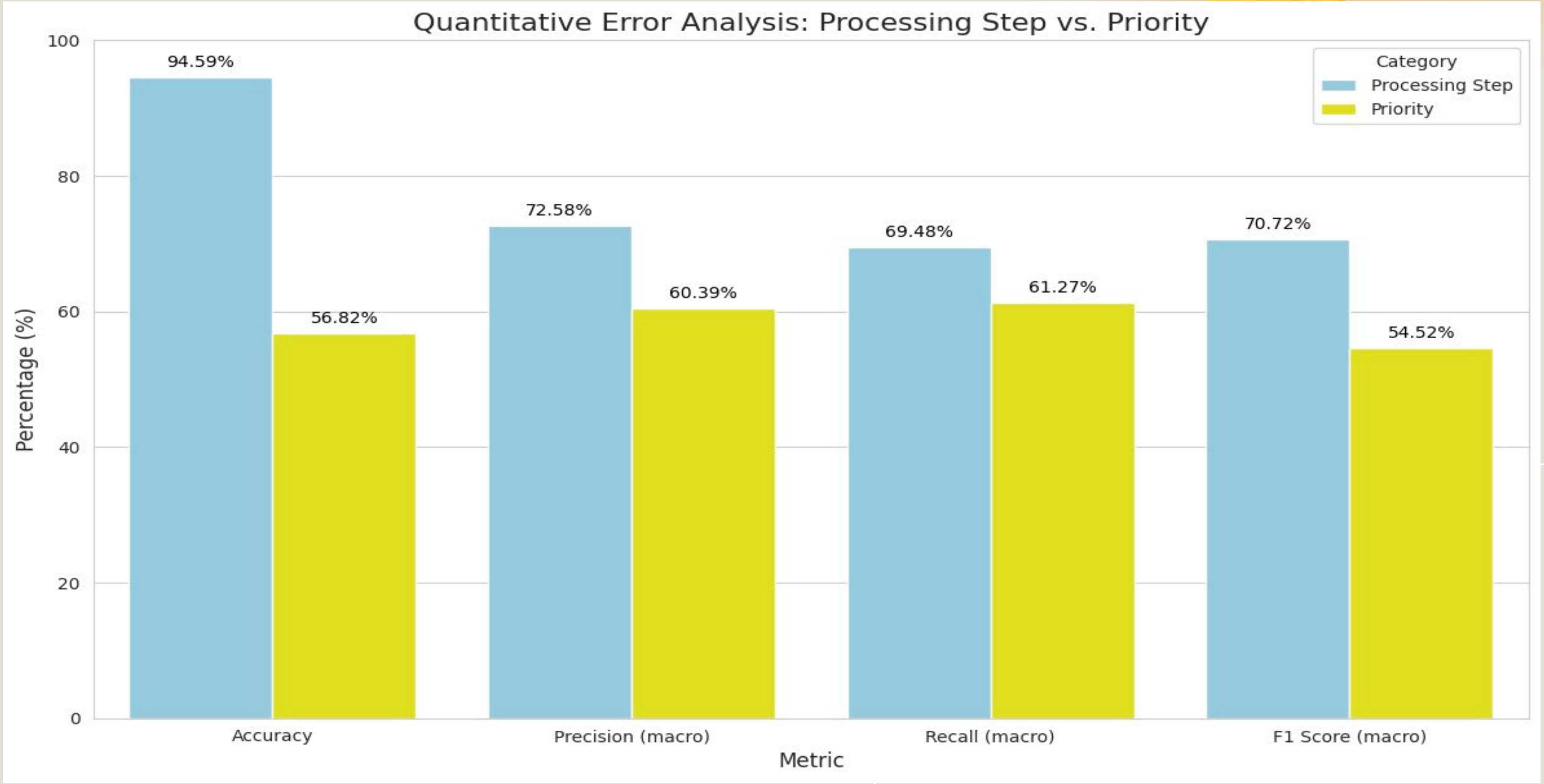
- The accuracy is quite high and the F1 score is tolerable.
- This shows that the system is able to leverage the LLM for direct classification

Quantitative Error Analysis - Priority

- Accuracy: 56.82%
- Precision (macro): 60.39%
- Recall (macro): 61.27%
- F1 Score (macro): 54.52%

- This is far lower than for the processing classification
  - A possible cause could be the focus in the system being to get the processing right

Quantitative Error Analysis - Processing vs Priority



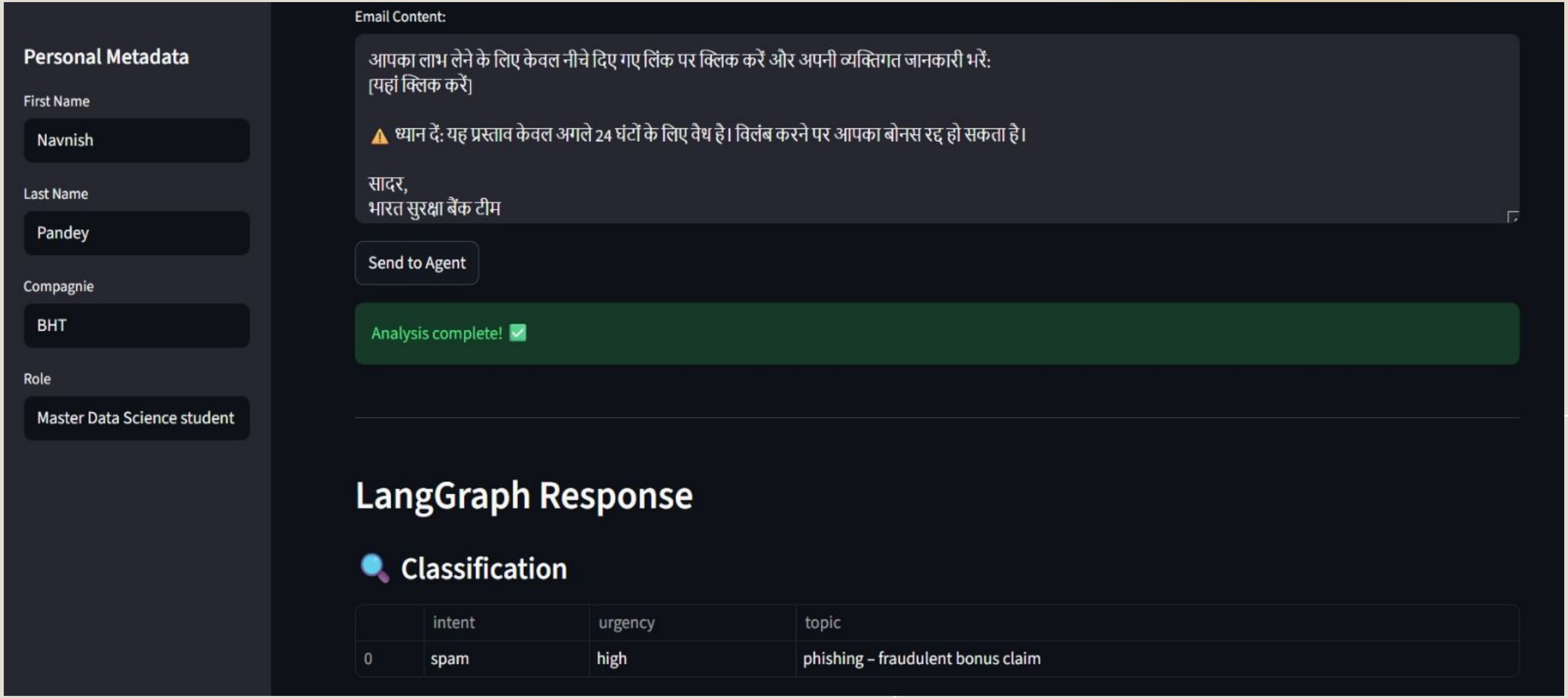
Analysis on different languages emails

- Multilingual comprehension: Ensures accurate understanding of intent and context across different languages like Hindi, German, and English.

- Consistent classification: Validates reliable spam detection and priority handling across different languages.

- Quality responses: Confirms natural and context-aware response generation for global scalability.

Classifying Hindi Language Email as Spam



Personal Metadata

First Name	Navnish
Last Name	Pandey
Compagnie	BHT
Role	Master Data Science student

Email Content:

आपका लाभ लेने के लिए केवल नीचे दिए गए लिंक पर क्लिक करें और अपनी व्यक्तिगत जानकारी भरें:
[यहां क्लिक करें]

⚠ ध्यान दें: यह प्रस्ताव केवल अगले 24 घंटों के लिए वैध है। विलंब करने पर आपका बोनस रद्द हो सकता है।

सादर,
भारत सुरक्षा बैंक टीम

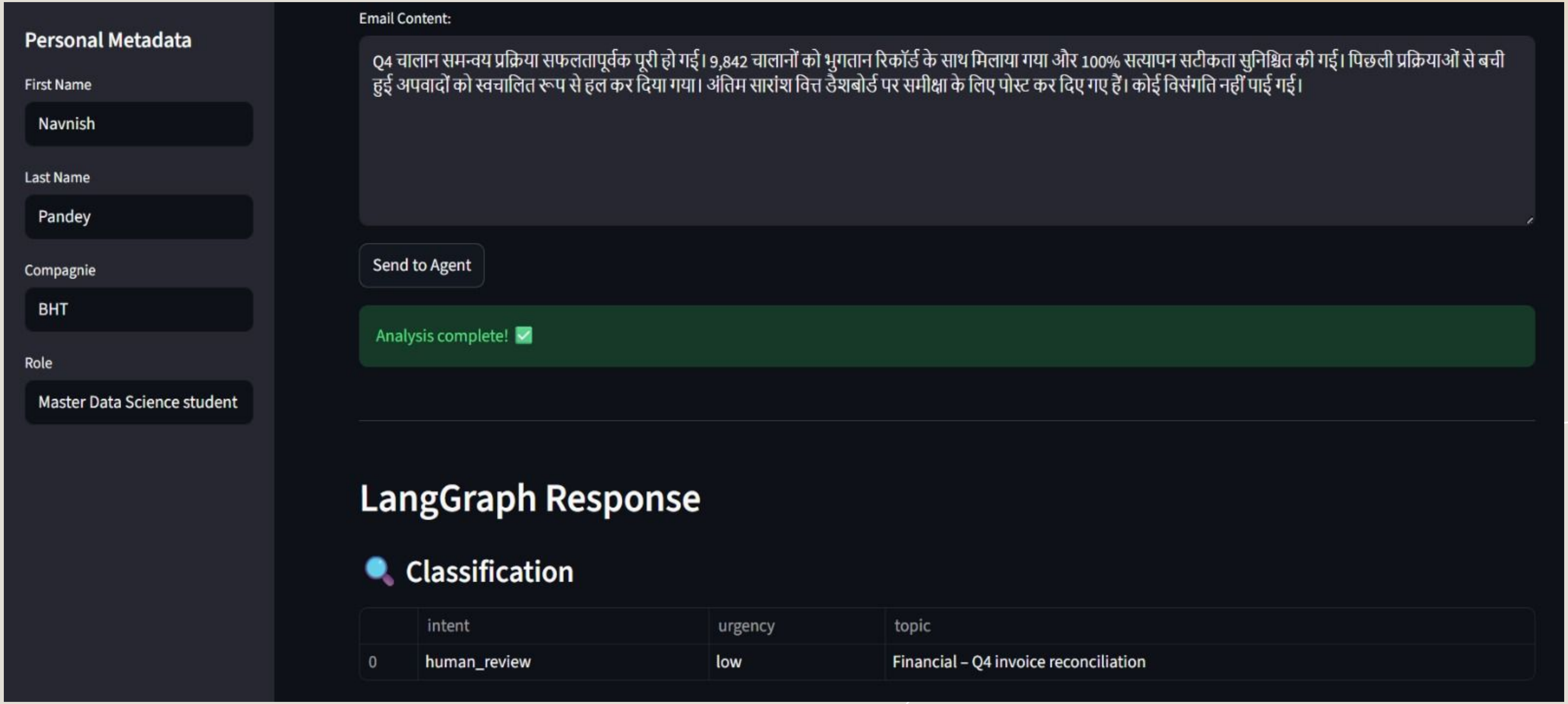
Analysis complete! ✅

LangGraph Response

🔍 Classification

	intent	urgency	topic
0	spam	high	phishing – fraudulent bonus claim

Classifying Hindi Language Email Intent for Human review



Personal Metadata

First Name	Navnish
Last Name	Pandey
Compagnie	BHT
Role	Master Data Science student

Email Content:

Q4 चालान समन्वय प्रक्रिया सफलतापूर्वक पूरी हो गई। 9,842 चालानों को भुगतान रिकॉर्ड के साथ मिलाया गया और 100% सत्यापन सटीकता सुनिश्चित की गई। पिछली प्रक्रियाओं से बची हुई अपवादों को स्वचालित रूप से हल कर दिया गया। अंतिम सारांश वित्त डैशबोर्ड पर समीक्षा के लिए पोस्ट कर दिए गए हैं। कोई विसंगति नहीं पाई गई।

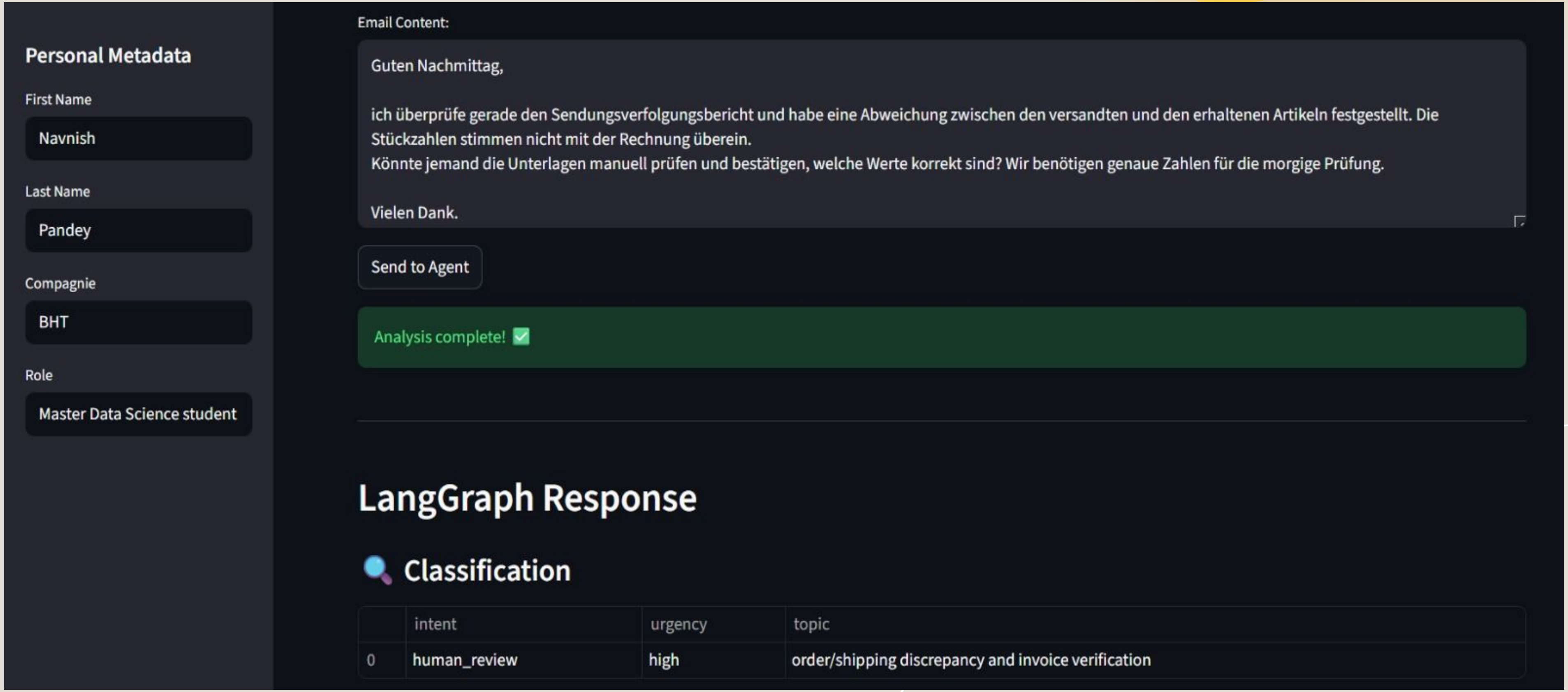
Analysis complete! ✅

LangGraph Response

🔍 **Classification**

	intent	urgency	topic
0	human_review	low	Financial – Q4 invoice reconciliation

Classifying Deutsch Language Email Intent as Human review



Personal Metadata

First Name	Navnish
Last Name	Pandey
Compagnie	BHT
Role	Master Data Science student

Email Content:

Guten Nachmittag,

ich überprüfe gerade den Sendungsverfolgungsbericht und habe eine Abweichung zwischen den versandten und den erhaltenen Artikeln festgestellt. Die Stückzahlen stimmen nicht mit der Rechnung überein.
Könnte jemand die Unterlagen manuell prüfen und bestätigen, welche Werte korrekt sind? Wir benötigen genaue Zahlen für die morgige Prüfung.

Vielen Dank.

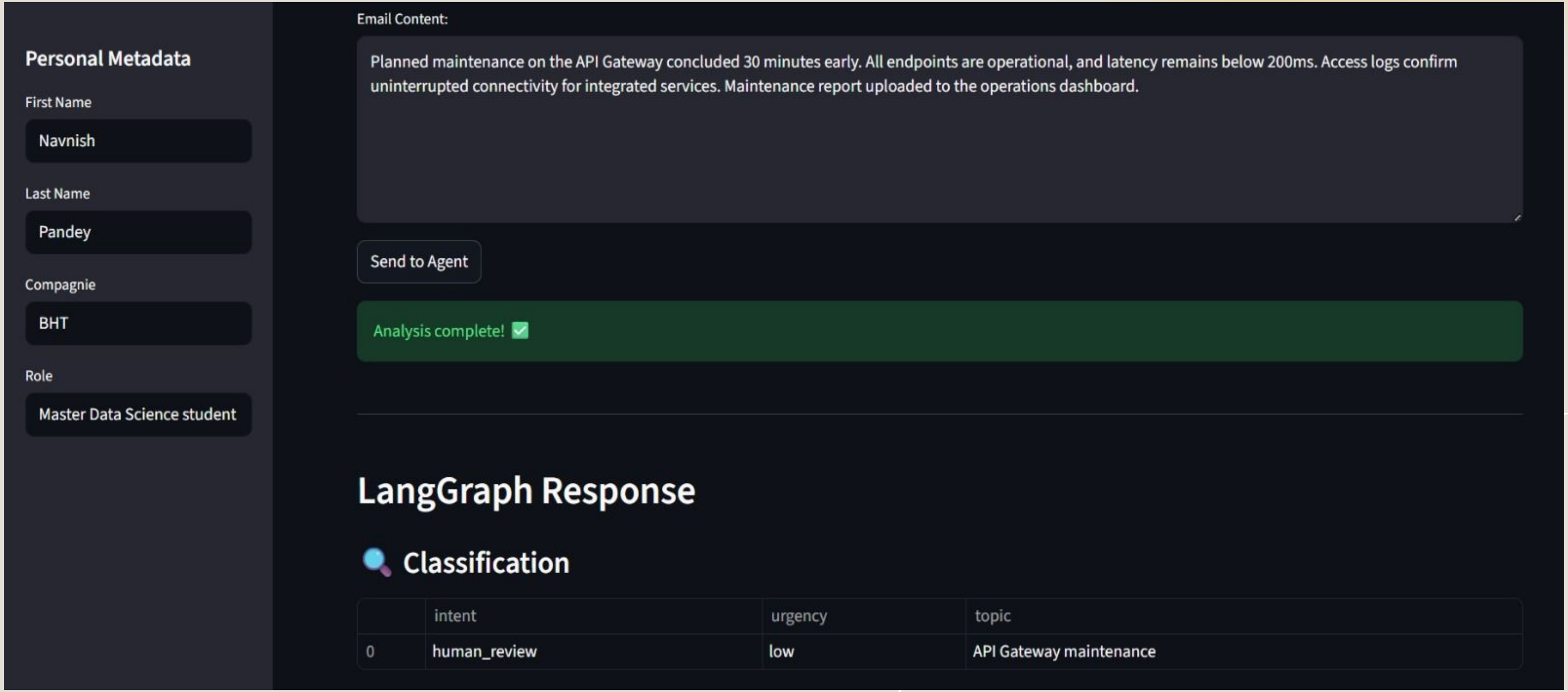
Analysis complete! ✅

LangGraph Response

🔍 **Classification**

	intent	urgency	topic
0	human_review	high	order/shipping discrepancy and invoice verification

Misclassification of Email Intent



Personal Metadata

First Name: Navnish

Last Name: Pandey

Compagnie: BHT

Role: Master Data Science student

Email Content:

Planned maintenance on the API Gateway concluded 30 minutes early. All endpoints are operational, and latency remains below 200ms. Access logs confirm uninterrupted connectivity for integrated services. Maintenance report uploaded to the operations dashboard.

Analysis complete! ✅

LangGraph Response

🔍 Classification

	intent	urgency	topic
0	human_review	low	API Gateway maintenance

Conclusion

What Did We Do?

- Built an agentic system using LangGraph to answer emails
- Examined how LangGraph works
- Did a technical deep dive into our system
- Looked at the system's performance

Thank you

Questions?

Sources

- https://openai.com/index/introducing-gpt-oss/
- https://docs.langchain.com/oss/python/langgraph/thinking-in-langgraph
- https://github.com/chandlerNick/langchain-multi-agent-demo
- https://forethought.ai/case-studies/achievers
- https://www.bbc.com/news/technology-68025677