Automating **Letter of Credit (LC) document checking** using **Agentic AI** is a strong candidate for intelligent task decomposition. As a Trade Finance expert, you already know LC document checking involves meticulous comparison of documents against LC terms under **UCP 600**, **ISBP 821**, and possibly **URC 522** for collections.

To implement this in **Agentic AI architecture**, you'd break down the process into **autonomous agents**, each responsible for a distinct skill aligned with real-world trade operations.

**🧠 Agentic AI Architecture for LC Document Checking**

**🎯 Goal**

Automatically analyze a set of trade documents submitted under an LC and determine **discrepancies**, **compliance**, and **next actions**.

**🧩 Agent Decomposition**

| **Agent Role** | **Function** | **Core Inputs** | **Outputs** |
| --- | --- | --- | --- |
| **1. LC Extractor Agent** | Parse and extract LC terms (amount, expiry, ports, latest shipment, documents required, etc.) | LC PDF or MT700 Swift message | Structured JSON with LC terms |
| **2. Document Classifier Agent** | Classify uploaded documents (e.g. Invoice, BL, CO, etc.) and match to LC requirements | Trade document set (PDF or scanned images) | Document-type-tagged data |
| **3. Data Extraction Agent** | Extract key fields (e.g., invoice value, BL shipper, CO origin) from each document | Tagged documents | JSON of extracted fields by doc |
| **4. Compliance Validator Agent** | Compare each document’s content with LC terms & UCP 600/ISBP 821 rules | LC terms + extracted doc data | List of compliant/non-compliant items |
| **5. Discrepancy Identification Agent** | Detect and explain discrepancies based on mismatches or missing items | Compliance results | Discrepancy list with ICC justification |
| **6. Decision Agent** | Recommend whether to accept, reject, or request waiver of discrepancies | Discrepancy data, bank policy | Recommendation with rationale |
| **7. Report Generator Agent** | Compile a full checking report for Trade Ops | All agent outputs | LC Document Checking Report (PDF/JSON) |
| **8. Feedback & Learning Agent** *(optional)* | Learn from user overrides (e.g. waived discrepancies) to improve logic | Manual corrections | Updated prompt/knowledge for validators |

**⚙️ Integration with Tools**

* **OCR** for scanned documents: Tesseract, Google Vision, or Azure OCR
* **LLMs**: For UCP/ISBP reasoning and extraction (GPT-4o, Claude, etc.)
* **Rule engine**: For hard-coded validations (e.g., date formats, field presence)
* **PDF parser**: PyMuPDF, PDFMiner, or LayoutLM for structured extraction

**📂 Example Flow**

1. User uploads LC (MT700) and 7 trade documents (Invoice, BL, Packing List, etc.)
2. LC Extractor Agent parses MT700 to JSON LC format.
3. Document Classifier Agent tags the uploaded files.
4. Data Extraction Agent parses fields like:
   * Invoice amount, date, terms
   * BL port of loading/discharge
   * Certificate country of origin
5. Compliance Validator Agent checks:
   * Is invoice amount ≤ LC amount?
   * Is port of discharge as per LC?
   * Is BL signed and dated before latest shipment?
6. Discrepancy Agent finds missing or mismatched fields.
7. Decision Agent uses bank policy (e.g., accept typographical discrepancy) to recommend "Accept with Waiver."
8. Report Generator outputs PDF summary with UCP references.

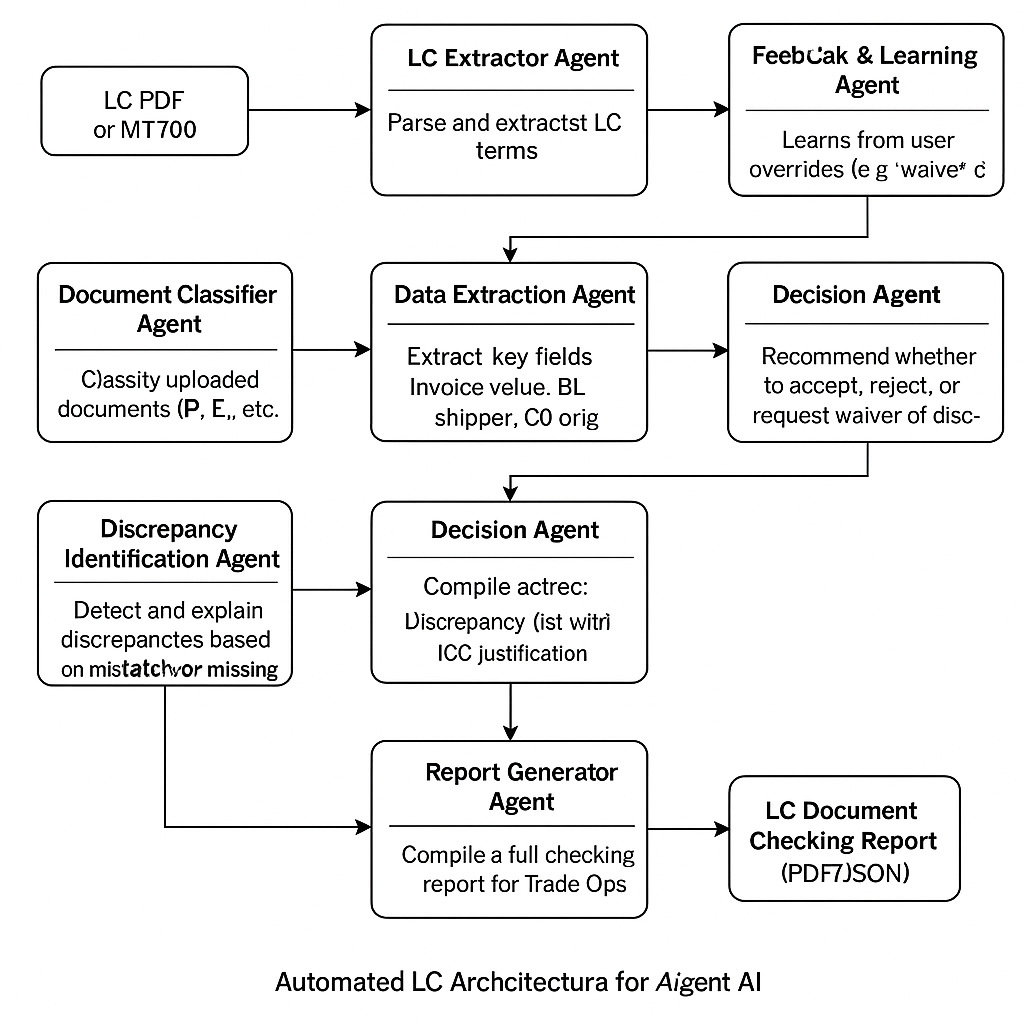
**✅ Real-World Extension**

* Integration into **back-office workflow tools** like Eximbills, Finastra, Surecomp, or Oracle Trade Finance
* API bridge to **SWIFT Alliance Access** for MT700/707/750/999 parsing
* Fine-tuning on **internal discrepancy archives** to improve Agent judgment

**🔐 Compliance Consideration**

* All outputs should **cite UCP/ISBP clauses** (e.g., “ISBP 745 para 25(a)”)
* System must support **auditable logs** for compliance review
* Manual override by LC checker must be preserved

**Demo architecture diagram**



**LangChain Code Example**

# lc\_doc\_check\_simulation.py

"""

Simulated Agentic AI pipeline for Letter of Credit (LC) document checking using LangChain.

This skeleton shows how you might wire together multiple autonomous agents (each as a LangChain

Tool or Runnable) to execute the full LC checking workflow:

1. LC Extractor Agent – parse MT700 / PDF LC to JSON terms

2. Document Classifier Agent – tag each uploaded document type

3. Data Extraction Agent – pull key fields from each document

4. Compliance Validator Agent – compare fields against LC terms & ICC rules

5. Discrepancy Identifier Agent– list discrepancies with justification

6. Decision Agent – recommend accept/reject/waive

7. Report Generator Agent – compile final PDF / JSON report

Replace each TODO section with concrete logic, e.g. OCR calls, regex/LLM parsing, rule‑based checks, etc.

"""

from \_\_future\_\_ import annotations

import json

from pathlib import Path

from typing import Dict, List, Any

from langchain.agents import initialize\_agent, AgentType, AgentExecutor

from langchain.chat\_models import ChatOpenAI

from langchain.schema import SystemMessage, HumanMessage

from langchain.tools import Tool

# -----------------------------------------------------------------------------

# Configuration

# -----------------------------------------------------------------------------

OPENAI\_MODEL = "gpt-4o-mini" # Or your preferred model

TEMPERATURE = 0.0

# -----------------------------------------------------------------------------

# Helper stubs (you will implement these)

# -----------------------------------------------------------------------------

def load\_file\_bytes(file\_path: Path) -> bytes:

"""Utility: read a file and return its bytes (for OCR or base64)."""

return file\_path.read\_bytes()

# 1️⃣ LC Extractor -------------------------------------------------------------

def lc\_extractor(file\_bytes: bytes) -> Dict[str, Any]:

"""Parse LC (MT700 PDF or SWIFT message) and return structured JSON terms.

TODO: implement OCR / LLM extraction / regex depending on input format.

"""

# dummy output

return {

"applicant": "ABC Importers Ltd.",

"beneficiary": "XYZ Exporters Inc.",

"amount": 100000,

"currency": "USD",

"expiry\_date": "2025-12-31",

"latest\_shipment": "2025-11-30",

"req\_documents": [

"Commercial Invoice",

"Bill of Lading",

"Packing List",

"Certificate of Origin"

],

}

# 2️⃣ Document Classifier ------------------------------------------------------

def doc\_classifier(files: List[bytes]) -> Dict[str, List[int]]:

"""Return a mapping: doc\_type -> list of index positions within `files`.

TODO: implement with ML model or LLM few‑shot classification.

"""

# dummy: assume sequential order

return {

"Commercial Invoice": [0],

"Bill of Lading": [1],

"Packing List": [2],

"Certificate of Origin": [3],

}

# 3️⃣ Data Extraction ----------------------------------------------------------

def data\_extractor(classified\_files: Dict[str, List[int]], files: List[bytes]) -> Dict[str, Dict[str, Any]]:

"""Extract required fields from each document and return nested JSON.

TODO: OCR / LLM / rule‑based extraction.

"""

return {

"Commercial Invoice": {

"invoice\_value": 99999,

"invoice\_date": "2025-10-01",

},

"Bill of Lading": {

"port\_of\_loading": "Shanghai",

"port\_of\_discharge": "New York",

"on\_board\_date": "2025-10-15",

},

"Packing List": {},

"Certificate of Origin": {

"country": "CN",

},

}

# 4️⃣ Compliance Validator -----------------------------------------------------

def compliance\_validator(lc\_terms: Dict[str, Any], doc\_data: Dict[str, Dict[str, Any]]) -> Dict[str, Any]:

"""Compare extracted document data with LC terms & ICC rules.

Return a dict containing pass/fail status and raw details.

TODO: encode UCP 600 + ISBP logic.

"""

# dummy compliance example

discrepancies = []

if doc\_data["Commercial Invoice"]["invoice\_value"] > lc\_terms["amount"]:

discrepancies.append({

"doc": "Commercial Invoice",

"field": "invoice\_value",

"issue": "Invoice amount exceeds LC amount",

"rule\_ref": "UCP 600 Art. 18(b)"

})

return {

"is\_compliant": len(discrepancies) == 0,

"discrepancies": discrepancies,

}

# 5️⃣ Discrepancy Identifier ---------------------------------------------------

def discrepancy\_identifier(validation\_result: Dict[str, Any]) -> List[Dict[str, str]]:

"""Produce human‑readable discrepancy list with ICC justification."""

return validation\_result["discrepancies"]

# 6️⃣ Decision Agent -----------------------------------------------------------

LLM\_SYSTEM\_PROMPT = (

"You are a senior LC document checker following UCP 600 and ISBP 821. "

"Given LC terms and discrepancy list, decide whether to Accept, Reject, or "

"Request Waiver. Justify your decision concisely citing ICC rules."

)

def decision\_agent(lc\_terms: Dict[str, Any], discrepancies: List[Dict[str, str]]) -> Dict[str, str]:

llm = ChatOpenAI(model\_name=OPENAI\_MODEL, temperature=TEMPERATURE)

messages = [

SystemMessage(content=LLM\_SYSTEM\_PROMPT),

HumanMessage(content=json.dumps({

"lc\_terms": lc\_terms,

"discrepancies": discrepancies,

}))

]

response = llm(messages)

return {"decision": response.content.strip()}

# 7️⃣ Report Generator ---------------------------------------------------------

def report\_generator(

lc\_terms: Dict[str, Any],

doc\_data: Dict[str, Dict[str, Any]],

discrepancies: List[Dict[str, str]],

decision\_summary: Dict[str, str],

) -> str:

"""Generate a simple JSON report (could be extended to PDF)."""

report = {

"LC Terms": lc\_terms,

"Document Data": doc\_data,

"Discrepancies": discrepancies,

"Decision": decision\_summary,

}

return json.dumps(report, indent=2, ensure\_ascii=False)

# -----------------------------------------------------------------------------

# Wrap Agent Functions as LangChain Tools

# -----------------------------------------------------------------------------

lc\_extractor\_tool = Tool(

name="lc\_extractor",

func=lambda x: lc\_extractor(x),

description="Extract structured LC terms from an LC PDF or MT700 bytes.",

)

doc\_classifier\_tool = Tool(

name="doc\_classifier",

func=lambda x: doc\_classifier(x),

description="Classify list[bytes] into document types required by the LC.",

)

data\_extractor\_tool = Tool(

name="data\_extractor",

func=lambda x: data\_extractor(x["classified\_files"], x["files"]),

description="Extract key trade data fields from classified documents.",

)

compliance\_validator\_tool = Tool(

name="compliance\_validator",

func=lambda x: compliance\_validator(x["lc\_terms"], x["doc\_data"]),

description="Validate documents against LC terms and ICC rules.",

)

discrepancy\_identifier\_tool = Tool(

name="discrepancy\_identifier",

func=lambda x: discrepancy\_identifier(x),

description="Generate discrepancy list with rule justification.",

)

decision\_agent\_tool = Tool(

name="decision\_agent",

func=lambda x: decision\_agent(x["lc\_terms"], x["discrepancies"]),

description="Decide accept/reject/waive based on discrepancies and LC.",

)

report\_generator\_tool = Tool(

name="report\_generator",

func=lambda x: report\_generator(

x["lc\_terms"], x["doc\_data"], x["discrepancies"], x["decision"]

),

description="Compile final LC checking report in JSON.",

)

# Collect tools in execution order (for OPENAI\_FUNCTIONS agent)

TOOLS = [

lc\_extractor\_tool,

doc\_classifier\_tool,

data\_extractor\_tool,

compliance\_validator\_tool,

discrepancy\_identifier\_tool,

decision\_agent\_tool,

report\_generator\_tool,

]

# -----------------------------------------------------------------------------

# Orchestrator / Pipeline Execution

# -----------------------------------------------------------------------------

def run\_pipeline(lc\_file: Path, doc\_files: List[Path]) -> str:

"""High‑level wrapper to run the full LC checking pipeline and return report."""

llm = ChatOpenAI(model\_name=OPENAI\_MODEL, temperature=TEMPERATURE)

agent: AgentExecutor = initialize\_agent(

TOOLS,

llm,

agent=AgentType.OPENAI\_FUNCTIONS,

verbose=True,

)

# 1. Read all files into bytes -------------------------------------------

lc\_bytes = load\_file\_bytes(lc\_file)

docs\_bytes = [load\_file\_bytes(p) for p in doc\_files]

# 2. Agent step 1 – extract LC terms -------------------------------------

lc\_terms = agent.run(tool="lc\_extractor", tool\_input=lc\_bytes)

# 3. Agent step 2 – classify documents -----------------------------------

classified\_files = agent.run(tool="doc\_classifier", tool\_input=docs\_bytes)

# 4. Agent step 3 – data extraction --------------------------------------

doc\_data = agent.run(

tool="data\_extractor",

tool\_input={"classified\_files": classified\_files, "files": docs\_bytes},

)

# 5. Agent step 4 – compliance validation --------------------------------

validation\_result = agent.run(

tool="compliance\_validator",

tool\_input={"lc\_terms": lc\_terms, "doc\_data": doc\_data},

)

# 6. Agent step 5 – discrepancy identification ---------------------------

discrepancies = agent.run(tool="discrepancy\_identifier", tool\_input=validation\_result)

# 7. Agent step 6 – decision agent ---------------------------------------

decision = agent.run(

tool="decision\_agent",

tool\_input={"lc\_terms": lc\_terms, "discrepancies": discrepancies},

)

# 8. Agent step 7 – report generation ------------------------------------

report = agent.run(

tool="report\_generator",

tool\_input={

"lc\_terms": lc\_terms,

"doc\_data": doc\_data,

"discrepancies": discrepancies,

"decision": decision,

},

)

return report

if \_\_name\_\_ == "\_\_main\_\_":

# Example usage ----------------------------------------------------------

SAMPLE\_LC = Path("sample\_mt700.pdf")

SAMPLE\_DOCS = [

Path("invoice.pdf"),

Path("bl.pdf"),

Path("packing\_list.pdf"),

Path("co.pdf"),

]

# For demo purposes, you might stub these files or mock load\_file\_bytes.

final\_report = run\_pipeline(SAMPLE\_LC, SAMPLE\_DOCS)

print("\n===== LC CHECKING REPORT =====\n")

print(final\_report)

**Agent AI SDK Example**

# lc\_doc\_check\_agentsdk.py

"""

Simulated Agentic AI pipeline for LC document checking using \*\*OpenAI Agents SDK\*\*.

The Agents SDK (beta) lets you programmatically define \*\*tool functions\*\* and

invoke them via automatic function‑calling. This skeleton shows how to expose

seven domain‑specific tools, then orchestrate them step‑by‑step to achieve the

same workflow we outlined earlier (LC Extractor → Document Classifier → … →

Report Generator).

Replace the TODO sections with real OCR / ICC rule logic as needed. The goal is

clarity on structure, not full production code.

"""

from \_\_future\_\_ import annotations

import json

import os

from pathlib import Path

from typing import Any, Dict, List

import openai

# -----------------------------------------------------------------------------

# 0. OpenAI client setup -------------------------------------------------------

# -----------------------------------------------------------------------------

openai.api\_key = os.getenv("OPENAI\_API\_KEY", "sk‑…REPLACE\_ME…") # noqa: E501

MODEL = "gpt-4o-mini"

# -----------------------------------------------------------------------------

# 1. DOMAIN TOOL DEFINITIONS ---------------------------------------------------

# -----------------------------------------------------------------------------

# Each tool is a normal Python function. Its signature & docstring become the

# JSON schema that the Agents SDK uses for function‑calling.

# -----------------------------------------------------------------------------

def lc\_extractor(lc\_bytes\_b64: str) -> Dict[str, Any]:

"""Parse an LC (MT700 PDF or Swift message) and return structured terms.

Args:

lc\_bytes\_b64: base64‑encoded binary of the LC document.

Returns:

JSON object containing applicant, beneficiary, amount, currency, expiry,

latest\_shipment, and list of required documents.

"""

# TODO: implement real extraction (OCR + LLM).

return {

"applicant": "ABC Importers Ltd.",

"beneficiary": "XYZ Exporters Inc.",

"amount": 100000,

"currency": "USD",

"expiry\_date": "2025-12-31",

"latest\_shipment": "2025-11-30",

"req\_documents": [

"Commercial Invoice",

"Bill of Lading",

"Packing List",

"Certificate of Origin",

],

}

def doc\_classifier(docs\_b64: List[str]) -> Dict[str, List[int]]:

"""Classify each uploaded document into trade‑doc types.

Args:

docs\_b64: list of base64‑encoded binary docs (order preserved).

Returns:

Mapping of doc\_type → list of indices referring back to docs\_b64.

"""

# TODO: implement ML / LLM classifier.

return {

"Commercial Invoice": [0],

"Bill of Lading": [1],

"Packing List": [2],

"Certificate of Origin": [3],

}

def data\_extractor(classified\_map: Dict[str, List[int]], docs\_b64: List[str]) -> Dict[str, Dict[str, Any]]:

"""Extract key fields from each classified document.

Args:

classified\_map: output of `doc\_classifier`.

docs\_b64: raw files list to extract from.

Returns:

Nested JSON of extracted fields by doc\_type.

"""

# TODO: implement custom logic.

return {

"Commercial Invoice": {

"invoice\_value": 99999,

"invoice\_date": "2025-10-01",

},

"Bill of Lading": {

"port\_of\_loading": "Shanghai",

"port\_of\_discharge": "New York",

"on\_board\_date": "2025-10-15",

},

"Packing List": {},

"Certificate of Origin": {

"country": "CN",

},

}

def compliance\_validator(lc\_terms: Dict[str, Any], doc\_data: Dict[str, Dict[str, Any]]) -> Dict[str, Any]:

"""Validate extracted data against LC terms & ICC rules.

Args:

lc\_terms: structured LC JSON from `lc\_extractor`.

doc\_data: field extractions from `data\_extractor`.

Returns:

Dict with `is\_compliant` bool and `discrepancies` list.

"""

disc = []

if doc\_data["Commercial Invoice"]["invoice\_value"] > lc\_terms["amount"]:

disc.append({

"doc": "Commercial Invoice",

"field": "invoice\_value",

"issue": "Invoice amount exceeds LC amount",

"rule\_ref": "UCP 600 Art. 18(b)",

})

return {"is\_compliant": not disc, "discrepancies": disc}

def discrepancy\_identifier(validation\_result: Dict[str, Any]) -> List[Dict[str, str]]:

"""Return list of discrepancy objects for human review."""

return validation\_result["discrepancies"]

def decision\_agent(lc\_terms: Dict[str, Any], discrepancies: List[Dict[str, str]]) -> Dict[str, str]:

"""Use GPT‑4o to recommend ACCEPT / REJECT / WAIVE and rationale."""

sys = (

"You are an expert trade‑finance document checker following UCP 600 & ISBP. "

"Review LC terms & discrepancy list, then respond with JSON: {'decision': str, 'reason': str}."

)

resp = openai.chat.completions.create(

model=MODEL,

messages=[

{"role": "system", "content": sys},

{"role": "user", "content": json.dumps({

"lc\_terms": lc\_terms,

"discrepancies": discrepancies,

})},

],

temperature=0,

)

return json.loads(resp.choices[0].message.content)

def report\_generator(lc\_terms: Dict[str, Any], doc\_data: Dict[str, Dict[str, Any]], discrepancies: List[Dict[str, str]], decision: Dict[str, str]) -> str:

"""Compile final LC checking report (JSON string)."""

report = {

"LC Terms": lc\_terms,

"Document Data": doc\_data,

"Discrepancies": discrepancies,

"Decision": decision,

}

return json.dumps(report, indent=2, ensure\_ascii=False)

# -----------------------------------------------------------------------------

# 2. TOOL REGISTRATION ---------------------------------------------------------

# -----------------------------------------------------------------------------

TOOLS = [

{

"type": "function",

"function": lc\_extractor,

},

{"type": "function", "function": doc\_classifier},

{"type": "function", "function": data\_extractor},

{"type": "function", "function": compliance\_validator},

{"type": "function", "function": discrepancy\_identifier},

{"type": "function", "function": decision\_agent},

{"type": "function", "function": report\_generator},

]

# -----------------------------------------------------------------------------

# 3. ORCHESTRATION LOGIC -------------------------------------------------------

# -----------------------------------------------------------------------------

# In the Agents SDK you usually drive the chain with a while‑loop handling every

# assistant response. Below is a linear helper (`step\_call`) to simplify.

# -----------------------------------------------------------------------------

def step\_call(function\_name: str, args: Any):

"""Invoke `function\_name` via function‑calling and return parsed JSON."""

chat\_resp = openai.chat.completions.create(

model=MODEL,

messages=[

{

"role": "user",

"content": json.dumps(args),

"tool\_choice": {"name": function\_name},

}

],

tools=TOOLS,

)

tool\_call = chat\_resp.choices[0].message.tool\_calls[0]

return json.loads(tool\_call.function.arguments)

# -----------------------------------------------------------------------------

# 4. PIPELINE WRAPPER ----------------------------------------------------------

# -----------------------------------------------------------------------------

def run\_pipeline(lc\_path: Path, doc\_paths: List[Path]) -> str:

"""Execute the full LC checking pipeline and return final JSON report."""

import base64

# Load & base64‑encode files so they can be passed as JSON safely.

lc\_b64 = base64.b64encode(lc\_path.read\_bytes()).decode()

docs\_b64 = [base64.b64encode(p.read\_bytes()).decode() for p in doc\_paths]

lc\_terms = step\_call("lc\_extractor", lc\_b64)

classified = step\_call("doc\_classifier", docs\_b64)

doc\_data = step\_call("data\_extractor", {"classified\_map": classified, "docs\_b64": docs\_b64})

validation = step\_call("compliance\_validator", {"lc\_terms": lc\_terms, "doc\_data": doc\_data})

discrepancies = step\_call("discrepancy\_identifier", validation)

decision = step\_call("decision\_agent", {"lc\_terms": lc\_terms, "discrepancies": discrepancies})

final\_report = step\_call(

"report\_generator",

{

"lc\_terms": lc\_terms,

"doc\_data": doc\_data,

"discrepancies": discrepancies,

"decision": decision,

},

)

return final\_report

if \_\_name\_\_ == "\_\_main\_\_":

SAMPLE\_LC = Path("sample\_mt700.pdf")

SAMPLE\_DOCS = [

Path("invoice.pdf"),

Path("bl.pdf"),

Path("packing\_list.pdf"),

Path("co.pdf"),

]

report\_json = run\_pipeline(SAMPLE\_LC, SAMPLE\_DOCS)

print("\n========== LC DOCUMENT CHECK REPORT ==========")

print(report\_json)