```
!pip -q install -U \
  langgraph langchain langchain-core langchain-community langchain-text-splitters \ langchain_openai \ langchain-ollama \
  pypdf chromadb tiktoken python-multipart \
  huggingface-hub
```

```
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 67.3/67.3 kB 6.0 MB/s eta 0:00:00
  Installing build dependencies ... done
  Getting requirements to build wheel ... done
  Preparing metadata (pyproject.toml) ... done
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 484.9/484.9 kB 43.3 MB/s eta 0:00:00
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 2.5/2.5 MB 106.8 MB/s eta 0:00:00
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 84.7/84.7 kB 9.2 MB/s eta 0:00:00
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 329.6/329.6 kB 32.1 MB/s eta 0:00:00
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 21.7/21.7 MB 119.6 MB/s eta 0:00:00
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 521.0/521.0 kB 47.4 MB/s eta 0:00:00
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 278.2/278.2 kB 31.3 MB/s eta 0:00:00
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 2.0/2.0 MB 98.8 MB/s eta 0:00:00
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 1.0/1.0 MB 65.5 MB/s eta 0:00:00
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 17.4/17.4 MB 120.4 MB/s eta 0:00:00
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 72.5/72.5 kB 7.9 MB/s eta 0:00:00
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 132.6/132.6 kB 14.2 MB/s eta 0:00:00
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 66.4/66.4 kB 6.9 MB/s eta 0:00:00
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 220.0/220.0 kB 25.2 MB/s eta 0:00:00
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 105.4/105.4 kB 11.4 MB/s eta 0:00:00
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 71.6/71.6 kB 7.9 MB/s eta 0:00:00
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 64.7/64.7 kB 7.3 MB/s eta 0:00:00
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 517.7/517.7 kB 45.1 MB/s eta 0:00:00
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 51.0/51.0 kB 5.5 MB/s eta 0:00:00
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 128.4/128.4 kB 14.8 MB/s eta 0:00:00
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 4.4/4.4 MB 120.7 MB/s eta 0:00:00
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 456.8/456.8 kB 43.9 MB/s eta 0:00:00
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 46.0/46.0 kB 4.5 MB/s eta 0:00:00
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 86.8/86.8 kB 9.6 MB/s eta 0:00:00
  Building wheel for pypika (pyproject.toml) ... done
ERROR: pip's dependency resolver does not currently take into account all the packages that are installed. This behaviour is the source of the following dependency conflicts.
google-colab 1.0.0 requires requests==2.32.4, but you have requests 2.32.5 which is incompatible.
google-adk 1.21.0 requires opentelemetry-api<=1.37.0,>=1.37.0, but you have opentelemetry-api 1.39.1 which is incompatible.
google-adk 1.21.0 requires opentelemetry-sdk<=1.37.0,>=1.37.0, but you have opentelemetry-sdk 1.39.1 which is incompatible.
opentelemetry-exporter-otlp-proto-http 1.37.0 requires opentelemetry-exporter-otlp-proto-common==1.37.0, but you have opentelemetry-exporter-otlp-proto-common 1.39.1 which is incompatible.
opentelemetry-exporter-otlp-proto-http 1.37.0 requires opentelemetry-proto==1.37.0, but you have opentelemetry-proto 1.39.1 which is incompatible.
opentelemetry-exporter-otlp-proto-http 1.37.0 requires opentelemetry-sdk~=1.37.0, but you have opentelemetry-sdk 1.39.1 which is incompatible.
opentelemetry-exporter-gcp-logging 1.11.0a0 requires opentelemetry-sdk<1.39.0,>=1.35.0, but you have opentelemetry-sdk 1.39.1 which is incompatible.
transformers 4.57.3 requires huggingface-hub<1.0,>=0.34.0, but you have huggingface-hub 1.2.3 which is incompatible.
```

```
import os
key = os.environ['OPENAI_API_KEY'] = ''
```

```
from google.colab import drive
drive.mount("/content/drive")
```

```
Mounted at /content/drive
```

```
!nvidia-smi
```

```
Tue Dec 30 08:15:25 2025
+-----------------------------------------------------------------------------------------+
| NVIDIA-SMI 550.54.15              Driver Version: 550.54.15      CUDA Version: 12.4     |
|-----------------------------------------+------------------------+----------------------+
| GPU  Name                 Persistence-M | Bus-Id          Disp.A | Volatile Uncorr. ECC |
| Fan  Temp   Perf          Pwr:Usage/Cap |           Memory-Usage | GPU-Util  Compute M. |
|                                         |                        |               MIG M. |
|=========================================+========================+======================|
|   0  NVIDIA A100-SXM4-80GB          Off |   00000000:00:05.0 Off |                    0 |
| N/A   33C    P0             54W /  400W |       0MiB /  81920MiB |      0%      Default |
|                                         |                        |             Disabled |
+-----------------------------------------+------------------------+----------------------+

+-----------------------------------------------------------------------------------------+
| Processes:                                                                              |
|  GPU   GI   CI        PID   Type   Process name                              GPU Memory |
|        ID   ID                                                               Usage      |
|=========================================================================================|
|  No running processes found                                                             |
```

```
import time
import subprocess
import shutil
import json
import re

!curl -fsSL https://ollama.com/install.sh | sh

p = subprocess.Popen(["ollama", "serve"], stdout=subprocess.DEVNULL, stderr=subprocess.STDOUT)
time.sleep(8)
print("Ollama server PID:", p.pid)

OLLAMA_HOST = "http://localhost:11434"
```

```
>>> Installing ollama to /usr/local
>>> Downloading Linux amd64 bundle
######################################################################## 100.0%
>>> Creating ollama user...
>>> Adding ollama user to video group...
>>> Adding current user to ollama group...
>>> Creating ollama systemd service...
WARNING: systemd is not running
WARNING: Unable to detect NVIDIA/AMD GPU. Install lspci or lshw to automatically detect and install GPU dependencies.
>>> The Ollama API is now available at 127.0.0.1:11434.
>>> Install complete. Run "ollama" from the command line.
Ollama server PID: 1395
```

```
from huggingface_hub import hf_hub_download
import httpx
from pathlib import Path
from string import Template
from typing import Any, Dict, List, Tuple, Optional

MODEL_DIR = Path("/content/models/bielik26")
MODEL_DIR.mkdir(parents=True, exist_ok=True)

REPO_ID = "speakleash/Bielik-11B-v2.6-Instruct-GGUF"
```

```
GGUF_NAME = "Bielik-11B-v2.6-Instruct.Q4_K_M.gguf"

gguf_path = hf_hub_download(
    repo_id=REPO_ID,
    filename=GGUF_NAME,
    local_dir=str(MODEL_DIR),
)
print("Downloaded GGUF:", gguf_path)

modelfile_path = MODEL_DIR / "Bielik-tools.Modelfile"

modelfile_tpl = Template(r'''FROM ./$GGUF

TEMPLATE """<s>{{- if .System }}<|start_header_id|>system<|end_header_id|>

{{ .System }}<|eot_id|>{{- end }}
{{- if .Tools }}<|start_header_id|>system<|end_header_id|>

You have access to the following tools:
{{ .Tools }}

When you need to call a tool, you MUST respond with ONLY this exact format:
<tool_call>
{"name": "function_name", "arguments": {"param": "value"}}
</tool_call>

Do not add any other text when calling a tool. After receiving tool results, provide your final answer.<|eot_id|>{{- end }}
{{- range .Messages }}
{{- if eq .Role "user" }}<|start_header_id|>user<|end_header_id|>

{{ .Content }}<|eot_id|>
{{- else if eq .Role "assistant" }}<|start_header_id|>assistant<|end_header_id|>

{{- if .ToolCalls }}
<tool_call>
{{- range .ToolCalls }}
{"name": "{{ .Function.Name }}", "arguments": {{ .Function.Arguments }}}
{{- end }}
</tool_call>
{{- else }}
{{ .Content }}
{{- end }}<|eot_id|>
{{- else if eq .Role "tool" }}<|start_header_id|>tool<|end_header_id|>

<tool_response>
{{ .Content }}
</tool_response><|eot_id|>
{{- end }}
{{- end }}<|start_header_id|>assistant<|end_header_id|>

"""

PARAMETER stop <|start_header_id|>
```

```
PARAMETER stop <|end_header_id|>
PARAMETER stop <|eot_id|>
PARAMETER temperature 0
''')

modelfile_path.write_text(modelfile_tpl.substitute(GGUF=GGUF_NAME), encoding="utf-8")
print("Wrote Modelfile:", str(modelfile_path))

!ls -lh /content/models/bielik26 | sed -n '1,60p'

# Create model
!cd /content/models/bielik26 && ollama create bielik-tools -f Bielik-tools.Modelfile

!ollama list
!ollama show bielik-tools | sed -n '1,220p'

model = "bielik-tools"
```

```
Bielik-11B-v2.6-Instruct.Q4_K_M.gguf: 100%  6.72G/6.72G [00:24<00:00, 85.4MB/s]
Downloaded GGUF: /content/models/bielik26/Bielik-11B-v2.6-Instruct.Q4_K_M.gguf
Wrote Modelfile: /content/models/bielik26/Bielik-tools.Modelfile
total 6.3G
-rw-r--r-- 1 root root 6.3G Dec 30 08:16 Bielik-11B-v2.6-Instruct.Q4_K_M.gguf
-rw-r--r-- 1 root root 1.3K Dec 30 08:16 Bielik-tools.Modelfile

NAME                  ID              SIZE      MODIFIED
bielik-tools:latest   10c0cb11b4f5    6.7 GB    Less than a second ago
  Model
    architecture        llama
    parameters          11.2B
    context length      32768
    embedding length    4096
    quantization        Q4_K_M

  Capabilities
    completion
    tools

  Parameters
    stop           "<|start_header_id|>"
    stop           "<|end_header_id|>"
    stop           "<|eot_id|>"
    temperature    0
```

```
import shutil
from langchain_openai import OpenAIEmbeddings
from langchain_community.document_loaders import PyPDFLoader
from langchain_community.vectorstores import Chroma
from langchain_text_splitters import RecursiveCharacterTextSplitter
from langchain.tools import tool

emb = OpenAIEmbeddings()

BASE = "/content/drive/MyDrive"
```

```
CHROMA_BASE = "/content/chroma_fado"

REBUILD_INDEX = True
if REBUILD_INDEX:
    shutil.rmtree(CHROMA_BASE, ignore_errors=True)
os.makedirs(CHROMA_BASE, exist_ok=True)

def _build_retriever(
    pdf_path: str,
    collection_name: str,
    persist_dir: str,
    chunk_size: int = 1000,
    chunk_overlap: int = 40,
    k: int = 4,
    lambda_mult: float = 0.5,
    fetch_k: int = 10,
):
    loader = PyPDFLoader(pdf_path, extraction_mode="layout", extract_images=False)
    data = loader.load()
    splitter = RecursiveCharacterTextSplitter(chunk_size=chunk_size, chunk_overlap=chunk_overlap)
    docs = splitter.split_documents(data)

    vs = Chroma.from_documents(
        documents=docs,
        embedding=emb,
        collection_name=collection_name,
        persist_directory=persist_dir,
    )
    return vs.as_retriever(
        search_type="mmr",
        search_kwargs={"k": k, "lambda_mult": lambda_mult, "fetch_k": fetch_k},
    )

general_retriever = _build_retriever(
    f"{BASE}/Case_PRO_1.pdf",
    "fado_general",
    f"{CHROMA_BASE}/general",
    chunk_size=1000, chunk_overlap=40,
    k=2, lambda_mult=0.5, fetch_k=5,
)

operational_retriever = _build_retriever(
    f"{BASE}/Case_PRO_1.pdf",
    "fado_operational",
    f"{CHROMA_BASE}/operational",
    chunk_size=1000, chunk_overlap=40,
    k=2, lambda_mult=0.5, fetch_k=5,
)

financial_retriever = _build_retriever(
    f"{BASE}/9_Baby_AGI/Dane_finansowe.pdf",
    "fado_financial",
    f"{CHROMA_BASE}/financial",
```

```
    chunk_size=1000, chunk_overlap=40,
    k=5, lambda_mult=0.0, fetch_k=5,
)

marketing_retriever = _build_retriever(
    f"{BASE}/9_Baby_AGI/Dane_sprzedażowe_i_marketingowe.pdf",
    "fado_marketing",
    f"{CHROMA_BASE}/marketing",
    chunk_size=1000, chunk_overlap=100,
    k=5, lambda_mult=0.5, fetch_k=10,
)

@tool("general_retriever")
def general_retriever_tool(query: str) -> str:
    """Use this tool whenever someone asks for FADO in general."""
    docs = general_retriever.invoke(query)
    return "\n\n".join(
        f"[{d.metadata.get('source')} | p{d.metadata.get('page')}]\n{d.page_content}"
        for d in docs
    )

@tool("operational_retriever")
def operational_retriever_tool(query: str) -> str:
    """Use this tool whenever someone asks for FADO business processes."""
    docs = operational_retriever.invoke(query)
    return "\n\n".join(
        f"[{d.metadata.get('source')} | p{d.metadata.get('page')}]\n{d.page_content}"
        for d in docs
    )

@tool("financial_retriever")
def financial_retriever_tool(query: str) -> str:
    """Use this tool whenever someone asks for FADO financial situation (including statements)."""
    docs = financial_retriever.invoke(query)
    return "\n\n".join(
        f"[{d.metadata.get('source')} | p{d.metadata.get('page')}]\n{d.page_content}"
        for d in docs
    )

@tool("marketing_retriever")
def marketing_retriever_tool(query: str) -> str:
    """Use this tool whenever someone asks for FADO products, sales, marketing and margins."""
    docs = marketing_retriever.invoke(query)
    return "\n\n".join(
        f"[{d.metadata.get('source')} | p{d.metadata.get('page')}]\n{d.page_content}"
        for d in docs
    )
```

```
from langchain_ollama import ChatOllama
from langchain.agents import create_agent
from langgraph.checkpoint.memory import InMemorySaver

OLLAMA_HOST = "http://localhost:11434"
```

```
MODEL_NAME = "bielik-tools"

llm = ChatOllama(
    model=MODEL_NAME,
    base_url=OLLAMA_HOST,
    temperature=0,
)

SYSTEM_PROMPT = """Jesteś asystentem pracującym na dokumentach firmy FADO.

Cel:
- Odpowiadaj po polsku, rzeczowo i na temat.
- Opieraj się na informacjach z narzędzi (retrieverów) i treści rozmowy.
- Jeśli brakuje danych w kontekście: NAJPIERW użyj narzędzia, dopiero potem odpowiadaj.

Dobór narzędzia:
- general_retriever: ogólne informacje o FADO / przekrojowe pytania
- operational_retriever: procesy, operacje, sposób działania
- financial_retriever: finanse, wyniki, wskaźniki, rachunki
- marketing_retriever: produkty, sprzedaż, marketing, marże

Zasady jakości:
- Nie zgaduj liczb ani faktów. Jeśli nie ma ich w wynikach narzędzi, powiedz wprost czego brakuje.
- Gdy wyniki są niejednoznaczne, podaj 2–3 możliwe interpretacje i wskaż co trzeba doprecyzować.
- Preferuj krótkie punkty + krótkie podsumowanie.
- Nie ujawniaj rozumowania krok-po-kroku ani “przemyśleń”. Pokaż tylko wynik.

Źródła:
- Zawsze podwaj źródło informacji na podstwiw którego formuujesz odpowidz (1. odpowiedzi na bazie contekstu narzędza lub 2. odwpowiedzna bazie wiedzy moelu) !!!

Proces przygotowania odpowiedzi (WEWNĘTRZNY – nie wypisuj kroków):
1) Wygeneruj wstępny szkic odpowiedzi.
2) Zrób krótką refleksję: czy odpowiedź jest kompletna i czy każde zdanie wynika z wyników narzędzi lub kontekstu rozmowy?
   - Jeśli coś jest domysłem albo nie wynika z danych: usuń to lub jasno powiedz „brak danych”.
3) Jeśli można poprawić precyzję: doprecyzuj, ale wyłącznie na podstawie wyników narzędzi i kontekstu rozmowy.
4) Zredaguj odpowiedź końcową.
5) Wypisz WYŁĄCZNIE odpowiedź końcową (bez kroków, bez metakomentarzy, bez „myślenia na głos”).
"""


checkpointer = InMemorySaver()

tools = [
    general_retriever_tool,
    operational_retriever_tool,
    financial_retriever_tool,
    marketing_retriever_tool,
]

agent = create_agent(
    model=llm,
    tools=tools,
    system_prompt=SYSTEM_PROMPT,
```

```
    checkpointer=checkpointer,
)
```

```
config = {"configurable": {"thread_id": "fado-session-1"}}

result = agent.invoke({"messages": [("user", "Jak działa FADO?")]}, config)
print(result["messages"])
print(result["messages"][-1].content)
```

```
[HumanMessage(content='Jak działa FADO?', additional_kwargs={}, response_metadata={}, id='3e7df7b7-efd8-4feb-8a97-fce4fa862cba'), AIMessage(content='', additional_kwargs={}, response_metad
Firma FADO działa na rynku AGD, prowadząc działalność od 5 lat w Polsce i Europie Wschodniej (Rosja, Ukraina, Białoruś). Po wejściu Polski do UE rozszerzyła działania o rynki Europy Zachod

Źródło: [/content/drive/MyDrive/Case_PRO_1.pdf | p2] oraz [/content/drive/MyDrive/Case_PRO_1.pdf | p1].
```

```
config = {"configurable": {"thread_id": "fado-session-2"}}

r1 = agent.invoke({"messages": [{"role": "user", "content": "Na jakich rynkach działa FADO?"}]}, config)
print(r1["messages"][-1].content)

r2 = agent.invoke({"messages": [{"role": "user", "content": "Przyponij o co pyatłem poprzednio?"}]}, config)
print(r2["messages"][-1].content)
```

```
FADO działa na rynku AGD, głównie w Polsce oraz na rynkach Europy Wschodniej (Rosja, Ukraina, Białoruś). Firma rozważała ekspansję na rynki Europy Zachodniej (Niemcy, Włochy, Francja), gdz

Źródła: [1] /content/drive/MyDrive/Case_PRO_1.pdf (strona p1), [2] /content/drive/MyDrive/Case_PRO_1.pdf (strona p2).
Pytałeś, na jakich rynkach działa FADO. Odpowiedź znajduje się powyżej.
```

```
from typing import List, Any
from pydantic import BaseModel, Field
from langchain_core.messages import SystemMessage, HumanMessage, ToolMessage

class ProcessGroup(BaseModel):
    kategoria: str = Field(description="Np. Sprzedaż, Operacje, Obsługa klienta, Finanse, IT")
    procesy: List[str] = Field(description="Lista procesów w tej kategorii")

class FadoProcesses(BaseModel):
    grupy: List[ProcessGroup]
    braki_danych: List[str] = Field(default_factory=list, description="Co jest nieznane / czego brakuje w dokumentach")

tools = [
    general_retriever_tool,
    operational_retriever_tool,
    financial_retriever_tool,
    marketing_retriever_tool,
]

llm_with_tools = llm.bind_tools(tools)

tool_map = {getattr(t, "name", t.__class__.__name__): t for t in tools}

def _stringify_tool_output(out: Any) -> str:
    """Zamienia różne typy zwrotek (Documents/list/dict/str) na czytelny tekst."""
    if out is None:
```

```
    if out is None:
        return ""
    if isinstance(out, list):
        parts = []
        for x in out:
            page = getattr(x, "page_content", None)
            if page is not None:
                parts.append(str(page))
            else:
                parts.append(str(x))
        return "\n\n".join([p for p in parts if p.strip()]).strip()
    return str(out).strip()

EXTRACTION_SYSTEM = """Jesteś asystentem pracującym na dokumentach firmy FADO.

Zasady:
- Jeśli brakuje danych w rozmowie, użyj narzędzi (retrieverów).
- Wykonaj ekstrakcję informacji o procesach biznesowych FADO.
- NIE twórz listy procesów z wiedzy ogólnej. Jeśli czegoś nie ma w wynikach narzędzi, to tego nie dopisuj.
- Możesz użyć wielu narzędzi, jeśli potrzeba.
- Na tym etapie Twoim celem jest ZEBRANIE materiału z narzędzi (tool calls), nie format końcowy.
"""

EXTRACTION_USER = (
    "Zbierz z dokumentów FADO informacje o procesach biznesowych i ich podziale na kategorie "
    "(np. Sprzedaż, Operacje, Obsługa klienta, Finanse, IT). "
    "Jeśli nie masz danych w kontekście rozmowy, wywołaj odpowiednie narzędzia."
)

messages = [
    SystemMessage(content=EXTRACTION_SYSTEM),
    HumanMessage(content=EXTRACTION_USER),
]

for _ in range(3):
    ai = llm_with_tools.invoke(messages)
    messages.append(ai)

    tool_calls = getattr(ai, "tool_calls", None) or []
    if not tool_calls:
        break

    for tc in tool_calls:
        name = tc.get("name")
        args = tc.get("args") or tc.get("arguments") or {}
        tc_id = tc.get("id")

        tool = tool_map.get(name)
        if tool is None:
            messages.append(
                ToolMessage(content=f"Nieznane narzędzie: {name}", tool_call_id=tc_id)
            )
            continue
```

```
        try:
            out = tool.invoke(args)
            out_text = _stringify_tool_output(out)
        except Exception as e:
            out_text = f"Błąd wywołania narzędzia {name}: {e}"

        messages.append(ToolMessage(content=out_text, tool_call_id=tc_id))


tool_context = "\n\n".join(
    [m.content for m in messages if isinstance(m, ToolMessage) and m.content.strip()]
).strip()
print("DEBUG - content z narzędzia", tool_context)

structured_llm = llm.with_structured_output(
    FadoProcesses,
    method="json_schema",
    include_raw=True,
)

STRUCTURE_SYSTEM = """Jesteś asystentem pracującym na dokumentach firmy FADO.

Zasady (twarde):
- Używaj WYŁĄCZNIE informacji z KONTEKSTU (wyniki narzędzi) oraz treści pytania.
- Nie dopowiadaj procesów ani faktów z wiedzy ogólnej.
- Jeśli w kontekście brakuje danych do pogrupowania procesów, wpisz to do braki_danych.
- Zwróć WYŁĄCZNIE JSON zgodny ze schematem (bez komentarzy).
"""

STRUCTURE_USER = f"""Pogrupuj procesy biznesowe FADO na kategorie i zwróć wynik zgodny ze schematem.
Jeśli nie masz danych w kontekście, wpisz je w braki_danych.

KONTEKST (wyniki narzędzi):
{tool_context if tool_context else "BRAK WYNIKÓW Z NARZĘDZI – nie ma danych do ekstrakcji."}
"""

response = structured_llm.invoke([
    SystemMessage(content=STRUCTURE_SYSTEM),
    HumanMessage(content=STRUCTURE_USER),
])

print("\nPARSED:\n", response["parsed"])
print("\nRAW:\n", response["raw"].content)
```

```
DEBUG - content z narzędzia [/content/drive/MyDrive/Case_PRO_1.pdf | p1]
Model biznesowy FADO obejmuje marketing i sprzedaż, rozwój produktów, produkcję, zakupy i logistykę, serwis oraz obsługę techniczną. Model
uzupełniony jest o zarządzanie zasobami ludzkimi, zarządzanie finansami, zrządzanie jakością oraz obsługę informatyczną. Marketing i sprzedaż
obejmuje procesy badania rynku, komunikacji marketingowej, kreowania wizerunku, sprzedaży krajowej oraz sprzedaży eksportowej. Rozwój
produktów obejmuje procesy         zarządzania portfelem produktów oraz rozwoju nowych i modyfikacji istniejących produktów. W ramach produkcji
realizowane są procesy planowania produkcji, przygotowania produkcji oraz wytwarzania. Zakupy obejmują procesy pozyskiwania dostawców
materiałów i surowców oraz planowania zakupów, zamawiania oraz realizacji dostaw surowców i materiałów. W ramach obszaru logistyki funkcjonuje
proces obsługi klienta, proces magazynowania oraz transportu produktów. W ramach obszaru serwisu realizowane są procesy serwisu krajowego

[/content/drive/MyDrive/Case_PRO_1.pdf | p2]
Wszystkie te działania mają na celu zwiększenie wartości wypracowywanej dla właścicieli FADO.
```

```
PARSED:
 grupy=[ProcessGroup(kategoria='Marketing i sprzedaż', procesy=['badanie rynku', 'komunikacja marketingowa', 'kreowanie wizerunku', 'sprzedaż krajowa', 'sprzedaż eksportowa']), ProcessGrou

RAW:
 {"grupy":[{"kategoria":"Marketing i sprzedaż","procesy":["badanie rynku","komunikacja marketingowa","kreowanie wizerunku","sprzedaż krajowa","sprzedaż eksportowa"]},{"kategoria":"Rozwój p
```