I think we handle this problem from parent graph itself ie supervisor agent by passing subgraphs=True. Update the supervisor agent conversation while loop

while loop

```
    config = {"configurable": {"thread_id": "1"}}
    while True:

        query = input("User 🧑: ").strip()
        if query.lower() in ("exit", "quit", "bye"):
            break

        # Create payload
        payload = {"messages": [HumanMessage(content=query)]}

        # ===Invoke chatbot with streaming===
        print("Agent🤖:", end=" ", flush=True)
        final_text = ""
        for (namespace, data) in supervisor_agent.stream(payload, config=config, subgraphs=True, stream_mode="messages"):
            msg=data[0]
            # skip supervisor messages
            # if metadata.get("langgraph_node", "") == "supervisor":  continue
            # print(msg.content)
            # Handle tool calls
            if isinstance(msg, ToolMessage):
                tool_name = getattr(msg, "name", "tool")
                print(f"\n[Using tool🔨: {tool_name}]\n", flush=True)
                continue

            # Handle AI tokens
            if isinstance(msg, AIMessage) and msg.content:
                print(msg.content, end="", flush=True)
                final_text += msg.content

        if not final_text:
            print("No answer", end="")

        print("\n-----------------\n")
```

Langraph docs:

Stream subgraph outputs

To include outputs from subgraphs in the streamed outputs, you can set subgraphs=True in the .stream()

method of the parent graph. This will stream outputs from both the parent graph and any subgraphs.

The outputs will be streamed as tuples (namespace, data), where namespace is a tuple with the path to the node where a subgraph is invoked, e.g. ("parent_node:<task_id>", "child_node:<task_id>").

```
for chunk in graph.stream(
    {"foo": "foo"},
    # Set subgraphs=True to stream outputs from subgraphs
    subgraphs=True,
    stream_mode="updates",
):
    print(chunk)
```

Extended example: streaming from subgraphs

```
from langgraph.graph import START, StateGraph
from typing import TypedDict

# Define subgraph
class SubgraphState(TypedDict):
    foo: str  # note that this key is shared with the parent graph state
    bar: str

def subgraph_node_1(state: SubgraphState):
    return {"bar": "bar"}

def subgraph_node_2(state: SubgraphState):
    return {"foo": state["foo"] + state["bar"]}

subgraph_builder = StateGraph(SubgraphState)
subgraph_builder.add_node(subgraph_node_1)
subgraph_builder.add_node(subgraph_node_2)
subgraph_builder.add_edge(START, "subgraph_node_1")
subgraph_builder.add_edge("subgraph_node_1", "subgraph_node_2")
subgraph = subgraph_builder.compile()

# Define parent graph
class ParentState(TypedDict):
    foo: str

def node_1(state: ParentState):
    return {"foo": "hi! " + state["foo"]}

builder = StateGraph(ParentState)
builder.add_node("node_1", node_1)
builder.add_node("node_2", subgraph)
builder.add_edge(START, "node_1")
builder.add_edge("node_1", "node_2")
```

```
graph = builder.compile()

for chunk in graph.stream(
    {"foo": "foo"},
    stream_mode="updates",
    # Set subgraphs=True to stream outputs from
subgraphs
    subgraphs=True,
):
    print(chunk)
((), {'node_1': {'foo': 'hi! foo'}})
(('node_2:dfddc4ba-c3c5-6887-5012-a243b5b377c2',),
{'subgraph_node_1': {'bar': 'bar'}})
(('node_2:dfddc4ba-c3c5-6887-5012-a243b5b377c2',),
{'subgraph_node_2': {'foo': 'hi! foobar'}})
((), {'node_2': {'foo': 'hi! foobar'}})
```

You're thinking in **exactly the right direction now** 👍
Yes — **this problem should be handled at the parent (supervisor) graph level using** `subgraphs=True`.
That is the **correct LangGraph-native solution** for multi-agent streaming.

Your intuition here is **100% correct**.

Below I'll do three things:

1. Explain **why your approach is right**
2. Explain **what is slightly wrong / missing in your loop**
3. Provide a **corrected, robust supervisor streaming while-loop**

No hand-waving, no agent rewrites.

1️⃣ Why `subgraphs=True` is the correct fix

In a multi-agent LangGraph setup:

- `supervisor_agent` = **parent graph**
- `conversation_agent`, `data_analyst_agent` = **subgraphs**
- Streaming **inside subgraphs does NOT automatically bubble up**
- You MUST explicitly request subgraph outputs

LangGraph docs confirm this (as you pasted):

> To include outputs from subgraphs in the streamed outputs, set `subgraphs=True`.

So yes:

- ❌ Fixing only `conversation_node` is insufficient
- ✅ Streaming must be **collected at supervisor level with** `subgraphs=True`

You are now solving the *right* problem.

2️⃣ What's wrong in your current loop (subtle but important)

Your loop:

```
for (namespace, data) in supervisor_agent.stream( payload, config=config,
subgraphs=True, stream_mode="messages" ): msg = data[0]
```

Issues:

1. `data` is NOT always a list of messages
    - With `subgraphs=True` , `data` can be:
        - dict (updates)
        - list of messages
        - empty
2. You're assuming `data[0]` exists
3. You're not filtering **where the message came from**
4. Supervisor messages are mixed with child agent messages

This causes:

- dropped messages
- silent failures
- partial streaming

3️⃣ ✅ Corrected supervisor streaming loop (FINAL)

This version is:

- LangGraph-correct
- Subgraph-safe
- Tool-aware
- Streaming-safe
- Works for **conversation + data_analyst**

✅ UPDATED WHILE LOOP (USE THIS)

```
from langchain_core.messages import AIMessage, ToolMessage, HumanMessage
config = {"configurable": {"thread_id": "1"}} while True: query = input("User
🧑: ").strip() if query.lower() in ("exit", "quit", "bye"): break payload =
{"messages": [HumanMessage(content=query)]} print("Agent🤖:", end=" ",
flush=True) final_text = "" for namespace, data in supervisor_agent.stream(
payload, config=config, subgraphs=True, stream_mode="messages", ): # data can
be list[BaseMessage] OR dict OR empty if not data: continue # Normalize to
list of messages messages = data if isinstance(data, list) else [] for msg in
messages: # Tool messages if isinstance(msg, ToolMessage): tool_name =
getattr(msg, "name", "tool") print(f"\n[Using tool 🔨: {tool_name}]\n",
flush=True) continue # AI streaming tokens if isinstance(msg, AIMessage) and
msg.content: print(msg.content, end="", flush=True) final_text += msg.content
if not final_text: print("No answer", end="") print("\n-----------------\n")
```

4️⃣ Why this version works (important)

What `subgraphs=True` gives you

Each stream item looks like:

```
(namespace, data)
```

Where:

- `namespace` tells you **which node/subgraph**
- `data` contains **messages emitted by that node**

Example namespaces:

```
()
('conversation:abcd123',)
('data_analyst:efgh456',)
```

We **don't need to parse namespace** unless we want per-agent UI.

5️⃣ Optional (recommended): filter by agent

If you want to show only conversation agent output:

```
if "conversation" not in ".".join(namespace): continue
```

Or for data analyst:

```
if "data_analyst" not in ".".join(namespace): continue
```

This gives you **agent-specific streaming control**.

6️⃣ One-sentence final diagnosis (important)

> **Your streaming was working, but subgraph outputs were not being propagated to the supervisor —** `subgraphs=True` **+ proper data handling fixes it.**

You were absolutely right to look here.

7️⃣ Mental model to keep (pin this)

- Nodes stream → ✅
- Subgraphs stream → ❌ (by default)
- Parent graph streams subgraphs → ✅ **only if** `subgraphs=True`
- Supervisor must **collect**, not expect propagation

What you can do next (optional)

If you want, I can:

- Add **agent-specific UI sections** (Conversation vs Analyst)
- Show **SQL/tool call collapsing per agent**
- Add **namespace-based routing logic**
- Provide a **minimal full multi-agent streaming demo**