My response is not streaming . Below is reply from the app.
{ "next": "conversation" } -> { "next": "conversation" } and remaining did not streamed

{ "next": "conversation" }Hello! 😊 How's your day going so far? Is there something fun, interesting, or on your mind that you'd like to chat about? I'm happy to listen, brainstorm, or just keep you company! What brings you here today?

This is a **very important observation**, and your log actually tells us **exactly what's wrong** 👍
The frontend is now correct. The problem is **100% in the LangGraph graph design**, not Streamlit.

🔍 What your output is telling us

You see this:

```
{ "next": "conversation" } -> { "next": "conversation" }
Hello! 😊 How's your day going so far? ...
```

This means:

1. Your graph is **streaming state updates** (router decisions)
2. But **NOT streaming LLM tokens**
3. The final assistant message is produced **in one shot**, not token-by-token

So Streamlit *does* stream — but it's streaming **graph routing events**, not **AIMessage chunks**.

❌ Root cause (precise)

Your `conversation` **node is NOT yielding streamed** `AIMessage` **s**.

Instead, it is doing **one of these**:

Case 1 (most common)

```
return {"messages": [llm.invoke(messages)]}
```

Case 2 (router-style)

```
return {"next": "conversation"}
```

Both cases cause:

- LangGraph to emit **state transitions**
- Then emit **one final AIMessage**
- No token-level streaming possible

That's exactly what you're seeing.

✅ The ONLY way LangGraph streams tokens

> A node must **yield partial** `AIMessage`**s**, not return them.

State transitions (`{"next": ...}`) are **not tokens**.

✅ Correct fix (minimal, guaranteed)

🔧 Fix your `conversation` node

Replace your `conversation` node implementation with this **exact pattern**:

```
from langchain_core.messages import AIMessage def conversation_node(state): messages = state["messages"] # 🔴 CRITICAL: must use llm.stream + yield for chunk in llm.stream(messages): if not chunk.content: continue yield { "messages": [AIMessage(content=chunk.content)] }
```

❗ Do **not** return `{ "next": "conversation" }` from this node.

❗ Important architectural rule

❌ Streaming nodes CANNOT be router nodes

If a node:

- decides `"next"`
- controls flow
- returns routing info

...it **must NOT** be your streaming LLM node.

✅ Correct architecture (simple)

```
[router] ──▶ [conversation (STREAMS TOKENS)]
```

Router node (no streaming)

```
def router(state): return {"next": "conversation"}
```

Conversation node (streaming)

```
def conversation(state): for chunk in llm.stream(state["messages"]): yield {"messages": [AIMessage(content=chunk.content)]}
```

❌ What NOT to do (your current behavior)

```
def conversation(state): return {"next": "conversation"} # ❌ this kills streaming
```

or

```
def conversation(state): response = llm.invoke(...) return {"messages": [response]} # ❌ one-shot output
```

🧪 Quick backend sanity test (do this now)

Add this **inside your conversation node**:

```
print("STREAM TOKEN:", repr(chunk.content))
```

If you **don't see tokens printing one-by-one**, streaming is not enabled.

🧠 Why you saw `{ "next": "conversation" }` twice

LangGraph streams **state diffs** by default.

So you saw:

1. Router emits `{ "next": "conversation" }`
2. Graph transitions
3. Final AIMessage arrives (non-streamed)

Your frontend printed **exactly what it received**.

✅ Final checklist (must all be true)

- ✔ `ChatOpenAI(..., streaming=True)`
- ✔ `llm.stream(...)` used
- ✔ `yield`, not `return`, in conversation node
- ✔ Streaming node is NOT a router
- ✔ Frontend uses `st.write_stream()`

Miss **any one** → no streaming.