# Samvaad (Sarvam)<> Vistaar (Ekstep) Integration

Samvaad, the conversational platform of Sarvam can handle voice-to-voice calls, rich conversations over WhatsApp, and in-app usecases like embedding within a website or mobile applications.

## Agent Pipeline of Samvaad

In the context of LLM Orchestration, Samvaad provides the following capabilities:

1. LLM Configuration - Sarvam Medium, Sarvam Tota, Sarvam Pro etc.
2. State management - memory (across conversations), thread context (within conversation).
3. Orchestration - tool calling, ReAct loops, and other agentic behaviour.

## BYOLO

To support BYOLO, Samvaad can expose an integration option where users can specify their endpoint configuration for “system of LLMs”, which take care of the above capabilities and Samvaad acts as a channel and normalisation layer to receive user inputs as voice / widgets / text and send the user message to this endpoint. Concretely, the following can be handled by Samvaad:

1. Scheduling (if outbound)
2. Inbound handling with channel integrations
3. ASR + VAD
4. BYOLO instead of Samvaad’s agent pipeline
5. Translation/Transliteration/Post LLM as configured by the author
6. TTS
7. Analytics

## Integration

To standardise based on what's already available in the industry, we can go with the following, which is similar to a single LLM with an external inference setup:

1. OpenAI-compatible API
2. Server-side events (SSE)
3. Authentication (Samvaad <> BYOLO)
4. User ID and Tenant ID
5. Payload structure within the `text` response

```
When you want to say the response to the user:
{
   "audio": "Thank you for confirming."
}

When you need to end the conversation with a sentence:
{
   "end_interaction": true,
   "audio": "..."
}
```

Sample code based on OpenAI Chat Completion

```
from fastapi import FastAPI
from fastapi.responses import StreamingResponse
from pydantic import BaseModel
from typing import List, Optional, Literal
import json
import time

app = FastAPI()
```

```
class Message(BaseModel):
    role: Literal["system", "user", "assistant"]
    content: str


class ChatCompletionRequest(BaseModel):
    model: str = "gpt-3.5-turbo"
    messages: List[Message]
    stream: Optional[bool] = True
    temperature: Optional[float] = 1.0
    max_tokens: Optional[int] = None


def generate_stream():
    """Generate OpenAI-compatible SSE streaming response"""

    # Static response text that will be streamed
    response_text = "Hello! This is a static streaming response from the Fast
API server."

    # Create a unique ID for this completion
    completion_id = f"chatcmpl-{int(time.time())}"
    created_timestamp = int(time.time())

    # Stream each word as a chunk
    words = response_text.split()

    for i, word in enumerate(words):
        chunk = {
            "id": completion_id,
            "object": "chat.completion.chunk",
            "created": created_timestamp,
            "model": "gpt-3.5-turbo",
            "choices": [
                {
                    "index": 0,
                    "delta": {
                        "content": word + " " if i < len(words) - 1 else word
                    },
                    "finish_reason": None
                }
            ]
        }
        yield f"data: {json.dumps(chunk)}\n\n"
        time.sleep(0.1)  # Simulate streaming delay

    # Send final chunk with finish_reason
    final_chunk = {
        "id": completion_id,
        "object": "chat.completion.chunk",
        "created": created_timestamp,
        "model": "gpt-3.5-turbo",
        "choices": [
            {
                "index": 0,
                "delta": {},
                "finish_reason": "stop"
            }
        ]
    }
    yield f"data: {json.dumps(final_chunk)}\n\n"

    # Send [DONE] message
    yield "data: [DONE]\n\n"


@app.post("/v1/chat/completions")
async def chat_completions(request: ChatCompletionRequest):
    """
    OpenAI-compatible chat completions endpoint with streaming support.

    This endpoint accepts messages in the standard OpenAI API format and
    returns a static streamed response.
    """
    if request.stream:
        return StreamingResponse(
            generate_stream(),
            media_type="text/event-stream",
            headers={
                "Cache-Control": "no-cache",
                "Connection": "keep-alive",
            }
        )
    else:
        # Non-streaming response (optional)
        return {
            "id": f"chatcmpl-{int(time.time())}",
            "object": "chat.completion",
            "created": int(time.time()),
            "model": request.model,
            "choices": [
                {
                    "index": 0,
                    "message": {
                        "role": "assistant",
                        "content": "Hello! This is a static response from the FastAPI s
erver."
                    },
                    "finish_reason": "stop"
                }
            ],
            "usage": {
                "prompt_tokens": 10,
                "completion_tokens": 10,
                "total_tokens": 20
            }
        }


@app.get("/")
async def root():
    return {"message": "OpenAI-compatible API server", "endpoint": "/v1/ch
at/completions"}
```

```
if __name__ == "__main__":
    import uvicorn
    uvicorn.run(app, host="0.0.0.0", port=8000)
```

Reference: https://platform.openai.com/docs/api-reference/chat/create

## Out of Scope

1. Language Switch
2. Analytics on Samvaad
3.