You're asking crucial questions for integrating your custom LLM with Deepgram's Voice Agent API! Let's break down the expected request and response formats for https://your-custom-llm-api.com/predict.

**1. What Deepgram will send to https://your-custom-llm-api.com/predict (Request Format):**

Deepgram's Voice Agent generally aims to be compatible with widely used LLM API formats, particularly those similar to OpenAI's Chat Completions API. When Deepgram sends a request to your custom LLM endpoint, it will typically include:

* **HTTP Method:** POST
* **Content-Type:** application/json

**Request Body (JSON):** The core of the request will be a JSON object containing the conversational history and the latest user utterance. This usually takes the form of a messages array, where each element represents a turn in the conversation.  
  
 Here's a common structure you can expect:  
  
 JSON  
{

"messages": [

{

"role": "system",

"content": "You are a helpful assistant." // This will be your `agent.prompt` or `system\_prompt` from your Deepgram agent configuration.

},

{

"role": "user",

"content": "Hello, how are you?" // The transcribed text from the user.

},

// ... potentially more messages for conversational history

],

"model": "your\_custom\_model\_name", // If you specified a model name in your Deepgram agent config.

"temperature": 0.7, // If you specified a temperature in your Deepgram agent config.

"stream": false, // Deepgram might default to non-streaming for custom endpoints, but check if it supports streaming.

// ... potentially other parameters Deepgram might send based on its internal logic or your configuration.

}

* **Key points about the request:**
  + **messages array:** This is the most important part. It represents the ongoing conversation.
    - role: Can be "system", "user", or "assistant".
    - content: The actual text of the message.
  + **system role:** The system message typically contains the initial prompt or instructions you've given to your Deepgram Agent (e.g., "You are a friendly customer service bot.").
  + **user role:** This is where Deepgram will put the latest transcribed text from the user.
  + **Conversational History:** Deepgram will automatically manage and send the conversational history to your LLM based on its internal state and how you configure the agent.

**2. What https://your-custom-llm-api.com/predict has to send back (Response Format):**

Your custom LLM's API endpoint needs to return a response that Deepgram can understand and use for Text-to-Speech (TTS). The most reliable and commonly expected format is one that mirrors the **OpenAI Chat Completions API response format**.

Here's the general structure your LLM should send back:

JSON

{

"id": "chatcmpl-unique\_id\_for\_this\_response", // A unique ID for the completion

"object": "chat.completion",

"created": 1677652288, // Unix timestamp of when the response was created

"model": "your\_custom\_model\_name", // The name of the model that generated the response

"usage": {

"prompt\_tokens": 10, // Number of tokens in the input prompt

"completion\_tokens": 20, // Number of tokens in the generated completion

"total\_tokens": 30 // Total number of tokens

},

"choices": [

{

"index": 0,

"message": {

"role": "assistant",

"content": "Hello! I am doing well, thank you for asking. How can I assist you today?" // Your LLM's generated response

},

"finish\_reason": "stop" // "stop", "length", "function\_call", etc.

}

]

}

**Key points about the response:**

* **object: "chat.completion"**: This is important for Deepgram to recognize the format.
* **choices array:** This array contains the generated responses. For most single-turn interactions, it will have one element.
* **choices[0].message.role: "assistant"**: Your LLM's response should always be from the "assistant" role.
* **choices[0].message.content**: This is the crucial part – the actual text your LLM generates as its response. Deepgram will take this text and send it to its Text-to-Speech (TTS) engine.
* **finish\_reason**: While Deepgram might not strictly require all usage or finish\_reason fields, it's good practice to include them for compatibility and debugging. "stop" is common for natural conversational turns.

**Important Considerations for your Custom LLM API:**

* **API Endpoint:** Ensure your https://your-custom-llm-api.com/predict endpoint is set up to handle POST requests and correctly parse the JSON body Deepgram sends.
* **Speed:** Latency is critical for real-time voice agents. Your LLM needs to respond as quickly as possible. Optimize your LLM for fast inference.
* **Concurrency:** Your LLM API should be able to handle multiple concurrent requests from Deepgram if you anticipate multiple users or agents.
* **Error Handling:** Implement proper error handling in your LLM API. If your LLM encounters an error, it should return an appropriate HTTP status code (e.g., 500 for internal server error) and a clear error message in the response body.
* **Authentication/Authorization:** If your LLM API requires authentication (highly recommended!), make sure your Deepgram Agent configuration includes the necessary headers (e.g., Authorization: Bearer YOUR\_LLM\_API\_KEY).
* **Streaming Responses (Advanced):** While the above example is for a single, complete response, some LLMs support streaming responses (where text is sent token by token). If your LLM supports this and you want to leverage it for faster perceived responses, you'll need to check Deepgram's documentation for how it handles streamed LLM outputs. Deepgram's general Voice Agent API can handle streaming, and it would typically involve sending partial JSON objects that represent chunks of the content. However, for initial integration, a single complete response is simpler.

By adhering to the OpenAI Chat Completions API format for both request and response, you'll maximize compatibility and simplify the integration of your custom LLM with Deepgram's Voice Agent.