Handling SMS Queries in n8n: Best Practices for Routing & LLM Integration

Overview: Twilio SMS in n8n Workflows

Integrating SMS with n8n is typically done using the Twilio node (or a Webhook set as a Twilio SMS callback). This allows incoming text messages to trigger your workflow. In fact, n8n provides ready-made templates for SMS-based chatbots. For example, an AI-powered SMS assistant template uses a Twilio trigger to receive messages, passes the text to an AI agent (which can query a database), and then sends an SMS reply via Twilio . This shows how powerful workflows can be built without a lot of custom code – the Twilio node handles inbound/outbound SMS, and the logic in between can be visual or AI-driven.

Key point: All your users' queries will originate from SMS (Twilio), and you want to determine how to process each query. The main challenge is designing the first step in the workflow (right after the SMS is received) that decides where to route the query without resorting to an unwieldy Switch/Case node with dozens of conditions. Below, we'll explore the types of queries and several approaches to handle this routing in a clean, maintainable way.

Types of SMS Queries: "Dashboard" vs. "Chat"

From your description, the queries fall into two broad categories:

- "Dashboard" style queries: These are short SMS requests asking for some factual or analytical data – for example, "How many customer requests today were classified as angry vs. praise?". Here the answer might be a count or summary drawn from your system's data (perhaps leveraging prior LLM-based sentiment classifications). In essence, the user is querying a statistic or status, akin to a mini-dashboard delivered via text.

- "Chat" or knowledge queries: These are more open-ended questions where a Large Language Model (LLM) will generate the answer. For instance, a user might ask, "What's our refund policy?" or "Can I change my appointment time?". These require understanding the query and possibly fetching info from a knowledge base or just using the LLM's trained knowledge to respond conversationally.

All queries are "LLM-powered" in the sense that an AI is involved in processing the user's input or formulating the answer. However, the workflow for each type differs. A dashboard query likely needs to retrieve structured data (counts, lists, etc.) from a database or API, whereas a general

chat query might be handled by the LLM alone (possibly with retrieval augmentation for company-specific info).

Avoiding a Giant Switch/Case: Routing Strategies

In n8n, the traditional way to handle conditional logic is using the Switch node or multiple IF nodes. The Switch node supports multiple output branches and is basically a multi-case conditional (similar to a series of IFs) . While you could create a Switch with dozens of rules (each checking if the SMS text contains a certain keyword or pattern), this quickly becomes hard to manage. Fortunately, the community and new n8n features offer better approaches:

1.

Switch Node with Expression (for few cases)

If the number of query categories is small, you can still use a Switch but in a smarter way. The Switch node has an “Expression” mode where you write a snippet of code to output an index (number) for which branch to take . For example, you might write an expression that examines the incoming SMS text and returns 0 for a dashboard query, 1 for a chat query, etc. This way, instead of manually listing many rules, you have one concise expression that decides the path. You’d then have (say) two outputs from the Switch: one leading to a sub-workflow or nodes handling the data query, and the other leading to the LLM chat response flow.

This approach keeps the workflow visual but avoids dozens of separate nodes. It works well if you can clearly distinguish a few categories of intent. However, if you truly had “dozens and dozens” of distinct intents, even an expression-based Switch might get complicated (since you’d then map many possible values to many outputs). In that case, consider the next approaches.

2.

Code Node for Routing Logic

Many n8n users leverage the Code node (JavaScript) to handle complex logic that would be cumbersome with multiple nodes. You can write a JavaScript function to parse the SMS text and determine what action or path to take. For instance, your code could check for certain keywords or patterns (using regex or string matching) and then set an output field like item.action = "getSentimentStats" or item.action = "answerQuestion". Based on that, you could use a simpler Switch (with just a few cases) or a series of IF nodes that read the action field.

Using a code node essentially shifts the complexity into code rather than the visual workflow. The benefit is that all the decision logic is in one place (easier to update than editing many node conditions). The downside is that debugging might be harder if the code fails silently, and it somewhat goes against the “low-code” philosophy. Still, for those comfortable with JavaScript, this is a robust technique. It’s common in the community to use code for parsing inputs when

flexibility is needed. Just remember to handle errors and perhaps log or output what decision was made for traceability.

Example: Suppose an SMS query starts with the word “Count” or “How many”. Your code node can detect that and interpret it as a dashboard query for statistics. It might further parse the text to see if it mentions “praise” or “angry”, etc., and then decide which specific metric to fetch. If it doesn’t match any known pattern, the code could default to treating it as a general chat question.

3.

LLM-Based Classification (Dynamic Intent Detection)

Instead of manually maintaining a list of keywords, you can use an LLM to classify the query’s intent. n8n’s AI capabilities (via LangChain integration) allow you to do things like prompt an LLM or use a Text Classifier node to output a category label for the query. For example, the first node after receiving the SMS could be a “Text Classifier” (or a Basic LLM Chain with a custom prompt) that reads the user’s question and outputs something like: {"intent": "DashboardQuery"} or {"intent": "GeneralQuery"} (or even more granular intents like “SentimentStats”, “OrderStatus”, “SmallTalk”, etc. depending on your needs).

This technique leverages AI to figure out the intent, so you don’t have to enumerate every possibility up front. A real-world example of this pattern is shown in n8n’s blog on Adaptive RAG: an initial AI agent is used solely to classify the user’s question, and then a Switch routes to one of four strategies based on that classification . In their case they categorized queries as Factual, Analytical, Opinion, or Contextual and branched accordingly, with each branch using a different retrieval/answering approach .

Example of using an AI classifier + Switch in an n8n workflow: The first AI agent determines the query type (here one of four categories), and a Switch node then routes the workflow into different handling paths . This avoids hard-coding dozens of rules by letting the AI do intent detection.

For your scenario, you might have the AI classify queries into just two or three buckets initially (e.g. “MetricQuery” vs “GeneralChat” – and possibly a third like “Unrecognized” or “Other”). With that result, you’d use a small Switch or IF to go down the appropriate path. This dramatically reduces the complexity of the first node. The LLM can be instructed with a prompt like: “Read the SMS and categorize it as one of: (A) DashboardQuery – asking for stats/counts, or (B) ChatQuery – asking a general question or support inquiry. Output only the category name.” The new Structured Output Parser features in n8n’s AI nodes can enforce a JSON output for easy parsing .

Community insight: Many users are experimenting with AI agents for routing. One user on the n8n forums described an AI agent that routes incoming queries to the appropriate tool based on intent – effectively letting the AI choose between different actions without needing separate

sub-workflows for each . This kind of setup suggests that relying on LLMs for decision-making is becoming a best practice to keep workflows lean when the logic is complex.

4.

LLM “Agent” with Tools (Minimize Explicit Branching)

Going a step further, n8n now supports advanced AI Agent nodes (built on LangChain) – including an OpenAI Functions Agent and a Tools Agent. These allow the LLM to not only classify intent but actually call defined “functions” or tools within the workflow. In practice, you can give the AI a list of possible actions it can take (each action tied to a function or sub-workflow). The LLM will decide which (if any) function to call based on the user’s input, and n8n will execute that. This approach can eliminate the Switch node entirely: the AI essentially is the first node’s logic.

For example, you might define two tools for the agent: one tool fetches the sentiment stats from your database, and another tool invokes a knowledge-base answer (or just returns a direct LLM answer). The OpenAI Functions Agent is perfect for this – you describe something like: function getSentimentStats(category) -> returns the count of requests of that category today and function answerGeneralQuery(query) -> uses AI to answer a general question. The LLM will receive the user’s SMS and choose the appropriate function (with parameters) if it fits one of those patterns. Under the hood, it’s using OpenAI’s function calling capability. No giant switch-case needed – the branching is learned by the AI from your descriptions .

The AI-powered student assistant template we mentioned earlier is a good illustration: it uses an AI agent that has access to a course database tool (Airtable) and the agent autonomously figures out which tool to use and what query to run, based on the question . The entire flow (interpreting the question, deciding on a DB lookup, performing it, then responding) is handled within the agent’s logic, not by a hardcoded series of Switch nodes. This design is very flexible to new query types – if you later add more tools (say, a CRM lookup or a different dataset), the same agent can leverage them without restructuring your workflow, aside from providing the new tool definition .

Note: Designing an agent requires careful prompt and tool setup, and testing to be sure it chooses correctly. But it’s arguably the most elegant solution when you have many possible user requests. It offloads decision-making to the LLM, which beyond just classification can handle multi-step reasoning. For instance, an agent could decide a query needs two steps (“first get the stat, then compose a message combining it with a standard answer”) all on its own. This is similar to how AI “function calling” or ReAct frameworks work, and n8n supports these patterns out of the box.

5.

Modular Sub-Workflows

As a complement to the above strategies, consider breaking your solution into modular workflows rather than one monolithic flow with dozens of nodes. You can have the main workflow do the initial parsing or classification (via one of the methods above) and then use an Execute Workflow node to call a sub-workflow for the specific task. For example, if the query is categorized as a dashboard query, execute the “SentimentStatsWorkflow” that contains all the steps to query the database and format the response. If it’s a general chat query, execute the “AnswerQueryWorkflow” that might call the OpenAI API with the query and any context.

This way, each workflow remains focused and easier to manage. It’s more of a design pattern than a different routing mechanism, but it helps avoid a single canvas crowded with nodes for every scenario. The community often uses sub-workflows for reuse and clarity, though as one forum discussion noted, for very simple LLM prompt calls it can feel like overkill to create separate workflows . Use sub-workflows judiciously – they’re great for organization and reusability, but for extremely small “one-node” branches, you might keep it inline. (Recent versions of n8n even introduced a Model Switcher node to help switch between AI models or prompts without separate workflows .)

Code vs. Visual Logic: Community Best Practices

Are these nodes usually coded or designed visually? There is no single “community consensus” – it really depends on the creator’s comfort and the complexity of the logic:

- Low-code (visual) approach: n8n is built to allow complex logic with nodes like Switch, IF, Merge, etc., so many creators try to use those as much as possible. They are easier for others (or your future self) to read at a glance. Techniques like stacking a few IF nodes or a Switch with a handful of branches are common for simpler routing. You mentioned not wanting huge switch/case blocks, which is wise – beyond a certain size, it’s neither robust nor readable. But a small number of clear conditional nodes is perfectly fine and often preferred for maintainability . If you can condense your logic into a manageable set of conditions (especially with the help of AI classification), the visual route is quite robust.

- Code-oriented approach: When logic gets very intricate, many in the community do turn to the Code node or even develop custom nodes. Your inclination towards coding is shared by those who want precise control or find it faster to implement complex parsing in code. Just keep in mind the advice from experienced n8n users: if you use a Code node, handle errors and maybe add logging (even if just to the console) because debugging a silent failure inside code can be tricky . Also, comment your code for clarity. The good news is that n8n’s fair-code nature means you can mix and match – use code where it makes sense and nodes elsewhere.

- AI/ML approach: The rise of integrated AI in n8n (LangChain, etc.) has introduced a semi-coded paradigm: you “prompt” the logic rather than hard-code it. This is becoming a best practice for natural language heavy workflows. The community templates and examples in 2024–2025 (like AI chatbots, support agents, etc.) heavily use the AI Agent

nodes and chains . These are designed on the canvas (so in that sense it's a visual, no-code configuration), but the decision-making is done by the AI model at runtime. It's a shift from explicit programming to guiding the model with prompts and letting it decide the path. If you're new to n8n but familiar with LLMs, this might actually be an accessible route: you configure the agent with a few clicks and prompts instead of writing a lot of JavaScript or creating 20 Switch cases.

In summary, don't be afraid to combine approaches. For example, you might use an LLM to classify the query (visual AI node), then a simple Switch (visual) to call one of two sub-workflows, where one sub-workflow is mostly code to query your database, and the other sub-workflow calls an AI model for a chat answer. This hybrid approach leverages the strength of each method.

Additional Tips for SMS Query Workflows

- Designing the "First Node": Aim for the first part of your workflow to reduce the problem, not solve everything. Whether it's a code function or an AI classifier, its job is just to figure out what kind of query this is. The actual answering logic comes after. This separation (detect intent → handle intent) is a proven robust design (it's essentially the classical intent detection then fulfillment model used in chatbots). It prevents you from writing one-off logic for every phrasing of a question.

- Data for Dashboard Queries: Ensure you have a reliable data source for those metrics. If your LLM is classifying customer requests by sentiment, consider storing counts (perhaps in a database or even a Google Sheet) each time a request comes in. Then the "dashboard query" workflow can simply read the latest counts. You likely don't want the LLM to scan raw text each time for counts (that would be slow/expensive). Instead, use n8n to update a counter or record whenever a request is classified. Then an incoming SMS like "How many angry tickets today?" triggers a DB query node that returns the number. This makes the workflow's job easier – basically a parameterized lookup.

- Context for Chat Queries: If all chat queries are answered by an LLM, consider how the LLM gets its knowledge. For general FAQs, you might bake some info into the prompt (system message with company policies, etc.), or use a Retrieval-Augmented Generation (RAG) approach where the workflow fetches relevant info (from documents, knowledge base, etc.) based on the question and feeds it into the LLM. N8n has nodes for vector databases and such if you go that route . If it's a small domain, a simple prompt with key Q&A or a few if/else to handle extremely common queries might suffice. The main point: ensure the LLM has the data it needs, because unlike dashboard queries, it might not call a tool unless you set that up. In an OpenAI Functions Agent scenario, one "function" could be a vector search or database query for info.

- Iterate and Involve the Community: Since you are new to n8n, start with a basic version of this workflow (perhaps using a simple IF/Switch or a code node) and get it working for a couple of example queries. n8n's execution logs and debugging will help you see the flow. You can then gradually replace or enhance parts (e.g. introduce the LLM classifier once basics are in place). The n8n community forum is very active – if you hit specific roadblocks, you can ask how others would handle it. Often, they might share snippet examples or even their own workflow JSON for similar problems.

Conclusion

Bottom line: You do not need a monstrous switch/case block to handle a variety of SMS queries in n8n. Best practices include using AI-based intent recognition or a bit of JavaScript logic to route queries into a few high-level paths. From there, you can either rely on modular sub-workflows or powerful LLM agents to get the answer. The community has embraced solutions like AI Agents that choose tools based on the query (essentially letting the workflow design itself at runtime) , as well as more traditional clean coding techniques.

For your scenario, a recommended approach is: Twilio Trigger → (AI classification or Code parsing) → minimal Switch (or direct function call) → handling branch. One branch would gather the requested metric(s) and respond, the other would feed the query to an LLM for a answer, and then you'd unify by sending the SMS response back via Twilio. This structure will be far easier to maintain than dozens of hardcoded cases, and it's aligned with how many n8n users build intelligent chatbots today. By balancing n8n's visual logic with the power of LLMs and/or code where appropriate, you'll create a robust SMS workflow that can handle a wide range of queries with ease.

Sources:

- n8n Workflow Template – AI-powered Student Assistant for Course Info via Twilio SMS (example of using Twilio + AI agent to answer SMS queries)

- n8n Docs – Switch node (explains multi-route switching and expression mode)

- n8n Blog – Agentic RAG (Adaptive Retrieval) (illustrates using an AI to classify query intent, then a Switch to route strategies)

- n8n Community Forum – AI agent routing by intent (community member describing an agent that chooses tools based on query intent)