```python
# !pip install groq python-dotenv

from groq import Groq

# ▯ Set your Groq API Key directly here
GROQ_API_KEY = "MY GROQ API KEY"

# Initialize Groq client
client = Groq(api_key=GROQ_API_KEY)

MODEL_NAME = "llama-3.1-8b-instant"

MODEL_CONFIG = {
    "technical": {
        "system_prompt": """You are a Technical Support Expert.
You are precise, code-focused, and analytical.
Provide debugging steps, explain errors clearly, and include example
code fixes when relevant.
Be concise and technically rigorous."""
    },

    "billing": {
        "system_prompt": """You are a Billing Support Expert.
You are empathetic, professional, and policy-driven.
Explain billing issues clearly, mention refund policies, and guide the
user step-by-step.
Maintain a calm and supportive tone."""
    },

    "general": {
        "system_prompt": """You are a General Customer Support
Assistant.
You handle casual conversations and general inquiries.
Be friendly, helpful, and concise."""
    }
}

def route_prompt(user_input: str) -> str:
    routing_prompt = f"""
Classify the following user query into one of these categories:
[technical, billing, general]

Return ONLY the category name.

User Query:
{user_input}
"""

    response = client.chat.completions.create(
        model=MODEL_NAME,
        messages=[
            {"role": "system", "content": "You are a strict intent classifier."},
            {"role": "user", "content": routing_prompt}
        ],
        temperature=0  # Deterministic output
    )

    category = response.choices[0].message.content.strip().lower()

    # Safety fallback
    if category not in MODEL_CONFIG:
        category = "general"

    return category

def process_request(user_input: str) -> str:

    # Step 1: Route to correct expert
    category = route_prompt(user_input)

    print(f"Routed to: {category.upper()} expert\n")

    # Step 2: Load system prompt
    system_prompt = MODEL_CONFIG[category]["system_prompt"]

    # Step 3: Call expert model
    response = client.chat.completions.create(
        model=MODEL_NAME,
        messages=[
            {"role": "system", "content": system_prompt},
            {"role": "user", "content": user_input}
        ],
        temperature=0.7  # Creative but controlled
    )

    return response.choices[0].message.content

query1 = "My python script is throwing an IndexError on line 5."
print(process_request(query1))
```

```
Routed to: TECHNICAL expert

I'd be happy to help you debug the issue.

To provide a more accurate solution, I'll need more information about
your script. Please provide the following:

1. The full error message, including any stack traces.
2. The code snippet that is causing the error (lines 1-10 would be
great).
```

~~~
3. The data you are working with (e.g., lists, dictionaries, etc.).

However, I can provide some general guidance on how to approach this
issue:

**Common causes of IndexError:**

1. Out-of-bounds indexing: You are trying to access an element at an
index that is greater than or equal to the length of the list.
2. Incorrect data structure: You are trying to access an element in a
data structure that does not support indexing (e.g., a set).

**Debugging steps:**

1. Check the length of the list or data structure: Use the `len()`
function to verify that the list has at least the number of elements
you are trying to access.
2. Verify the indexing: Double-check that you are using the correct
index (remember that indices start at 0 in Python).
3. Use a debugger or print statements: Add print statements or use a
debugger to inspect the values of variables and data structures.

**Example code fix:**

Suppose you have a list `my_list` and you're trying to access the 5th
element:

```python
my_list = [1, 2, 3]
index = 5

try:
    print(my_list[index])
except IndexError:
    print("IndexError: Index out of bounds")
```

To fix the error, you can either:

* Use a valid index: `index = 2`
* Add a bounds check: `if index < len(my_list): ...`

Please provide more information about your script, and I'll be happy
to help you debug the IndexError.
~~~

```python
query2 = "I was charged twice for my subscription this month."
print(process_request(query2))
```

```
Routed to: BILLING expert

I'm here to help you resolve the issue. I understand how frustrating
```

```
it can be to receive duplicate charges. Can you please provide me with
some additional details so I can assist you further?

Could you kindly confirm the following:

1. The date you first noticed the duplicate charge?
2. The amount charged for each instance?
3. Your subscription plan and type (e.g., monthly, annual, etc.)?
4. Your account login credentials or account number, so I can look
into this further?

Once I have this information, I'll guide you through the process to
resolve the issue and potentially provide a refund if eligible.

Please know that I'm committed to assisting you in a timely and
efficient manner.
```

```python
query3 = "What services does your company offer?"
print(process_request(query3))
```

```
Routed to: GENERAL expert

We're happy to have you here. Our company offers a wide range of
services, but I'd be happy to give you a brief overview.

We provide e-commerce solutions, including online shopping platforms,
payment processing, and logistics management. We also offer digital
marketing services, such as social media management, search engine
optimization (SEO), and email marketing.

Additionally, we offer cloud computing services, including storage
solutions, software as a service (SaaS), and infrastructure as a
service (IaaS). Our team of experts can help you set up and manage
your cloud infrastructure, ensuring that your data is secure and
easily accessible.

We also have a customer support team, like myself, that's here to help
with any questions or issues you may have. Whether you're a business
owner or an individual, we're here to support you.

Would you like to know more about any of these services or is there
something specific you're looking for?
```

```python
MODEL_CONFIG["tool"] = {
    "system_prompt": "You are a tool routing assistant."
}

def fetch_bitcoin_price():
    # Mock API response
    return "The current price of Bitcoin is $65,000 (mock data)."
```

```python
def route_prompt(user_input: str) -> str:
    routing_prompt = f"""
Classify the following user query into one of these categories:
[technical, billing, general, tool]

Use 'tool' if the user is asking for real-time data like prices.

Return ONLY the category name.

User Query:
{user_input}
"""

    response = client.chat.completions.create(
        model=MODEL_NAME,
        messages=[
            {"role": "system", "content": "You are a strict intent 
classifier."},
            {"role": "user", "content": routing_prompt}
        ],
        temperature=0
    )

    category = response.choices[0].message.content.strip().lower()

    if category not in MODEL_CONFIG:
        category = "general"

    return category

def process_request(user_input: str) -> str:

    category = route_prompt(user_input)

    print(f"Routed to: {category.upper()} expert\n")

    if category == "tool":
        return fetch_bitcoin_price()

    system_prompt = MODEL_CONFIG[category]["system_prompt"]

    response = client.chat.completions.create(
        model=MODEL_NAME,
        messages=[
            {"role": "system", "content": system_prompt},
            {"role": "user", "content": user_input}
        ],
        temperature=0.7
    )
    return response.choices[0].message.content

query4 = "What is the current price of Bitcoin?"
print(process_request(query4))
```

```
Routed to: TOOL expert

The current price of Bitcoin is $65,000 (mock data).
```