Add fallbacks

There are many possible points of failure in an LLM application, whether that be issues with LLM API's, poor model outputs, issues with other integrations, etc. Fallbacks help you gracefully handle and isolate these issues.

Crucially, fallbacks can be applied not only on the LLM level but on the whole runnable level.

Handling LLM API Errors

This is maybe the most common use case for fallbacks. A request to an LLM API can fail for a variety of reasons - the API could be down, you could have hit rate limits, any number of things. Therefore, using fallbacks can help protect against these types of things.

IMPORTANT: By default, a lot of the LLM wrappers catch errors and retry. You will most likely want to turn those off when working with fallbacks. Otherwise the first wrapper will keep on retrying and not failing.

```
%pip install --upgrade --quiet  langchain
langchain-openai
```

```
from langchain_community.chat_models import
ChatAnthropic
from langchain_openai import ChatOpenAI
```

First, let's mock out what happens if we hit a RateLimitError from OpenAI

```
from unittest.mock import patch

import httpx
from openai import RateLimitError

request = httpx.Request("GET", "/")
response = httpx.Response(200,
request=request)
error = RateLimitError("rate limit",
response=response, body="")
```

```
# Note that we set max_retries = 0 to avoid
retrying on RateLimits, etc
openai_llm = ChatOpenAI(max_retries=0)
anthropic_llm = ChatAnthropic()
```

```
llm =
openai_llm.with_fallbacks([anthropic_llm])
```

```
# Let's use just the OpenAI LLm first, to show
into an error
with
patch("openai.resources.chat.completions.Comple
side_effect=error):
    try:
        print(openai_llm.invoke("Why did the ch
the road?"))
    except RateLimitError:
        print("Hit error")
```

```
Hit error
```

```
# Now let's try with fallbacks to Anthropic
with
patch("openai.resources.chat.completions.Comple
side_effect=error):
    try:
        print(llm.invoke("Why did the chicken c
road?"))
    except RateLimitError:
        print("Hit error")
```

```
content=' I don\'t actually know why the
chicken crossed the road, but here are some
possible humorous answers:\n\n- To get to the
other side!\n\n- It was too chicken to just
stand there. \n\n- It wanted a change of
scenery.\n\n- It wanted to show the possum it
could be done.\n\n- It was on its way to a
poultry farmers\' convention.\n\nThe joke
plays on the double meaning of "the other
side" - literally crossing the road to the
other side, or the "other side" meaning the
afterlife. So it\'s an anti-joke, with a
silly or unexpected pun as the answer.'
additional_kwargs={} example=False
```

We can use our “LLM with Fallbacks” as we would a normal LLM.

```
from langchain_core.prompts import ChatPromptTe

prompt = ChatPromptTemplate.from_messages(
    [
        (
            "system",
            "You're a nice assistant who always
compliment in your response",
        ),
        ("human", "Why did the {animal} cross t
    ]
```

```
)
chain = prompt | llm
with
patch("openai.resources.chat.completions.Comple
side_effect=error):
    try:
        print(chain.invoke({"animal": "kangaroo
    except RateLimitError:
        print("Hit error")
```

```
content=" I don't actually know why the
kangaroo crossed the road, but I'm happy to
take a guess! Maybe the kangaroo was trying
to get to the other side to find some tasty
grass to eat. Or maybe it was trying to get
away from a predator or other danger.
Kangaroos do need to cross roads and other
open areas sometimes as part of their normal
activities. Whatever the reason, I'm sure the
kangaroo looked both ways before hopping
across!" additional_kwargs={} example=False
```

Specifying errors to handle

We can also specify the errors to handle if we want to be more specific about when the fallback is invoked:

```
llm = openai_llm.with_fallbacks(
    [anthropic_llm], exceptions_to_handle=
(KeyboardInterrupt,)
)

chain = prompt | llm
with
patch("openai.resources.chat.completions.Comple
side_effect=error):
    try:
        print(chain.invoke({"animal": "kangaroo
    except RateLimitError:
        print("Hit error")
```

```
Hit error
```

Fallbacks for Sequences

We can also create fallbacks for sequences, that are sequences themselves. Here we do that with two different models: ChatOpenAI and then normal OpenAI (which does not use a chat model). Because OpenAI is NOT a chat model, you likely want a different prompt.

```
# First let's create a chain with a ChatModel
# We add in a string output parser here so
the outputs between the two are the same type
from langchain_core.output_parsers import
StrOutputParser

chat_prompt =
ChatPromptTemplate.from_messages(
    [
        (
            "system",
            "You're a nice assistant who
always includes a compliment in your
response",
        ),
        ("human", "Why did the {animal} cross
the road"),
    ]
)
# Here we're going to use a bad model name to
easily create a chain that will error
chat_model = ChatOpenAI(model_name="gpt-
fake")
bad_chain = chat_prompt | chat_model |
StrOutputParser()
```

```
# Now lets create a chain with the normal
OpenAI model
from langchain.prompts import PromptTemplate
```

```
from langchain_openai import OpenAI

prompt_template = """Instructions: You should
always include a compliment in your response.

Question: Why did the {animal} cross the
road?"""
prompt =
PromptTemplate.from_template(prompt_template)
llm = OpenAI()
good_chain = prompt | llm
```

```
# We can now create a final chain which
combines the two
chain =
bad_chain.with_fallbacks([good_chain])
chain.invoke({"animal": "turtle"})
```

```
'\n\nAnswer: The turtle crossed the road to
get to the other side, and I have to say he
had some impressive determination.'
```