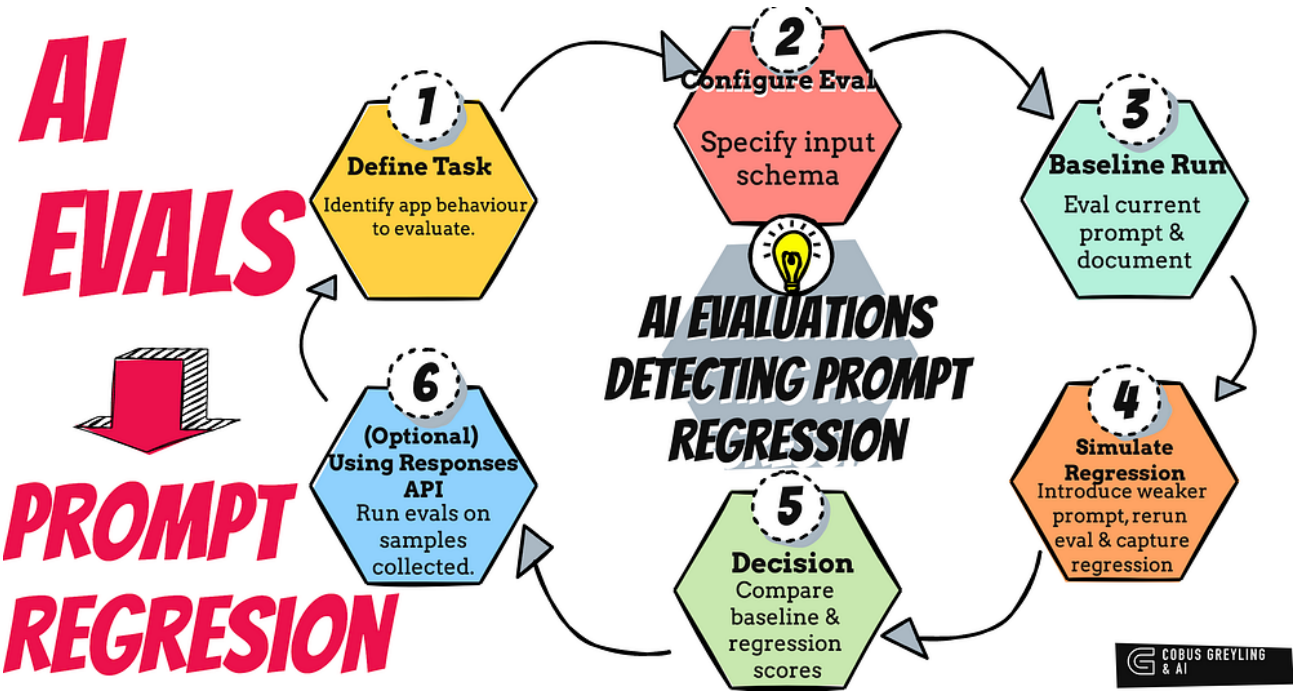
AI Evals (Evaluations) Detecting Regression

AI Evals are useful in cases where you want to migrate Language Models or make Prompt Engineering Changes…

Cobus Greyling • 7 min read • 2025-09-08

https://cobusgreyling.medium.com/ai-evals-evaluations-detecting-regression-548e0a348a96



AI Evals are useful in cases where you want to migrate Language Models or make Prompt Engineering Changes…

I'm trying to get my head around what AI Evals encompass, and the best way of doing this is working through notebooks. Before I get to into the AI Evals portion, there are aa number of reasons you would want to run Evals.

And not necessarily that you are migrating Language Models or changing prompts, but because your underlying model is chancing…

Or will be deprecated and you need settle on new models…

Model Drift

There has been a number studies on model drift, one was called ***catastrophic forgetting***, where the model forgets information.

Another study was called ***lost in the middle*** where with long context or large context window information at the beginning or end of the text corpus is favoured above context and information which is in the middle.

There has been a number of iterations on this principle and study.

There has been anecdotal evidence surfaced on Reddit where the changing behaviour of models were noted. In most cases degradation.

So if you as an organisation is not hosting your own instance of a model, you have no idea on how the model is changing under the hood.

Hence you might be happily running your evaluations, and tweaking where you can, but the underlying model is out of your control.

API Orchestration

Recently OpenAI revealed their Deep Research API architecture.

The architecture behind the API showed that there is not a single model at play, but a number of models (at least 3) are being orchestrated to facilitate and make the API work.

OpenAI also revealed some of the architecture of ChatGPT, and it is the same case. A number of models are orchestrated, Small Language Models, to create the user experience.

So this goes to proof that there are complexities beyond our control and even knowledge.

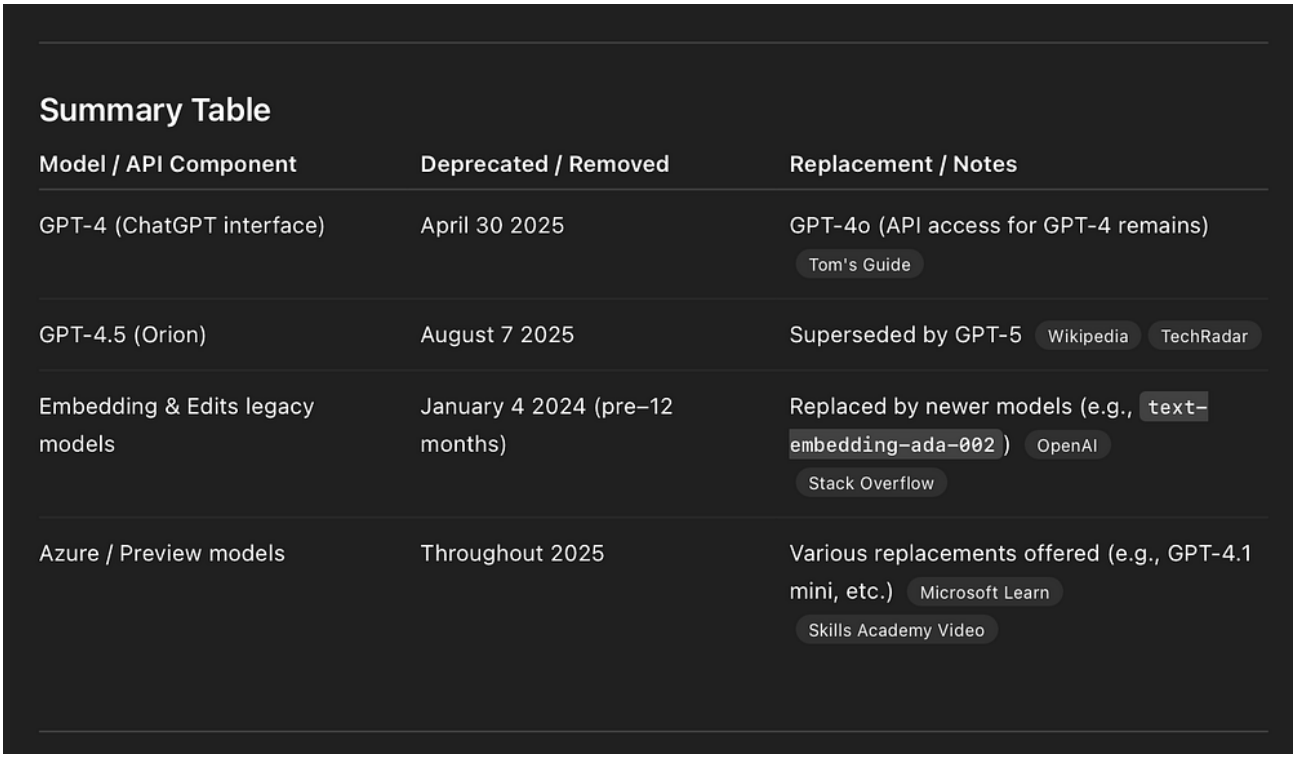
I cannot help but feel, again, for critical applications you need your own private instance of an open-sourced model.

Model Deprecation

In the last 12 months, multiple models were deprecated by OpenAI alone; GPT-4, GPT-4.5, multiple Azure preview model retirements.

So you plan and design around a model, an much work is created for you when a model is deprecated to choose a new model.

And again, the need for AI Evals..



Summary Table

Model / API Component	Deprecated / Removed	Replacement / Notes
GPT-4 (ChatGPT interface)	April 30 2025	GPT-4o (API access for GPT-4 remains) Tom's Guide
GPT-4.5 (Orion)	August 7 2025	Superseded by GPT-5 Wikipedia TechRadar
Embedding & Edits legacy models	January 4 2024 (pre–12 months)	Replaced by newer models (e.g., `text-embedding-ada-002`) OpenAI Stack Overflow
Azure / Preview models	Throughout 2025	Various replacements offered (e.g., GPT-4.1 mini, etc.) Microsoft Learn Skills Academy Video

Source: ChatGPT

Fine-Tuning

Within short while, model fine-tuning will become commonplace. NVIDIA is predicting that the future is one of multiple Small Language Models (SLMs) which will be orchestrated.

And also fine-tuned on a very regular cadence.

They are also a proponent of creating a data-flywheel, or a continuous data feedback loop. And hence this produces highly contextualised and customised Language Models.

AI Evals

> Benchmarks, tests, metrics to assess model performance, safety & capabilities

AI evaluations have various practical applications in development and deployment workflows.

For now, I will focus on Prompt Migration/Update, and Model Migration.

Prompt Migration

Evaluations can help gauge how well existing prompts perform when transferred from one AI model or version to another, identifying any degradation in output quality, relevance or consistency.

Model Migration

Similar to prompt migration, evaluations are used to compare a new model against the old one on key tasks, measuring improvements in accuracy, speed, or robustness before fully switching over.

Prompt Migration Example

AI Evals are

• Task oriented and

• Iterative.

They're the best way to check how your LLM integration is doing and improve it.

Here is a basic example you can run in a Colab Notebook…

```
import openai
from openai.types.chat import ChatCompletion
import pydantic
import os

os.environ["OPENAI_API_KEY"] = os.environ.get("OPENAI_API_KEY", "Your API Key")
```

Here are the evaluation schema…

```
class PushNotifications(pydantic.BaseModel):
    notifications: str

print(PushNotifications.model_json_schema())
```

```
{'properties':

     {'notifications':
        {'title': 'Notifications', 'type': 'string'}},
              'required': ['notifications'],
                    'title': 'PushNotifications',
                          'type': 'object'}
```

The old or rather the correct prompt, you can see the model is defined with the messages and roles.

```
DEVELOPER_PROMPT = """
You are a helpful assistant that summarizes push notifications.
You are given a list of push notifications and you need to collapse them into a
single one.
Output only the final summary, nothing else.
"""

def summarize_push_notification(push_notifications: str) -> ChatCompletion:
    result = openai.chat.completions.create(
        model="gpt-4o-mini",
        messages=[
            {"role": "developer", "content": DEVELOPER_PROMPT},
            {"role": "user", "content": push_notifications},
        ],
    )
    return result

example_push_notifications_list = PushNotifications(notifications="""
- Alert: Unauthorized login attempt detected.
- New comment on your blog post: "Great insights!"
- Tonight's dinner recipe: Pasta Primavera.
""")
result =
summarize_push_notification(example_push_notifications_list.notifications)
print(result.choices[0].message.content)
```

```
# We want our input data to be available in our variables, so we set the
item_schema to
# PushNotifications.model_json_schema()
data_source_config = {
    "type": "custom",
    "item_schema": PushNotifications.model_json_schema(),
```

```
    # We're going to be uploading completions from the API, so we tell the Eval
to expect this
    "include_sample_schema": True,
}
```

Here the valuation prompt is defined, defining what must be evaluated and the model to use in the grader…

```
GRADER_DEVELOPER_PROMPT = """
Label the following push notification summary as either correct or incorrect.
The push notification and the summary will be provided below.
A good push notificiation summary is concise and snappy.
If it is good, then label it as correct, if not, then incorrect.
"""
GRADER_TEMPLATE_PROMPT = """
Push notifications: {{item.notifications}}
Summary: {{sample.output_text}}
"""
push_notification_grader = {
    "name": "Push Notification Summary Grader",
    "type": "label_model",
    "model": "o3-mini",
    "input": [
        {
            "role": "developer",
            "content": GRADER_DEVELOPER_PROMPT,
        },
        {
            "role": "user",
            "content": GRADER_TEMPLATE_PROMPT,
        },
    ],
    "passing_labels": ["correct"],
    "labels": ["correct", "incorrect"],
}
```

```
eval_create_result = openai.evals.create(
    name="Push Notification Summary Workflow",
    metadata={
        "description": "This eval checks if the push notification summary is
correct.",
    },
    data_source_config=data_source_config,
    testing_criteria=[push_notification_grader],
)

eval_id = eval_create_result.id
```

Here are the test data…

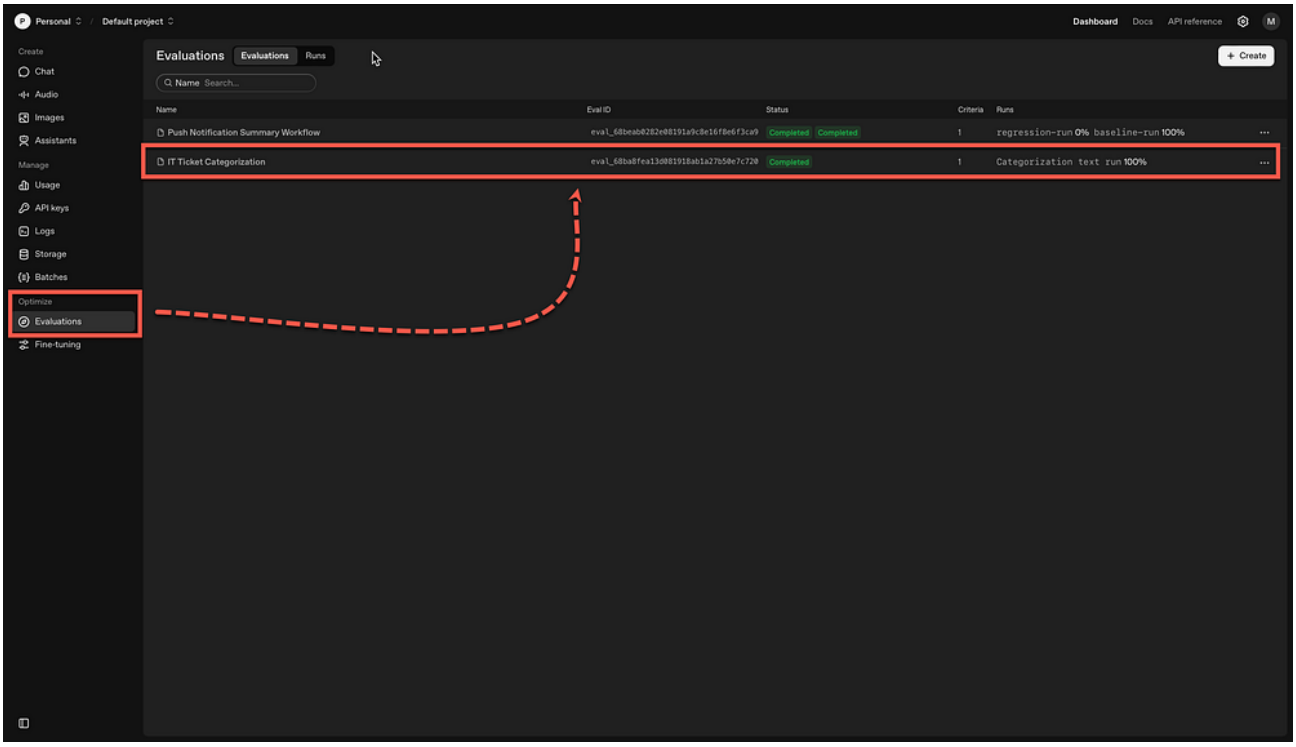
```
push_notification_data = [
        """
- New message from Sarah: "Can you call me later?"
- Your package has been delivered!
- Flash sale: 20% off electronics for the next 2 hours!
""",
        """
- Weather alert: Thunderstorm expected in your area.
- Reminder: Doctor's appointment at 3 PM.
- John liked your photo on Instagram.
""",
        """
- Breaking News: Local elections results are in.
- Your daily workout summary is ready.
- Check out your weekly screen time report.
""",
        """
- Your ride is arriving in 2 minutes.
- Grocery order has been shipped.
- Don't miss the season finale of your favorite show tonight!
""",
        """
- Event reminder: Concert starts at 7 PM.
- Your favorite team just scored!
- Flashback: Memories from 3 years ago.
""",
        """
- Low battery alert: Charge your device.
- Your friend Mike is nearby.
- New episode of "The Tech Hour" podcast is live!
""",
        """
- System update available.
- Monthly billing statement is ready.
- Your next meeting starts in 15 minutes.
""",
        """
- Alert: Unauthorized login attempt detected.
- New comment on your blog post: "Great insights!"
- Tonight's dinner recipe: Pasta Primavera.
""",
        """
- Special offer: Free coffee with any breakfast order.
- Your flight has been delayed by 30 minutes.
- New movie release: "Adventures Beyond" now streaming.
""",
        """
- Traffic alert: Accident reported on Main Street.
- Package out for delivery: Expected by 5 PM.
- New friend suggestion: Connect with Emma.
```

```
"""]
```

The baseline results are generated..

```
run_data = []
for push_notifications in push_notification_data:
    result = summarize_push_notification(push_notifications)
    run_data.append({
        "item":
PushNotifications(notifications=push_notifications).model_dump(),
        "sample": result.model_dump()
    })

eval_run_result = openai.evals.runs.create(
    eval_id=eval_id,
    name="baseline-run",
    data_source={
        "type": "jsonl",
        "source": {
            "type": "file_content",
            "content": run_data,
        }
    },
)
print(eval_run_result)
# Check out the results in the UI
print(eval_run_result.report_url)
```

In the dashboard, if you click on ***Evaluations***, you will see I have two evaluations there. The bottom one now holds the baseline test.



Now we use a prompt which is obviously a worse prompt just by visual inspection…

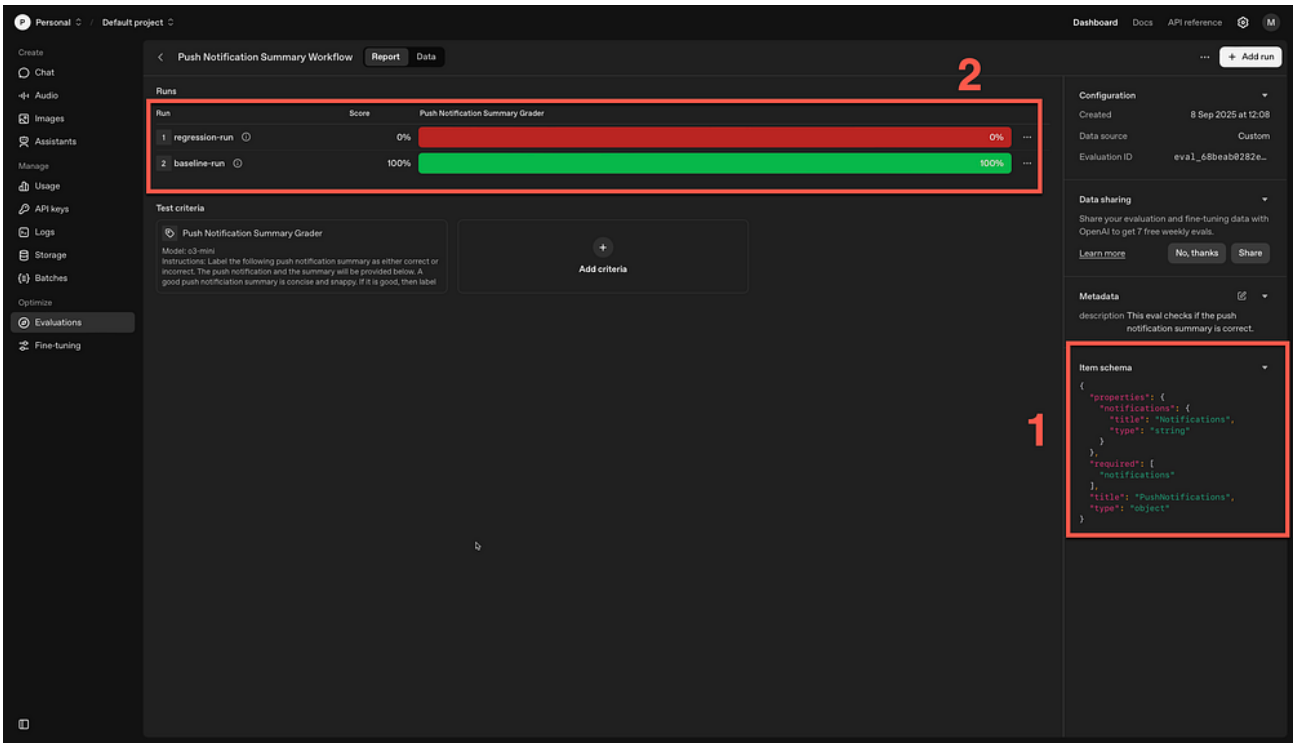
```
DEVELOPER_PROMPT = """
You are a helpful assistant that summarizes push notifications.
You are given a list of push notifications and you need to collapse them into a
single one.
You should make the summary longer than it needs to be and include more
information than is necessary.
"""

def summarize_push_notification_bad(push_notifications: str) -> ChatCompletion:
    result = openai.chat.completions.create(
        model="gpt-4o-mini",
        messages=[
            {"role": "developer", "content": DEVELOPER_PROMPT},
            {"role": "user", "content": push_notifications},
        ],
    )
    return result
```

An the evaluation is run…

```
run_data = []
for push_notifications in push_notification_data:
    result = summarize_push_notification_bad(push_notifications)
    run_data.append({
        "item":
PushNotifications(notifications=push_notifications).model_dump(),
        "sample": result.model_dump()
    })
```

```
eval_run_result = openai.evals.runs.create(
    eval_id=eval_id,
    name="regression-run",
    data_source={
        "type": "jsonl",
        "source": {
            "type": "file_content",
            "content": run_data,
        }
    },
)
print(eval_run_result.report_url)
```

Below you can see the run schema (1) and also the two runs, with the same data, but different prompts.



This example is as simple as can be, and obviously in a production setting you your test data and schema will be much more complex and multiple runs can include a number of models, prompts and graders.

Chief Evangelist *@ Kore.ai | I'm passionate about exploring the intersection of AI and language. Language Models, AI Agents, Agentic Apps, Dev Frameworks & Data-Driven Tools shaping tomorrow.*

---

https://platform.openai.com/docs/guides/evals?api-mode=chat

Other mentions by Author

- www.cobusgreyling.com | COBUS GREYLING
- cookbook.openai.com | Evals API Use-case - Detecting prompt regressions | OpenAI Cookbook
- cobusgreyling.medium.com | Evaluation Driven Design of AI Agents
- cobusgreyling.medium.com | AI Evaluations
- cobusgreyling.medium.com | Written by Cobus Greyling