

# Boîte de savoir

Un petit bot GPT-3 pour répondre aux tickets GitLab

---

19 août 2022



# Table des matières

	Introduction . . . . .	1
1.	Me fixer un livrable . . . . .	1
2.	Jouons avec GitLab . . . . .	2
3.	Jouons avec OpenAI . . . . .	3
4.	Une interface minimaliste . . . . .	3
5.	Customisons notre modèle . . . . .	5
6.	Livrons . . . . .	7
	Conclusion . . . . .	8

## Introduction

J'ai plusieurs idées de petits projets qui ne me prendraient pas plus d'un week-end à faire (un gartic phone avec DALL-E, jouer avec OpenPush, etc) mais à force de répondre à des tickets similaires, je me suis demandé à quel point ce serait automatisable (pour le fun, en vrai on a écrit une FAQ et on redirige vers cette page si nécessaire ou autre article dédié répondant à la question en général).

J'ai donc pris quelques heures pour jouer avec GPT-3 de mon côté (j'avais un accès à la bêta depuis 2 mois sans l'avoir utilisé) afin de réaliser un petit bot rapide qui répondrait aux nouveaux tickets "à ma place" appelé GePeTTTo.

## 1. Me fixer un livrable

Des projets trop gros, trop long et pas terminés, j'en ai des dizaines. Et déjà que je programme 40+ heures par semaine (travail + associatif, tout ca...), le soir j'ai plus le temps que je me donnais il y a quelques années et je préfère faire autre chose (cuisiner, apprendre des tours au chat, voir des gens).

Du coup j'ai pris l'habitude de bien plus scoper mes projets personnels qu'avant.

Et donc, voici ce que je souhaitais donc avoir un petit programme CLI qui :

- Récupérez les dernières issues d'un GitLab
- Pourrait prendre l'issue depuis une URL ou un fichier (ca me permet de tester des sources autres que le GitLab facilement)
- Génère depuis un modèle customisé des réponses que je peux soit envoyer, soit sauvegarder (pour modifier). Il aurait été possible de l'envoyer automatiquement via un compte GitLab, mais en soit, à part héberger yet another bot, ca m'aurait pas été tant utile.

## 2. Jouons avec GitLab

Et donc pour ce faire, les étapes étaient assez simples :

1. Jouer avec l'API de GitLab pour récupérer les projets/issues/discussions et poster des commentaires
2. Récupérer tous mes commentaires postés pour le transformer en modèle.
3. Jouer avec l'API GPT-3 pour customiser un modèle et générer des réponses
4. Faire une petite interface Python jolie pour rendre ça utilisable
5. Fixer les bugs et publier sur un repo quelconque
6. Écrire un article ici

## 2. Jouons avec GitLab

GitLab possède une API relativement énorme, mais bien documentée (après GitLab est une usine à gaz à elle toute seule, donc l'API suit le mouvement).

La première étape étant de récupérer un token d'accès ([https://INSTANCE\\_GITLAB/-/profile/personal\\_access\\_tokens](https://INSTANCE_GITLAB/-/profile/personal_access_tokens) une fois connecté) avec les permissions `api` et `read_api` pour la suite.

Puis, récupérer les APIs intéressantes, dans mon cas:

<https://docs.gitlab.com/ee/api/issues.html> ↗

<https://docs.gitlab.com/ee/api/discussions.html> ↗

<https://docs.gitlab.com/ee/api/projects.html> ↗

Pour la suite, la manière la plus simple de voir les résultats à parser est d'utiliser ce bon vieux `curl`:

```
1 # Pour récupérer les projets d'une instance.
2 curl https://gitlab.com/api/v4/projects
3 # ou encore pour poster un commentaire dans l'issue 42 du projet 42
4 curl -XPOST --header "PRIVATE-TOKEN:tres_tres_prive" \
    "https://gitlab.com/api/v4/projects/42/issues/42/notes?body=zds"
```

Côté script final, le plus simple est d'utiliser le module [requests](#) ↗

Un petit exemple d'utilisation:

```
1 def answer(self, project_id, issue_id, body):
2     endpoint = f_{self.git_url}/api/v4/projects/{project_id}/issues/{issue_id}/notes?body={body}
3     response = requests.post(endpoint, headers=self.headers)
4     if 'error' in response.json():
5         print(f'An error occurred while posting to {endpoint}')
6     else:
7         print(f'Answered to issue {issue_id} with success')
```

### 3. Jouons avec OpenAI

## 3. Jouons avec OpenAI

### 3.0.1. Récupérons un token

L'API que je vais utiliser vient de <https://openai.com/api/> ↗

Une fois l'accès obtenu, il est possible de se générer une clé d'API ici : <https://beta.openai.com/account/api-keys> ↗

### 3.0.2. Jouons un peu

De nombreux exemples d'applications sont disponibles [ici](#) ↗ et un playground est disponible [ici](#) ↗

Par exemple, avec <https://stackoverflow.com/questions/73311716/gpt3-point-as-terminator-of-sentence> ↗, voici la réponse de GPT-3 depuis le playground.

Answer to this:

"

I am trying to fine tune GPT 3 for question answering task with a customized data. But my data contains sentences like "Highest temperature of India 50.6 degree Celsius". But the model takes the point in between 50.6 as termination. and if I ask question like "What is the highest temperature of India?", it gives me a answer as "Highest temperature of India is 50." How do I overcome this problem.

"

You could try to add a special token to your data to indicate the end of a number, such as "Highest temperature of India 50.6 degree Celsius." -> "Highest temperature of India 50.6 degree Celsius. <END>" This might help the model to better learn when to terminate a number.

FIGURE 3.1. – gpt-3

Mais au final, dans le use-case que je me suis fixé, demandé à GPT-3 de répondre à l'issue (sinon par défaut il continue généralement juste le texte)

## 4. Une interface minimaliste

### 4.0.1. Minimaliste mais jolie

Pour ce faire, je vais utiliser [cmd](#) ↗, qui est facile à prendre en main et gère déjà pas mal les inputs en CLI (revenir dans les commandes précédentes, `Ctrl+W`, `Ctrl+U`, etc.) et il est plutôt facile d'ajouter des commandes. Par exemple:

```
1 def do_save(self, arg):
2     filename = arg if arg != '' else
3         f'{self.answeringTo["id"]}.answer'
4     with open(filename, 'w+') as f:
5         f.write(self.answer)
```

#### 4. Une interface minimaliste

Ce qui permet d'implémenter en quelques lignes une commande `save` (qui sauvegarde dans un fichier passé en argument ou généré). ou encore pour la commande `help`:

```
1 def do_help(self, args=''):
2     print('start: Start to analyze issues')
3     print('save <file>: Save generated answer to <file>')
4     print('send: Send answer without saving it')
5     print('send <file>: Send answer from file')
6     print('add <url>: Add an issue from an url')
7     print(
8         'add <file>: Add an issue from a file (no send possible)')
9     print('next: Pass to the next issue')
10    print('exit: Close GePeTTTo')
```

Enfin il est possible d'afficher du texte dans une couleur souhaitée:

```
1 print(colored('text', 'green'))
```

#### 4.0.2. Quelques screenshots

Et voici ce que donne l'interface:

## 5. Customisons notre modèle

```
Answer to Issue #800 (by ovari) - Description:
W: https://dl.jami.net/nightly/ubuntu_22.04/dists/jami/InRelease: Key is stored in legacy trusted.gpg
keyring (/etc/apt/trusted.gpg), see the DEPRECATION sections in apt-key(8) for details.

An error occurred
The following details are provided:
W: https://dl.jami.net/nightly/ubuntu_22.04/dists/jami/InRelease: Key is stored in legacy trusted.gpg
keyring (/etc/apt/trusted.gpg), see the DEPRECATION sections in apt-key(8) for details.

Linux Mint 21 Cinnamon
jami-client-qt Version: 202208091632

=====

Proposed answer from GePeTTTo:

The error message is telling you that the key for the Jami repository is stored in the legacy trusted
.gpg keyring. This keyring is no longer used by apt, so you need to remove the key from it.

To do this, you can use the apt-key command:

sudo apt-key del <key id>

Replace <key id> with the key ID of the Jami repository key.
Type help or ? to list commands.
(GePeTTTo) 
```

FIGURE 4.2. – Une première réponse

```
Answer to Issue #798 (by ovari) - Description:
[Small screen] First window is wider than previously
First window is wider than previously.
The smallest width and height are shown in the image below.

Will it fit on the screen of a PinePhone/Librem 5?
Thank you

=====

Proposed answer from GePeTTTo:

Yes, the PinePhone/Librem 5 will be able to accommodate the first window.
Type help or ? to list commands.
(GePeTTTo) 
```

FIGURE 4.3. – Et une seconde

## 5. Customisons notre modèle

Maintenant qu'on a un petit script python pour faire ce que l'on souhaite, il est temps d'essayer d'améliorer les générations. Pour ce faire, je suis tombé sur cet article: [Customizing GPT-3 for Your Application](#) [↗](#) qui semble donner la marche à suivre.

## 5. Customisons notre modèle

J'ai donc décidé de faire une première tentative en parsant toutes les discussions d'un GitLab afin de récupérer tous les commentaires que j'avais fait. Ceci m'a donné quelques milliers d'entrées après avoir filtré (pas mal de ces commentaires étaient juste "Je ferme", "fixé", "Ceci est un duplicata de ...", ce que je ne veux pas que le bot génère en soit). Ces entrées étaient de la forme:

```
1 { "prompt": "Answer to this:\n<commentaire_précédent>",  
  "completion": "<ma_réponse>###"}  
2 { "prompt": "Answer to this:\n<commentaire2_précédent>",  
  "completion": "<ma_réponse2>###"}  
3 ...
```

que je propose alors en entrée à l'API d'OpenAI avec la commande suivante:

```
1 openai api fine_tunes.create -m davinci --n_epochs 2 -t monfichier
```

qui me génère un modèle que je peux alors utiliser via le module Python OpenAI avec:

```
1 openai.Completion.create(  
2     model="davinci:ft-personal-2022-00-00-00-00-00",\  
3     prompt=f"Answer to this issue:\n\n{json.dumps(issue['body'])}"\  
4     ,\  
5     temperature=0, max_tokens=500\  
6 )
```

1. Beaucoup de réponses avaient encore moins de sens qu'avant la customisation et beaucoup de déchets étaient générés.
2. Des caractères étranges étaient générés au milieu de la réponse.
3. Bizarrement peu importe ce que je générerais, la réponse était toujours de longueur `max_token` (soit ~500 mots) contrairement au modèle `text-davinci-002`.

En voyant l'état des réponses, je me suis dit qu'au final les issues en entrées étaient déjà pas mal diversifiées et possédaient souvent des traces de logs, parfois du code ou du bruit autre et que les meilleurs tickets étaient au final les questions sur le fonctionnement général. Ainsi, j'avais 2 choix, qui dans tous les cas allaient me prendre plus de temps que prévu pour générer un meilleur modèle. Soit je triais le modèle généré à la mano, soit je faisais moi-même un mini modèle d'entraînement en ne prenant qu'une FAQ que j'avais écrite + mes réponses à des commentaires d'un certain tag (le tag `question` sur le GitLab que j'utilisais).

Une fois ce problème de déchets générés, je me suis rendu compte que pour obtenir de meilleurs résultats, je pouvais facilement supprimer le maximum de formattage possible en convertissant le markdown récupéré en plain text (à l'aide des modules `markdown` et `BeautifulSoup`).

## 6. Livrons

Finalement, le 3<sup>e</sup> point m'a pris un peu de temps. J'ai essayé plusieurs modèles avec ou sans le marqueur de fin (que j'ai changé par `<END>`) avant de me rendre compte que l'API d'OpenAI prenait un paramètre **stop**:

```
1 openai.Completion.create(\
2     model="davinci:ft-personal-2022-00-00-00-00-00",\
3     prompt=f"Answer to this issue:\n\n{json.dumps(issue['body'])}"
4     ,\
5     temperature=0, max_tokens=500\
6     stop="<END>"
7 )
```

Et voilà!

## 6. Livrons

### 6.0.1. Le projet

Au final, le code est disponible ici <https://github.com/AmarOk1412/GePeTTTo> et devrait être utilisable.

### 6.0.2. Des exemples

### 6.0.3. D'autres exemples, en français

Maintenant, qu'est-ce que ça donne si, à la place d'un bot GitLab, on en profite pour faire un bot ZdS (je ne l'ai pas fait, mais dans l'idée, ce serait [trivial](#)).

Voilà les 3 réponses en français que j'ai fait générer :

1. Pour répondre au premier post de <https://zestedesavoir.com/forums/sujet/16395/charger-ordinateur-avec-thunderbolt-ou-port-natif/>

Oui, vous pouvez recharger votre ordinateur portable Thunderbolt 4 via le port USB sans rien configurer. Cependant, il est recommandé d'utiliser le chargeur natif pour une meilleure performance.

1. <https://zestedesavoir.com/forums/sujet/16355/probleme-performances-chrome/> (seulement le premier post)

Il y a plusieurs raisons qui pourraient expliquer les performances de Chrome. Il se peut que vous ayez trop d'onglets ouverts, ce qui ralentit votre ordinateur. Vous pouvez également avoir des extensions ou des programmes en cours d'exécution qui utilisent beaucoup de mémoire et de CPU.

Il existe plusieurs façons de résoudre ce problème. Vous pouvez d'abord essayer de fermer

## Conclusion

quelques onglets ou de désactiver temporairement les extensions que vous n'utilisez pas. Vous pouvez également vider le cache et les cookies de Chrome pour voir si cela améliore les performances.

1. Un petit test custom :

**Moi** : Répond à ceci :

”Bonjour, J’essaye d’héberger chez moi une instance de Jitsi. Cependant, je ne comprends pas tous les paramètres notamment `useTurnUdp`. Qu’est-ce qu’un serveur TURN et pourquoi est-ce désactivé par défaut ? ”

**GPT-3** : Bonjour,

Un serveur TURN est un serveur qui peut être utilisé pour relayer des flux audio et vidéo en cas de besoin. Il est désactivé par défaut car il n’est pas toujours nécessaire, mais peut être activé si vous rencontrez des problèmes de connectivité.

**Moi** : ”Quel genre de problème de connectivité ?”

**GPT-3** : Les problèmes de connectivité peuvent être dus à des blocages de pare-feu, à des connexions instables ou à d’autres facteurs. Le serveur TURN peut aider à contourner ces problèmes en relayant les flux audio et vidéo directement entre les participants, sans passer par le réseau.

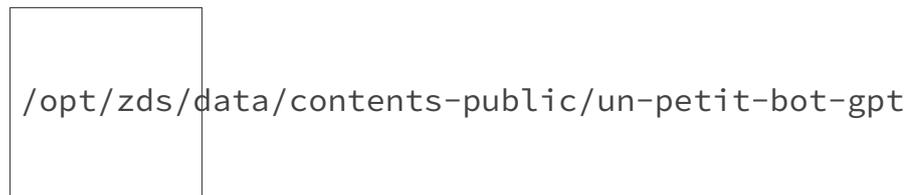


FIGURE 6.4. – Une petite démo animée

## Conclusion

Conclusion de ce petit week-end (au final, forcément un peu plus long) d’expérimentations, c’est que ça marche plutôt bien. Y a tout de même pas mal de limites/ratés, mais des réponses sont vraiment bien et le dataset derrière doit être monstrueusement énorme. Comme DALL-E 2, les tests en d’autres langues ont l’air de fonctionner aussi et le tout était tout de même rapide à faire et facile à mettre en place.

Cependant, il s’agit encore d’une bêta. La documentation semble parfois imprécise, des modèles ne sont pas faits pour être customisés et c’est bien dommage (davinci versus text-davinci-002...). Aussi, customiser son modèle demande de faire gaffe à son dataset d’entrée et les prompts doivent idéalement être le plus propre possible (pas de code ou du markdown, sauf si c’est pour générer du code ou le clarifier).