Transformer Discord Notifier

Release 0.4.4

Erik Körner

Dec 23, 2020

CONTENTS

CHAPTER

ONE

OVERVIEW

docs	
tests	
package	

A Discord Notifier to send progress updates, params and results to a Discord channel.

- Free software: MIT license

1.1 Installation

```
pip install transformer-discord-notifier
```

You can also install the in-development version with:

```
pip install https://github.com/Querela/python-transformer-discord-notifier/archive/
↪master.zip
```

1.2 Documentation

https://python-transformer-discord-notifier.readthedocs.io/

```
git clone https://github.com/Querela/python-transformer-discord-notifier.git
cd python-transformer-discord-notifier
sphinx-build -b html docs dist/docs
```

1.3 Development

To run all the tests run:

```
tox
```

Note, to combine the coverage data from all the tox environments run:

Windows	`set PYTEST_ADDOPTS=--cov-append` `tox`
Other	`PYTEST_ADDOPTS=--cov-append tox`

CHAPTER

TWO

INSTALLATION

At the command line:

```
pip install transformer-discord-notifier
```

CHAPTER

THREE

USAGE

3.1 Using `DiscordProgressCallback`

How to use the *`DiscordProgressCallback`* in a huggingface.co Transformer in a project/training script:

```
from transformers import Trainer
# ... other import ...
from transformer_discord_notifier import DiscordProgressCallback

def run_trainer():
        # ... set up things beforehand ...

        # Initialize the Discord bot
        dpc = DiscordProgressCallback(token=None, channel=None)
        dpc.start()

        # Initialize our Trainer
        trainer = Trainer(
                model=model,
                args=training_args,
                train_dataset=train_dataset,
                eval_dataset=eval_dataset,
                # ...
                # add our callback to the trainer
                callbacks=[dpc]
        )

        # ... do things like train/eval/predict

        # shutdown our discord handler as it would continue to run indefinitely
        dpc.end()
```

Alternatively, since version *v0.2.0* it is possible to omit the starting and stopping of the *`DiscordProgressCallback`*, and it can be used like any other huggingface.co callback handler:

```
from transformers import Trainer
# ... other import ...
from transformer_discord_notifier import DiscordProgressCallback

def run_trainer():
        # ... set up transformer stuff beforehand ...

        # Initialize our Trainer
        trainer = Trainer(
```

(continues on next page)

(continued from previous page)

```
                model=model,
                args=training_args,
                train_dataset=train_dataset,
                eval_dataset=eval_dataset,
                # ...
                # add our callback to the trainer
                callbacks=[DiscordProgressCallback]
        )

        # ... do things like train/eval/predict
        # ... when the trainer instance is garbage collected, it will clean up the␣
↪Discord bot
```

Note, however, that the both `token` and `channel` should be provided, either as class initialization parameters or as environment variables, `DISCORD_TOKEN` and `DISCORD_CHANNEL`. The handler will try to load from environment variables if the instance properties are `None`. Both should be explicitely provided to have it working correctly!

3.2 How to setup a Discord bot

How to setup a Discord bot, how to get the token or the channel id? Please visit the following links:

- How to create a bot?
- Related project discord-notifier-bot, setup guide in README

CHAPTER

FOUR

REFERENCE

4.1 transformer_discord_notifier

Imports *DiscordClient* and *DiscordProgressCallback*.

4.2 transformer_discord_notifier.discord

class transformer_discord_notifier.discord.**DiscordClient**(*token: Optional[str] = None*, *channel: Optional[Union[str, int]] = None*)

A blocking wrapper around the asyncio Discord.py client.

_load_credentials() → None

Try to load missing Discord configs (token, channel) from environment variables.

_find_default_channel(*name: Optional[str] = None*, *default_name: str = 'default'*) → int

Try to find a writable text channel.

Follow the following algorithm:

1. if `name` is being provided, search for this channel first
2. if not found, search for `self._discord_channel`, then channel that can be configured on instance creation or by loading environment variables. Check first for a channel with the given name as string, then fall back to an integer channel id.
3. if still not found, search for a channel with a given default name, like "default" or "Allgemein". As this seems to depend on the language, it might not find one.

If after all this still no channel has been found, either because no channel with the given names/id exists, or because the Discord token gives no acces to guilds/channels which we have access to, we throw a `RuntimeError`. We now can't use this callback handler.

Parameters

- **name** (*Optional[str], optional*) – channel name to search for first, by default None
- **default_name** (*str, optional*) – alternative default Discord channel name, by default "default"

Returns *int* – channel id

Raises

- **RuntimeError** – raised if no *guild* Discord server found (i.e. Discord bot has no permissions / was not yet invited to a Discord server)
- **RuntimeError** – raised if channel could not be found

init()
Initialize Discord bot for accessing Discord/writing messages.

It loads the credentials, starts the asyncio Discord bot in a separate thread and after connecting searches for our target channel.

Raises **RuntimeError** – raised on error while initializing the Discord bot, like invalid token or channel not found, etc.

_quit_client()
Internal. Try to properly quit the Discord client if neccessary, and close the asyncio loop if required.

quit()
Shutdown the Discord bot.

Tries to close the Discord bot safely, closes the asyncio loop, waits for the background thread to stop (deamonized, so on program exit it will quit anyway).

send_message(*text: str = ''*, *embed: Optional[discord.embeds.Embed] = None*) → Optional[int]
Sends a message to our Discord channel. Returns the message id.

Parameters

- **text** (*str, optional*) – text message to send, by default “”
- **embed** (*Optional[discord.Embed], optional*) – embed object to attach to message, by default None

Returns *Optional[int]* – message id if *text* and *embed* were both not `None`, `None` if nothing was sent

get_message_by_id(*msg_id: int*) → Optional[discord.message.Message]
Try to retrieve a Discord message by its id.

Parameters **msg_id** (*int*) – message id of message sent in Discord channel

Returns *Optional[discord.Message]* – `None` if message could not be found by *msg_id*, else return the message object

update_or_send_message(*msg_id: Optional[int] = None*, ***fields*) → Optional[int]
Wrapper for *send_message()* to updated an existing message, identified by *msg_id* or simply send a new message if no prior message found.

Parameters

- **msg_id** (*Optional[int], optional*) – message id of prior message sent in channel, if not provided then send a new message.
- **text** (*str, optional*) – text message, if set to `None` it will remove prior message content
- **embed** (*Optional[discord.Embed], optional*) – Discord embed, set to `None` to delete existing embed

Returns *Optional[int]* – message id of updated or newly sent message, `None` if nothing was sent

delete_later(*msg_id: int*, *delay: Union[int, float] = 5*) → bool
Runs a delayed message deletion function.

Parameters

- **msg_id** (*int*) – message id of message sent in Discord channel
- **delay** (*Union[int, float], optional*) – delay in seconds for then to delete the message, by default 5

Returns *bool* – `True` if message deletion is queued, `False` if message could not be found in channel

static build_embed(*kvs: Dict[str, Any]*, *title: Optional[str] = None*, *footer: Optional[str] = None*) → discord.embeds.Embed

Builds an rich Embed from key-values.

Parameters

- **kvs** (*Dict[str, Any]*) – Key-Value dictionary for embed fields, non `int`/`float` values will be formatted with `pprint.pformat()`
- **title** (*Optional[str], optional*) – title string, by default None
- **footer** (*Optional[str], optional*) – footer string, by default None

Returns *discord.Embed* – embed object to send via *send_message()*

The *DiscordClient* can be used standalone, but it might be easier to just extract the module code to avoid having to install all the related transformers requirements. It wraps the asyncio Discord.py client inside a background thread and makes its calls essentially blocking. This eases the usage of it in foreign code that does not uses asyncio.

```
from transformer_discord_notifier.discord import DiscordClient

# configuration
token = "abc123.xyz..."
channel = "Allgemein"

# create client and start background thread, to connect/login ...
# if token/channel are None, it will try to load from environment variables
client = DiscordClient(token=token, channel=channel)
client.init()

# send message
msg_id = client.send_message("test")

# update message content
msg_id = client.update_or_send_message(text="abc", msg_id=msg_id)

# delete it after 3.1 seconds,
# NOTE: this call will not block!
client.delete_later(msg_id, delay=3.1)

# quit client (cancel outstanding tasks!, quit asyncio thread)
client.quit()
```

4.3 transformer_discord_notifier.transformers

class transformer_discord_notifier.transformers.**DiscordProgressCallback**(*token: Optional[str] = None*, *channel: Optional[Union[str, int]] = None*)

Bases: transformers.trainer_callback.ProgressCallback

An extended transformers.trainer_callback.ProgressCallback that logs training and evaluation progress and statistics to a Discord channel.

Variables

- **client** (DiscordClient) – a blocking Discord client
- **disabled** (*bool*) – True if Discord client couldn't not be initialized successfully, all callback methods are disabled silently

Parameters

- **token** (*Optional[str], optional*) – Discord bot token, by default None
- **channel** (*Optional[Union[str, int]], optional*) – Discord channel name or numeric id, by default None

start() → None
Start the Discord bot.

end() → None
Stop the Discord bot. Cleans up resources.

on_init_end(*args: transformers.training_args.TrainingArguments*, *state: transformers.trainer_callback.TrainerState*, *control: transformers.trainer_callback.TrainerControl*, ***kwargs*)
Event called at the end of the initialization of the Trainer.

_new_tqdm_bar(*desc: str*, *msg_fmt: str*, *delete_after: bool = True*, ***kwargs*) → Tuple[tqdm.std.tqdm, transformer_discord_notifier.transformers.MessageWrapperTQDMWriter]
Builds an internal tqdm wrapper for progress tracking.

Patches its file.write method to forward it to Discord. Tries to update existing messages to avoid spamming the channel.

on_train_begin(*args: transformers.training_args.TrainingArguments*, *state: transformers.trainer_callback.TrainerState*, *control: transformers.trainer_callback.TrainerControl*, ***kwargs*)
Event called at the beginning of training.

on_prediction_step(*args: transformers.training_args.TrainingArguments*, *state: transformers.trainer_callback.TrainerState*, *control: transformers.trainer_callback.TrainerControl*, *eval_dataloader=None*, ***kwargs*)
Event called after a prediction step.

on_step_end(*args: transformers.training_args.TrainingArguments*, *state: transformers.trainer_callback.TrainerState*, *control: transformers.trainer_callback.TrainerControl*, ***kwargs*)
Event called at the end of a training step. If using gradient accumulation, one training step might take several inputs.

on_epoch_begin(*args: transformers.training_args.TrainingArguments*, *state: transformers.trainer_callback.TrainerState*, *control: transformers.trainer_callback.TrainerControl*, ***kwargs*)
Event called at the beginning of an epoch.

on_epoch_end(*args: transformers.training_args.TrainingArguments*, *state: transformers.trainer_callback.TrainerState*, *control: transformers.trainer_callback.TrainerControl*, ***kwargs*)
Event called at the end of an epoch.

on_train_end(*args: transformers.training_args.TrainingArguments*, *state: transformers.trainer_callback.TrainerState*, *control: transformers.trainer_callback.TrainerControl*, ***kwargs*)
Event called at the end of training.

on_evaluate(*args: transformers.training_args.TrainingArguments*, *state: transformers.trainer_callback.TrainerState*, *control: transformers.trainer_callback.TrainerControl*, ***kwargs*)
Event called after an evaluation phase.

_send_log_results(*logs: Dict[str, Any]*, *state: transformers.trainer_callback.TrainerState*, *args: transformers.training_args.TrainingArguments*, *is_train: bool*) → int
Formats current log metrics as Embed message.

Given a huggingface transformers Trainer callback parameters, we create an `discord.Embed` with the metrics as key-values. Send the message and returns the message id.

on_log(*args: transformers.training_args.TrainingArguments*, *state: transformers.trainer_callback.TrainerState*, *control: transformers.trainer_callback.TrainerControl*, *logs: Optional[Dict[str, Any]] = None*, ***kwargs*)
Event called after logging the last logs.

CHAPTER

FIVE

CONTRIBUTING

Contributions are welcome, and they are greatly appreciated! Every little bit helps, and credit will always be given.

5.1 Bug reports

When reporting a bug please include:

- Your operating system name and version.
- Any details about your local setup that might be helpful in troubleshooting.
- Detailed steps to reproduce the bug.

5.2 Documentation improvements

Transformer Discord Notifier could always use more documentation, whether as part of the official Transformer Discord Notifier docs, in docstrings, or even on the web in blog posts, articles, and such.

5.3 Feature requests and feedback

The best way to send feedback is to file an issue at https://github.com/Querela/python-transformer-discord-notifier/issues.

If you are proposing a feature:

- Explain in detail how it would work.
- Keep the scope as narrow as possible, to make it easier to implement.
- Remember that this is a volunteer-driven project, and that code contributions are welcome :)

5.4 Development

To set up *python-transformer-discord-notifier* for local development:

1. Fork python-transformer-discord-notifier (look for the "Fork" button).
2. Clone your fork locally:

```
git clone git@github.com:YOURGITHUBNAME/python-transformer-discord-notifier.git
```

3. Create a branch for local development:

```
git checkout -b name-of-your-bugfix-or-feature
```

   Now you can make your changes locally.

4. When you're done making changes run all the checks, tests and rebuild docs:

```
python setup.py check --strict --metadata --restructuredtext
check-manifest
flake8
isort --verbose --check-only --diff --filter-files src

sphinx-build -b doctest docs dist/docs
sphinx-build -b html docs dist/docs
sphinx-build -b linkcheck docs dist/docs

pytest
```

   Or you can use tox to automatically run those commands:

```
tox
```

   or just a single test:

```
tox -e check,docs
tox -e py38
```

   Note, that the tests with `pytest` require a valid Discord token and channel. They must be provided as `--discord-token token`, `--discord-channel chan` or `DISCORD_TOKEN=token`, `DISCORD_CHANNEL=chan`.

   You can set the environment variables in the `.env` file to make them visible to both `pytest` and `tox` environments.

   If you use VSCode, configure it to use an environment variable file, in `.vscode/settings.json` the setting `"python.envFile": "${workspaceFolder}/.env"`.

5. Commit your changes and push your branch to GitHub:

```
git add .
git commit -m "Your detailed description of your changes."
git push origin name-of-your-bugfix-or-feature
```

6. Submit a pull request through the GitHub website.

5.4.1 Pull Request Guidelines

If you need some code review or feedback while you're developing the code just make the pull request.

For merging, you should:

1. RUn extensive tests.
2. Update documentation when there's new API, functionality etc.
3. Add a note to `CHANGELOG.rst` about the changes.
4. Add yourself to `AUTHORS.rst`.

CHAPTER

SIX

AUTHORS

- Erik Körner - koerner@informatik.uni-leipzig.de

CHAPTER

SEVEN

CHANGELOG

7.1 0.4.5 (WIP)

- ignore linkcheck with version tag (if tags have not been pushed it will fails)
- Blocking message deletion?

7.2 0.4.4 (2020-12-22)

- Github Actions - tox tests

7.3 0.4.3 (2020-12-18)

- Github Actions - pypi publishing

7.4 0.4.2 (2020-12-18)

- Add travis build jobs.
- Add coveralls coverage statistics.

7.5 0.4.1 (2020-12-17)

- Reintroduce tests with `pytest` and `tox`.
- Add simple tests for *DiscordClient*.
- Add tests for *DiscordProgressCallback*.

7.6 0.3.1 (2020-12-17)

- Let Discord bot gracefully handle initialization failures.
- Let transformer callback handler handle invalid configs gracefully, to simply exit.
- Better handling of edge cases of Discord client login.

7.7 0.3.0 (2020-12-16)

- Add (private) scripts (make venv, run checks).
- Update usage docs.
- Extend / rewrite discord client methods.
- Reuse existing `tqdm` `transformers.trainer_callback.ProgressCallback` for progress tracking.
- Fancy aggregation of prediction runs, split train progress into epochs.

7.8 0.2.1 (2020-12-15)

- Correct `setup.py` validation.
- Add (private) distribution/docs build scripts.

7.9 0.2.0 (2020-12-15)

- Refactor blocking discord code into *`discord`* submodule.
- Fix behaviour for `__del__` with refactoring, so it work as intended.
- Improve documentation for *`discord`* module.

7.10 0.1.0 (2020-12-11)

- First release on PyPI.
- First working version, tested manually.
- Cleaned up skeleton files.
- Updated docs.

7.11 0.0.0 (2020-12-10)

- Initial code skeleton.

CHAPTER

EIGHT

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

t

INDEX

Symbols

B

D

E

G

I

M

O

Q

S

T

U