```
from swarms.utils.auto_download_check_packages import (
    auto_check_and_download_package,
)



try:
    import torch
except ImportError:
    auto_check_and_download_package(
        "torch", package_manager="pip", upgrade=True
    )
    import torch

try:
    import transformers
except ImportError:
    auto_check_and_download_package(
        "transformers", package_manager="pip", upgrade=True
    )
    import transformers



class StringStoppingCriteria(transformers.StoppingCriteria):
    def __init__(
        self, tokenizer: transformers.PreTrainedTokenizer, prompt_length: int  # type: ignore
    ):
```

```
        self.tokenizer = tokenizer
        self.prompt_length = prompt_length

    def __call__(
        self,
        input_ids: torch.LongTensor,  # type: ignore
        _,
    ) -> bool:
        if len(input_ids[0]) <= self.prompt_length:
            return False

        last_token_id = input_ids[0][-1]
        last_token = self.tokenizer.decode(
            last_token_id, skip_special_tokens=True
        )

        result = "" in last_token

        return result


class NumberStoppingCriteria(transformers.StoppingCriteria):
    def __init__(
        self,
        tokenizer: transformers.PreTrainedTokenizer,  # type: ignore
        prompt_length: int,
```

```
    precision: int = 3,
):
    self.tokenizer = tokenizer
    self.precision = precision
    self.prompt_length = prompt_length

def __call__(
    self,
    input_ids: torch.LongTensor,  # type: ignore
    scores: torch.FloatTensor,  # type: ignore
) -> bool:
    decoded = self.tokenizer.decode(
        input_ids[0][self.prompt_length :],
        skip_special_tokens=True,
    )

    if decoded.count(".") > 1:
        return True

    if (
        decoded.count(".") == 1
        and len(decoded.strip().split(".")[1]) > self.precision
    ):
        return True

    if (
```

```
            len(decoded) > 1
            and any(c.isdigit() for c in decoded)
            and decoded[-1] in [" ", "\n"]
        ):
            return True

        return False



class OutputNumbersTokens(transformers.LogitsWarper):
    def __init__(self, tokenizer: transformers.PreTrainedTokenizer, prompt: str):  # type: ignore
        self.tokenizer = tokenizer
        self.tokenized_prompt = tokenizer(prompt, return_tensors="pt")
        vocab_size = len(tokenizer)
        self.allowed_mask = torch.zeros(vocab_size, dtype=torch.bool)

        for _, token_id in tokenizer.get_vocab().items():
            token_str = tokenizer.decode(token_id).strip()

            if token_str == "" or (
                all(c.isdigit() or c == "." for c in token_str)
                and token_str.count(".") <= 1
            ):
                self.allowed_mask[token_id] = True

    def __call__(self, _, scores):
```

```
mask = self.allowed_mask.expand_as(scores)

scores[~mask] = -float("inf")


return scores
```