```
import requests
import pandas as pd
from collections import Counter
import pandas as pd
import plotly.express as px
```

Reading The Dataset

```
url = "https://api.github.com/repos/rails/rails/issues"

issues = []

params = {
    "state": "all",
    "per_page": 100,
}

for i in range(1, 6):
    params["page"] = i
    response = requests.get(url, params=params)
    issues.extend(response.json())

df = pd.DataFrame(issues)
```

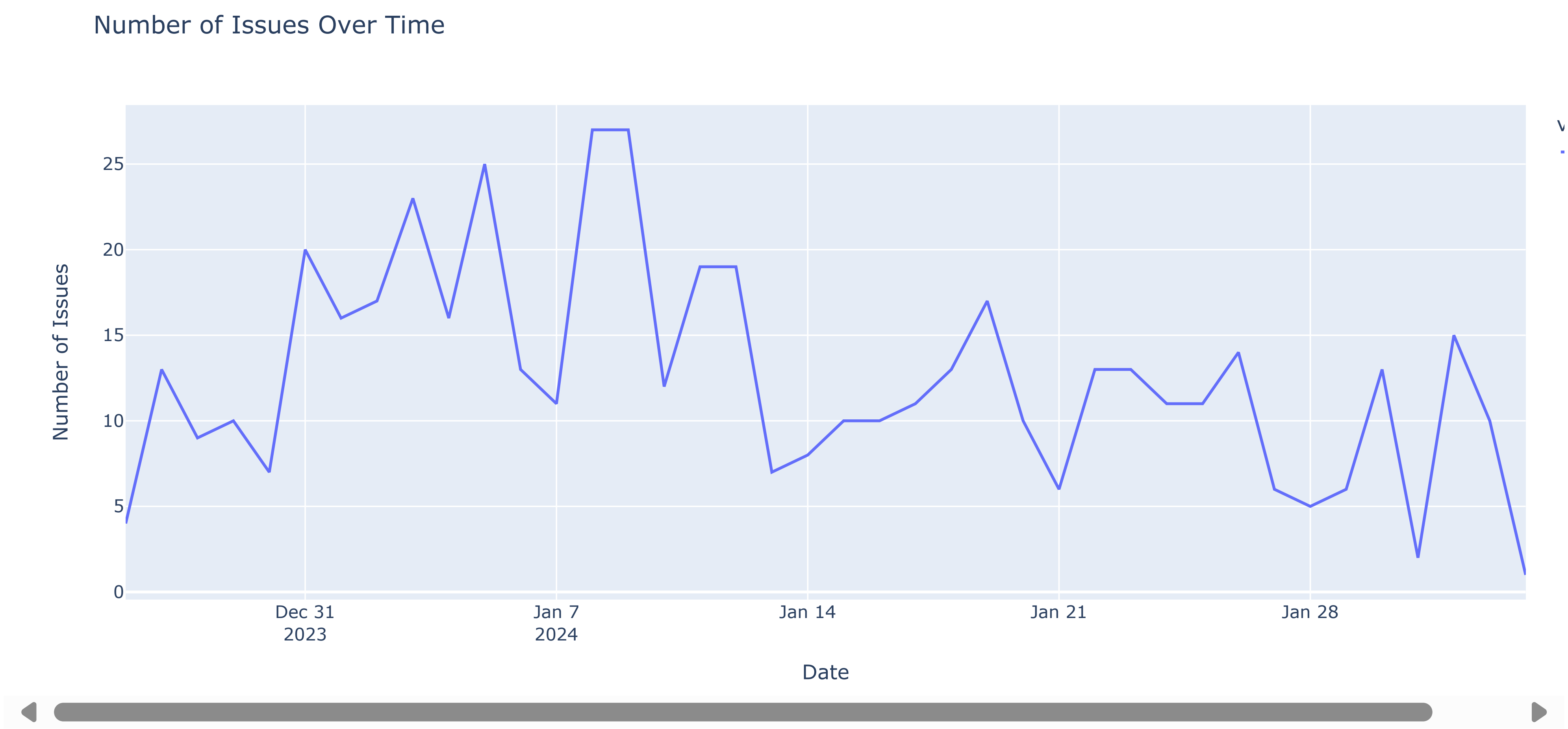
```
df.head()
```

	url	repository_url	labels_url	
0	https://api.github.com/repos/rails/rails/issue...	https://api.github.com/repos/rails/rails	https://api.github.com/repos/rails/rails/issue...	https://api.github.co
1	https://api.github.com/repos/rails/rails/issue...	https://api.github.com/repos/rails/rails	https://api.github.com/repos/rails/rails/issue...	https://api.github.co
2	https://api.github.com/repos/rails/rails/issue...	https://api.github.com/repos/rails/rails	https://api.github.com/repos/rails/rails/issue...	https://api.github.co
3	https://api.github.com/repos/rails/rails/issue...	https://api.github.com/repos/rails/rails	https://api.github.com/repos/rails/rails/issue...	https://api.github.co
4	https://api.github.com/repos/rails/rails/issue...	https://api.github.com/repos/rails/rails	https://api.github.com/repos/rails/rails/issue...	https://api.github.co

5 rows × 30 columns

1. How do the number of issues evolve over time?

```
df['created_at'] = pd.to_datetime(df['created_at'])

issue_counts_by_date = df.resample('D', on='created_at').size()

# Plot
fig = px.line(issue_counts_by_date, title='Number of Issues Over Time')
fig.update_xaxes(title_text='Date')
fig.update_yaxes(title_text='Number of Issues')
fig.show()
```



2. Are there any periods in which we get more issues?

Yes, there are distinct periods where more issues are reported. These periods are represented by the peaks in the line chart. Specifically, from the provided image, it appears that:

- There is a significant peak in the number of issues reported around the first week of January 2024. This could indicate a surge in activity, possibly due to new year code sprints, releases, or other community activities. Another noticeable peak occurs in the third week of January 2024, suggesting another period with increased issue reporting.

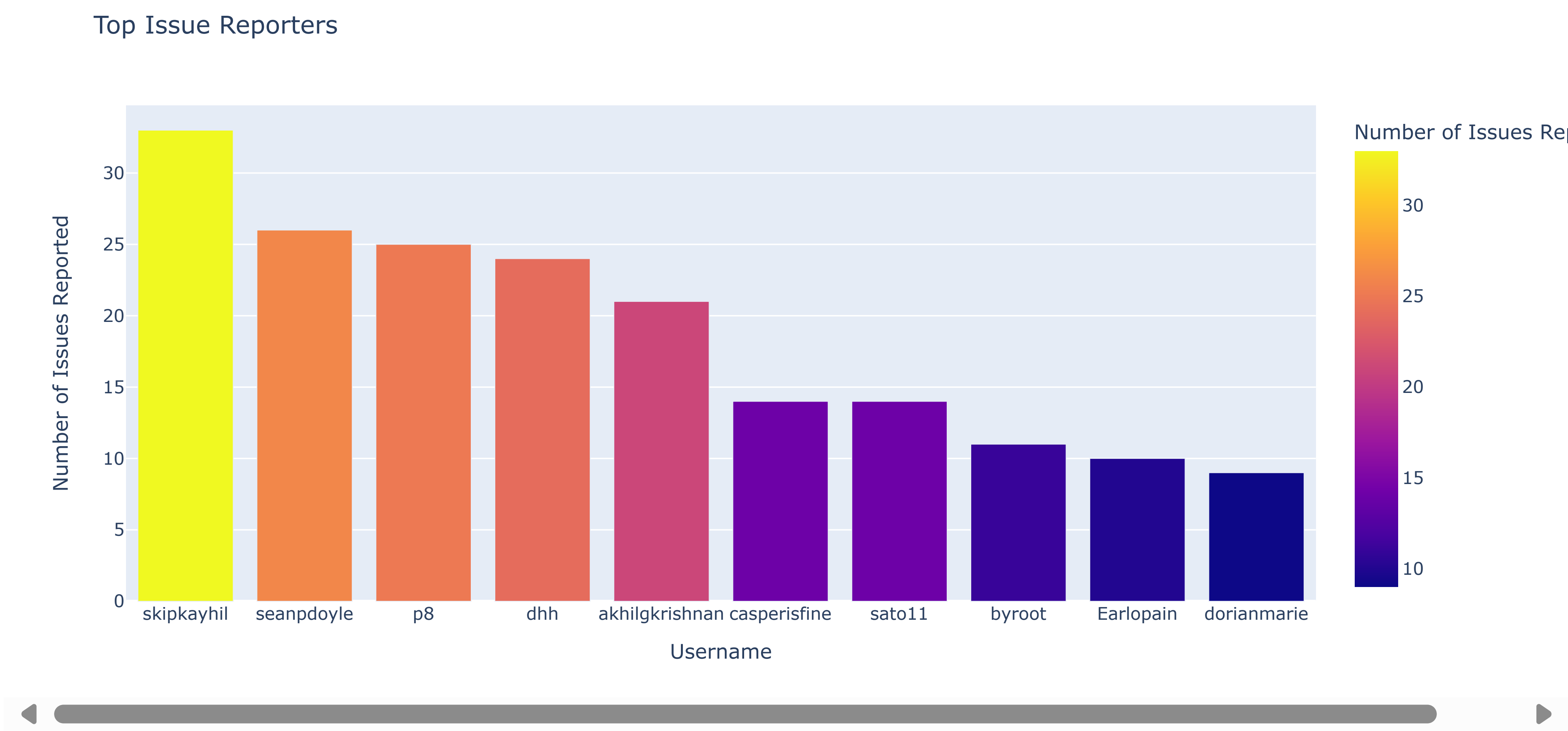
- The final week of January 2024 also shows an increased number of issues, though not as pronounced as the first peak.
- These peaks could be due to various reasons such as new feature deployments, version updates, or discovery of bugs that coincide with these dates. It would be beneficial to cross-reference these dates with the Rails project's update logs, community forums, or other documentation to understand the context behind the increased number of issues.

3. Is there anyone who reports more issues than others?

```
df['reporter'] = df['user'].apply(lambda x: x['login'] if isinstance(x, dict) else None)

top_reporters = df['reporter'].value_counts().head(10)

print(top_reporters)
```

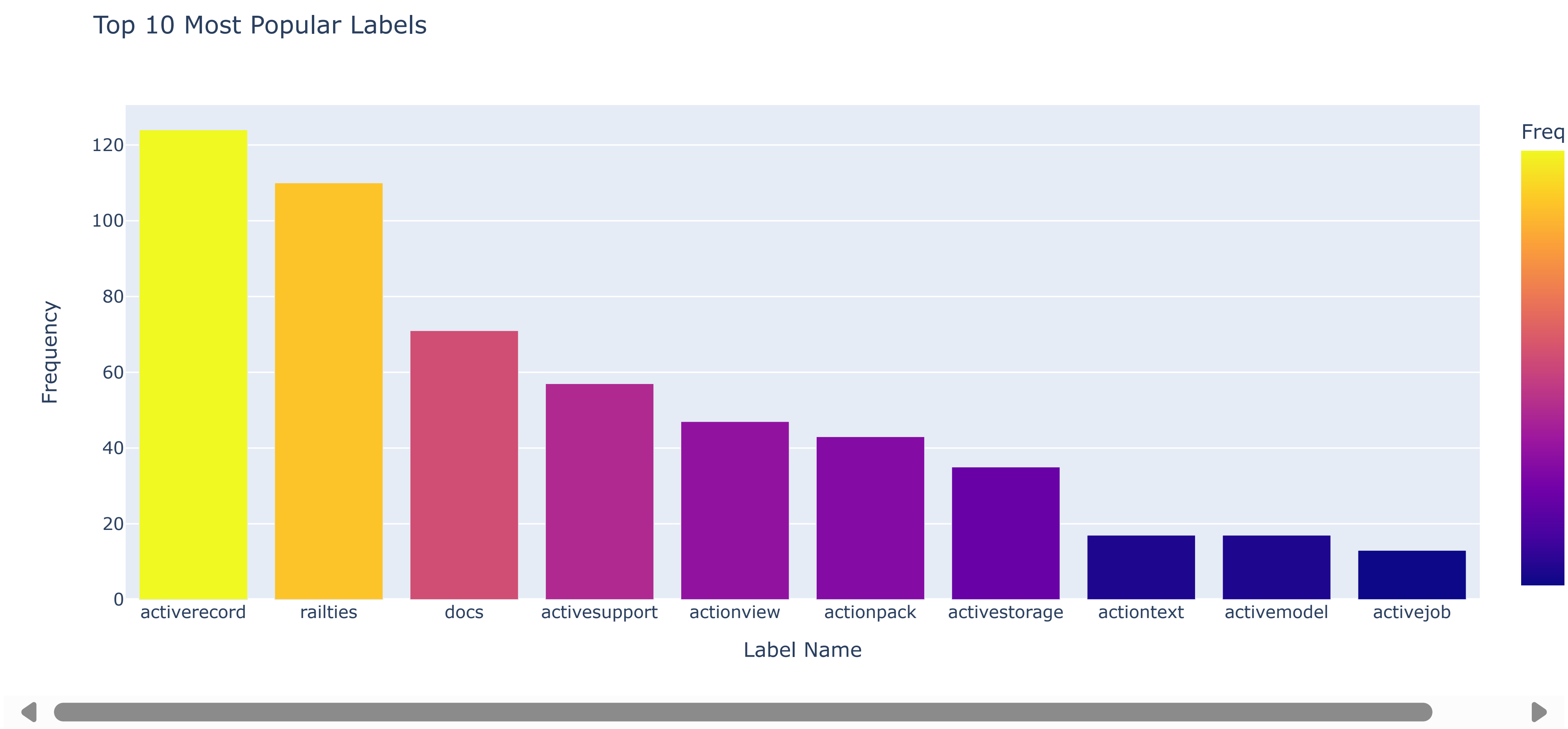
```
skipkayhil        33
seanpdoyle        26
p8                25
dhh               24
akhilgkrishnan    21
casperisfine      14
sato11            14
byroot            11
Earlopain         10
dorianmarie        9
Name: reporter, dtype: int64
```

```
top_reporters_data = top_reporters.reset_index()
top_reporters_data.columns = ['Reporter', 'Frequency']

fig = px.bar(top_reporters_data, x='Reporter', y='Frequency',
             title='Top Issue Reporters',
             labels={'Reporter': 'Username', 'Frequency': 'Number of Issues Reported'},
             color='Frequency')

fig.show()
```



4. What is the most popular category (label)?

```
def extract_label_names(labels):
    return [label['name'] for label in labels if 'name' in label]


df['label_names'] = df['labels'].apply(extract_label_names)
all_label_names = sum(df['label_names'].tolist(), [])
label_counts = Counter(all_label_names)
labels_frequency_df = pd.DataFrame(label_counts.items(), columns=['Label Name', 'Frequency']).sort_values(by='Frequency', ascending=False).reset_



fig = px.bar(labels_frequency_df.head(10),
             x='Label Name',
             y='Frequency',
             title='Top 10 Most Popular Labels',
             labels={'Label Name': 'Label Name', 'Frequency': 'Frequency'},
             color='Frequency',
             )


fig.update_layout(xaxis_title="Label Name",
                  yaxis_title="Frequency",
                  xaxis={'categoryorder':'total descending'}
                  )

fig.show()
```



Most Discussed Issues

```
most_discussed_issues = df.sort_values(by='comments', ascending=False)
print(most_discussed_issues[['title', 'comments']].head(10))
```

```
                                             title  comments
496         Set a new default for the Puma thread count        83
490             Add (a very basic!!) Rubocop by default        30
180   Request: rename Rails console as irr or someth...        23
384               Generate devcontainer files by default        20
```

```
467            Warning on serialize from ActiveStorage        13
492  Extract Action Notifier framework for push not...        13
439                 Default to creating git pre-commit        13
460  Add rate limiting to Action Controller via the...        13
63          Introduce `ActiveSupport::TestCase.around`        11
471  Add Thruster to Docker setup to get HTTP/2, X-...        11
```

```
issue_counts_by_month = df.resample('M', on='created_at').size()

# Plot
fig = px.line(issue_counts_by_month, title='Number of Issues Each Month')
fig.update_xaxes(title_text='Month')
fig.update_yaxes(title_text='Number of Issues')
fig.show()
```

Classification Task

```
import pandas as pd
import torch
from torch.utils.data import Dataset, DataLoader
from sklearn.model_selection import train_test_split
from transformers import AutoTokenizer, AutoModelForSequenceClassification, AdamW, get_linear_schedule_with_warmup, Trainer, TrainingArgument
from sklearn.metrics import accuracy_score, precision_recall_fscore_support
import numpy as np

def get_first_label(label_list):
    if label_list:
        return label_list[0]['name']  # This gets the name of the first label
    else:
        return 'No Label'

df['single_label'] = df['labels'].apply(get_first_label)

# Assuming df is your DataFrame
df['body'] = df['body'].astype(str)  # Ensure text column is string
df['single_label'] = pd.factorize(df['single_label'])[0]

class GitHubIssueDataset(Dataset):
    def __init__(self, encodings, labels):
        self.encodings = encodings
        self.labels = labels

    def __getitem__(self, idx):
        item = {key: torch.tensor(val[idx]) for key, val in self.encodings.items()}
        item['labels'] = torch.tensor(self.labels[idx])
        return item

    def __len__(self):
        return len(self.labels)

# Load tokenizer
tokenizer = AutoTokenizer.from_pretrained('bert-base-uncased')

# Prepare datasets
X_train, X_val, y_train, y_val = train_test_split(df['body'], df['single_label'], test_size=0.2)
train_encodings = tokenizer(X_train.tolist(), truncation=True, padding=True)
val_encodings = tokenizer(X_val.tolist(), truncation=True, padding=True)
train_dataset = GitHubIssueDataset(train_encodings, y_train.tolist())
```

```
val_dataset = GitHubIssueDataset(val_encodings, y_val.tolist())

# Load model
model = AutoModelForSequenceClassification.from_pretrained('bert-base-uncased', num_labels=len(df['single_label'].unique()))

# Training arguments
training_args = TrainingArguments(
    output_dir='./results',
    num_train_epochs=3,
    per_device_train_batch_size=8,
    warmup_steps=500,
    weight_decay=0.01,
    logging_dir='./logs',
    evaluation_strategy="epoch",
    logging_steps=10,
    save_strategy="epoch",
    load_best_model_at_end=True,
)

# Custom compute_metrics function
def compute_metrics(pred):
    labels = pred.label_ids
    preds = pred.predictions.argmax(-1)  # Convert model logits to class predictions
    # Use 'macro', 'micro', or 'weighted' averaging based on your specific needs
    precision, recall, f1, _ = precision_recall_fscore_support(labels, preds, average='macro')
    acc = accuracy_score(labels, preds)
    return {
        'accuracy': acc,
        'f1': f1,
        'precision': precision,
        'recall': recall
    }


# Initialize Trainer
trainer = Trainer(
    model=model,
    args=training_args,
    train_dataset=train_dataset,
    eval_dataset=val_dataset,
    compute_metrics=compute_metrics,
```

```
)

# Train the model
trainer.train()
```