CHEAT SHEET

SCIKIT-LLM

LLMS AS SCIKIT-LEARN ESTIMATORS

SERIES: PYTHON | LEVEL: INTERMEDIATE | UPDATED: 2026

Scikit-LLM wraps language models in the estimator API you already use. Every model below has `fit` and either `predict` or `transform`, so it drops into a `Pipeline` or a cross-validation loop unchanged. What differs is `fit` itself: usually it only records the label set, because the work happens at predict time, one API call per sample. Budget in tokens, not epochs.

Installation

Install Scikit-LLM with:

```
pip install scikit-llm
```

SKLLMConfig

Credentials are set once, globally, so estimators need none of their own. Any estimator can still override them with `key` and `org`.

```
from skllm.config import SKLLMConfig

SKLLMConfig.set_openai_key("<YOUR_KEY>")
SKLLMConfig.set_openai_org("<YOUR_ORG_ID>")
```

skllm.datasets

Small labelled demo sets ship with the library. Enough to check that a prompt and a key work before spending tokens on your own data.

```
from skllm.datasets import (
    get_classification_dataset,
    get_multilabel_classification_dataset)

# labels: positive, negative, neutral
X, y = get_classification_dataset()
```

ZeroShotGPTClassifier

Classifies with no training data at all. `fit` only collects the candidate labels, and those labels carry the task. Keep them descriptive.

```
from skllm.models.gpt.classification.zero_shot \
    import ZeroShotGPTClassifier

clf = ZeroShotGPTClassifier(model="gpt-4o")
clf.fit(None, ["positive", "negative", "neutral"])
labels = clf.predict(X)
```

```
['positive', 'neutral', 'negative']
```

MultiLabelZeroShotGPTClassifier

Assigns several labels to one text. The candidates go in as a single nested list, and `max_labels` caps how many come back.

```
from skllm.models.gpt.classification.zero_shot \
    import MultiLabelZeroShotGPTClassifier

clf = MultiLabelZeroShotGPTClassifier(max_labels=3)
clf.fit(None, [["Quality", "Price", "Delivery"]])
labels = clf.predict(X)
```

FewShotGPTClassifier

Writes the training examples into the prompt. All of them are used, so keep it near ten per class and permute to blunt recency bias.

```
from skllm.models.gpt.classification.few_shot \
    import FewShotGPTClassifier

clf = FewShotGPTClassifier(model="gpt-4o")
clf.fit(X_train, y_train)
labels = clf.predict(X_test)
```

DynamicFewShotGPTClassifier

Retrieves the closest examples per class for each sample instead of sending all of them, which keeps a large training set inside the context window.

```
from skllm.models.gpt.classification.few_shot \
    import DynamicFewShotGPTClassifier

clf = DynamicFewShotGPTClassifier(n_examples=3)
clf.fit(X_train, y_train)
```

CoTGPTClassifier

Asks for the reasoning alongside the label. `predict` hands back two columns, label and explanation, at the cost of many more output tokens.

```
from skllm.models.gpt.classification.zero_shot \
    import CoTGPTClassifier

clf = CoTGPTClassifier(model="gpt-4o").fit(X, y)
pred = clf.predict(X)
labels, reasoning = pred[:, 0], pred[:, 1]
```

GPTClassifier

Fine-tunes the base model in the provider's cloud, so no local GPU is needed. Reach for it only once in-context prompting falls short.

```
from skllm.models.gpt.classification.tunable \
    import GPTClassifier

clf = GPTClassifier(base_model="gpt-3.5-turbo-0613",
                    n_epochs=1)
clf.fit(X_train, y_train)
```

GPTVectorizer

Embeds text of any length to a fixed-width vector that any downstream scikit-learn model can consume. `batch_size` sets texts per request.

```
from skllm.models.gpt.vectorization \
    import GPTVectorizer

vec = GPTVectorizer(model="text-embedding-3-small",
                    batch_size=8)
X_emb = vec.fit_transform(X)
```

```
X_emb.shape
(2000, 1536)
```

Pipeline

The vectorizer is an ordinary transformer, so the LLM is step one and the rest is plain scikit-learn. Every fold re-queries the API.

```
from sklearn.pipeline import Pipeline

pipe = Pipeline([
    ("emb", GPTVectorizer()),
    ("clf", LogisticRegression()),
])
pipe.fit(X_train, y_train)
```

GPTSummarizer

Shortens each document, alone or as a preprocessing step. `max_words` is a soft limit written into the prompt, not enforced afterwards.

```
from skllm.models.gpt.text2text.summarization \
    import GPTSummarizer

summ = GPTSummarizer(model="gpt-4o", max_words=15,
                     focus="pricing")
X_short = summ.fit_transform(X)
```

GPTTranslator

Translates every text into `output_language`. Being a transformer, it fits ahead of a classifier that only saw one language in training.

```
from skllm.models.gpt.text2text.translation \
    import GPTTranslator

t = GPTTranslator(model="gpt-4o",
                  output_language="English")
X_en = t.fit_transform(X)
```

GPTExplainableNER

Tags entities you define yourself and explains each choice. Sparse output is the default: fewer tokens and stricter validation.

```
from skllm.models.gpt.tagging.ner \
    import GPTExplainableNER

entities = {"PERSON": "A name of an individual.",
            "ORG": "A name of a company."}
ner = GPTExplainableNER(entities=entities)
tagged = ner.fit_transform(data)
```

Backend namespaces

A prefix on `model` swaps the backend: Azure, a quantized GGUF file pulled locally, or any OpenAI-compatible URL.

```
ZeroShotGPTClassifier(model="azure::my-deployment")
ZeroShotGPTClassifier(model="gguf::llama3-8b-q4")

SKLLMConfig.set_gpt_url("http://localhost:8000/")
ZeroShotGPTClassifier(model="custom_url::my-model")
```

Evaluation

`predict` returns an ordinary label array, so the usual metrics apply with no adapter. Every fold re-queries the API.

```
from sklearn.metrics import classification_report
from sklearn.model_selection import cross_val_score

print(classification_report(y_test, labels))
cross_val_score(clf, X, y, cv=3)   # 3x the calls
```

```
              precision  recall  f1-score
    positive       0.94    0.91      0.92
```

MINING INSIGHTS IN DATA SCIENCE, MACHINE LEARNING, ANALYTICS, AND AI