Large Language Models

Integrating LLMs into structured NLP pipelines

The `spacy-llm` package integrates Large Language Models (LLMs) into spaCy, featuring a modular system for **fast prototyping** and **prompting**, and turning unstructured responses into **robust outputs** for various NLP tasks, **no training data** required.

Config and implementation

An LLM component is implemented through the `LLMWrapper` class. It is accessible through a generic `llm` component factory as well as through task-specific component factories: `llm_ner`, `llm_spancat`, `llm_rel`, `llm_textcat`, `llm_sentiment`, `llm_summarization`, `llm_entity_linker`, `llm_raw` and `llm_translation`. For these factories, the GPT-3-5 model from OpenAI is used by default, but this can be customized.

LLMWrapper.__init__ METHOD

Create a new pipeline instance. In your application, you would normally use a shortcut for this and instantiate the component using its string name and `nlp.add_pipe` .

NAME	DESCRIPTION
`name`	String name of the component instance. `llm` by default. **TYPE:** `str`
KEYWORD-ONLY	
`vocab`	The shared vocabulary. **TYPE:** `Vocab`
`task`	An LLM Task can generate prompts and parse LLM responses. **TYPE:** `LLMTask`
`model`	The LLM Model queries a specific LLM API.. **TYPE:** `Callable[[Iterable[Any]], Iterable[Any]]`
`cache`	Cache to use for caching prompts and responses per doc. **TYPE:** `Cache`
`save_io`	Whether to save LLM I/O (prompts and responses) in the `Doc._.llm_io` custom attribute. **TYPE:** `bool`

LLMWrapper.__call__ METHOD

Apply the pipe to one document. The document is modified in place and returned. This usually happens under the hood when the `nlp` object is called on a text and all pipeline components are applied to the `Doc` in order.

NAME	DESCRIPTION
`doc`	The document to process. **TYPE:** `Doc`
RETURNS	The processed document. **TYPE:** `Doc`

LLMWrapper.pipe METHOD

Apply the pipe to a stream of documents. This usually happens under the hood when the `nlp` object is called on a text and all pipeline components are applied to the `Doc` in order.

NAME	DESCRIPTION
`docs`	A stream of documents. **TYPE:** `Iterable[Doc]`
KEYWORD-ONLY `batch_size`	The number of documents to buffer. Defaults to `128`. **TYPE:** `int`
YIELDS	The processed documents in order. **TYPE:** `Doc`

LLMWrapper.add_label METHOD

Add a new label to the pipe's task. Alternatively, provide the labels upon the task definition, or through the `[initialize]` block of the config.

NAME	DESCRIPTION
`label`	The label to add. **TYPE:** `str`
RETURNS	`0` if the label is already present, otherwise `1`. **TYPE:** `int`

LLMWrapper.to_disk METHOD

Serialize the pipe to disk.

NAME	DESCRIPTION
`path`	A path to a directory, which will be created if it doesn't exist. Paths may be either strings or `Path`-like objects. **TYPE:** `Union[str, Path]`
KEYWORD-ONLY `exclude`	String names of serialization fields to exclude. **TYPE:** `Iterable[str]`

LLMWrapper.from_disk METHOD

Load the pipe from disk. Modifies the object in place and returns it.

NAME	DESCRIPTION
`path`	A path to a directory. Paths may be either strings or `Path`-like objects. **TYPE:** `Union[str, Path]`
KEYWORD-ONLY `exclude`	String names of serialization fields to exclude. **TYPE:** `Iterable[str]`
RETURNS	The modified `LLMWrapper` object. **TYPE:** `LLMWrapper`

LLMWrapper.to_bytes METHOD

Serialize the pipe to a bytestring.

NAME	DESCRIPTION
KEYWORD-ONLY `exclude`	String names of serialization fields to exclude. TYPE: `Iterable[str]`
RETURNS	The serialized form of the `LLMWrapper` object. TYPE: `bytes`

LLMWrapper.from_bytes METHOD

Load the pipe from a bytestring. Modifies the object in place and returns it.

NAME	DESCRIPTION
`bytes_data`	The data to load from. TYPE: `bytes`
KEYWORD-ONLY `exclude`	String names of serialization fields to exclude. TYPE: `Iterable[str]`
RETURNS	The `LLMWrapper` object. TYPE: `LLMWrapper`

LLMWrapper.labels PROPERTY

The labels currently added to the component. Empty tuple if the LLM’s task does not require labels.

NAME	DESCRIPTION
RETURNS	The labels added to the component. TYPE: `Tuple[str, …]`

Tasks

In `spacy-llm`, a *task* defines an NLP problem or question and its solution using an LLM. It does so by implementing the following responsibilities:

1. Loading a prompt template and injecting documents' data into the prompt. Optionally, include fewshot examples in the prompt.
2. Splitting the prompt into several pieces following a map-reduce paradigm, *if* the prompt is too long to fit into the model's context and the task supports sharding prompts.
3. Parsing the LLM's responses back into structured information and validating the parsed output.

Two different task interfaces are supported: `ShardingLLMTask` and `NonShardingLLMTask`. Only the former supports the sharding of documents, i. e. splitting up prompts if they are too long.

All tasks are registered in the `llm_tasks` registry.

On Sharding

"Sharding" describes, generally speaking, the process of distributing parts of a dataset across multiple storage units for easier processing and lookups. In `spacy-llm` we use this term (synonymously: "mapping") to describe the splitting up of prompts if they are too long for a model to handle, and "fusing" (synonymously: "reducing") to describe how the model responses for several shards are merged back together into a single document.

Prompts are broken up in a manner that *always* keeps the prompt in the template intact, meaning that the instructions to the LLM will always stay complete. The document content however will be split, if the length of the fully rendered prompt exceeds a model context length.

A toy example: let's assume a model has a context window of 25 tokens and the prompt template for our fictional, sharding-supporting task looks like this:

```
Estimate the sentiment of this text:
"{text}"
Estimated sentiment:
```

Depending on how tokens are counted exactly (this is a config setting), we might come up with `n = 12` tokens for the number of tokens in the prompt instructions. Furthermore let's assume that our `text` is

“This has been amazing - I can’t remember the last time I left the cinema so impressed.” - which has roughly 19 tokens.

Considering we only have 13 tokens to add to our prompt before we hit the context limit, we’ll have to split our prompt into two parts. Thus `spacy-llm`, assuming the task used supports sharding, will split the prompt into two (the default splitting strategy splits by tokens, but alternative splitting strategies splitting e. g. by sentences can be configured):

(Prompt 1/2)

```
Estimate the sentiment of this text:
"This has been amazing - I can't remember "
Estimated sentiment:
```

(Prompt 2/2)

```
Estimate the sentiment of this text:
"the last time I left the cinema so impressed."
Estimated sentiment:
```

The reduction step is task-specific - a sentiment estimation task might e. g. do a weighted average of the sentiment scores. Note that prompt sharding introduces potential inaccuracies, as the LLM won’t have access to the entire document at once. Depending on your use case this might or might not be problematic.

`NonShardingLLMTask`

task.generate_prompts

Takes a collection of documents, and returns a collection of “prompts”, which can be of type `Any`. Often, prompts are of type `str` - but this is not enforced to allow for maximum flexibility in the framework.

ARGUMENT	DESCRIPTION
`docs`	The input documents. **TYPE:** `Iterable[Doc]`
RETURNS	The generated prompts. **TYPE:** `Iterable[Any]`

task.parse_responses

Takes a collection of LLM responses and the original documents, parses the responses into structured information, and sets the annotations on the documents. The `parse_responses` function is free to set the annotations in any way, including `Doc` fields like `ents`, `spans` or `cats`, or using custom defined fields.

The `responses` are of type `Iterable[Any]`, though they will often be `str` objects. This depends on the return type of the model.

ARGUMENT	DESCRIPTION
`docs`	The input documents. **TYPE:** `Iterable[Doc]`
`responses`	The responses received from the LLM. **TYPE:** `Iterable[Any]`
RETURNS	The annotated documents. **TYPE:** `Iterable[Doc]`

`ShardingLLMTask`

task.generate_prompts

Takes a collection of documents, breaks them up into shards if necessary to fit all content into the model's context, and returns a collection of collections of "prompts" (i. e. each doc can have multiple shards, each of which have exactly one prompt), which can be of type `Any`. Often, prompts are of type `str` - but this is not enforced to allow for maximum flexibility in the framework.

ARGUMENT	DESCRIPTION
`docs`	The input documents. **TYPE:** `Iterable[Doc]`
RETURNS	The generated prompts. **TYPE:** `Iterable[Iterable[Any]]`

task.parse_responses

Receives a collection of collections of LLM responses (i. e. each doc can have multiple shards, each of which have exactly one prompt / prompt response) and the original shards, parses the responses into structured information, sets the annotations on the shards, and merges back doc shards into single docs. The `parse_responses` function is free to set the annotations in any way, including `Doc` fields like `ents`, `spans` or `cats`, or using custom defined fields.

The `responses` are of type `Iterable[Iterable[Any]]`, though they will often be `str` objects. This depends on the return type of the model.

ARGUMENT	DESCRIPTION
`shards`	The input document shards. **TYPE:** `Iterable[Iterable[Doc]]`
`responses`	The responses received from the LLM. **TYPE:** `Iterable[Iterable[Any]]`
RETURNS	The annotated documents. **TYPE:** `Iterable[Doc]`

Translation

The translation task translates texts from a defined or inferred source to a defined target language.

spacy.Translation.v1

`spacy.Translation.v1` supports both zero-shot and few-shot prompting.

ARGUMENT	DESCRIPTION
`template`	Custom prompt template to send to LLM model. Defaults to translation.v1.jinja </> . TYPE: `str`
`examples`	Optional function that generates examples for few-shot learning. Defaults to `None`. TYPE: `Optional[Callable[[], Iterable[Any]]]`
`parse_responses` (NEW)	Callable for parsing LLM responses for this task. Defaults to the internal parsing method for this task. TYPE: `Optional[TaskResponseParser[TranslationTask]]`
`prompt_example_type` (NEW)	Type to use for fewshot examples. Defaults to `TranslationExample`. TYPE: `Optional[Type[FewshotExample]]`
`source_lang`	Language to translate from. Doesn't have to be set. TYPE: `Optional[str]`
`target_lang`	Language to translate to. No default value, has to be set. TYPE: `str`
`field`	Name of extension attribute to store translation in (i. e. the translation will be available in `doc._.{field}`). Defaults to `translation`. TYPE: `str`

To perform few-shot learning, you can write down a few examples in a separate file, and provide these to be injected into the prompt to the LLM. The default reader `spacy.FewShotReader.v1` supports `.yml`, `.yaml`, `.json` and `.jsonl`.

```yaml
- text: 'Top of the morning to you!'
  translation: '¡Muy buenos días!'
- text: 'The weather is great today.'
  translation: 'El clima está fantástico hoy.'
- text: 'Do you know what will happen tomorrow?'
  translation: '¿Sabes qué pasará mañana?'
```

```
[components.llm.task]
@llm_tasks = "spacy.Translation.v1"
target_lang = "Spanish"
[components.llm.task.examples]
@misc = "spacy.FewShotReader.v1"
path = "translation_examples.yml"
```

Raw prompting

Different to all other tasks `spacy.Raw.vX` doesn't provide a specific prompt, wrapping doc data, to the model. Instead it instructs the model to reply to the doc content. This is handy for use cases like question answering (where each doc contains one question) or if you want to include customized prompts for each doc.

spacy.Raw.v1

Note that since this task may request arbitrary information, it doesn't do any parsing per se - the model response is stored in a custom `Doc` attribute (i. e. can be accessed via `doc._.{field}`).

It supports both zero-shot and few-shot prompting.

ARGUMENT	DESCRIPTION
`template`	Custom prompt template to send to LLM model. Defaults to raw.v1.jinja </> . TYPE: `str`
`examples`	Optional function that generates examples for few-shot learning. Defaults to `None`. TYPE: `Optional[Callable[[], Iterable[Any]]]`
`parse_responses`	Callable for parsing LLM responses for this task. Defaults to the internal parsing method for this task. TYPE: `Optional[TaskResponseParser[RawTask]]`
`prompt_example_type`	Type to use for fewshot examples. Defaults to `RawExample`. TYPE: `Optional[Type[FewshotExample]]`
`field`	Name of extension attribute to store model reply in (i. e. the reply will be available in `doc._.{field}`). Defaults to `reply`. TYPE: `str`

To perform few-shot learning, you can write down a few examples in a separate file, and provide these to be injected into the prompt to the LLM. The default reader `spacy.FewShotReader.v1` supports `.yml`, `.yaml`, `.json` and `.jsonl`.

```
# Each example can follow an arbitrary pattern. It might help the prompt performance t
# the actual docs' content.
- text: "3 + 5 = x. What's x?"
  reply: '8'

- text: 'Write me a limerick.'
  reply:
    "There was an Old Man with a beard, Who said, 'It is just as I feared! Two
    Owls and a Hen, Four Larks and a Wren, Have all built their nests in my
    beard!"

- text: "Analyse the sentiment of the text 'This is great'."
  reply: "'This is great' expresses a very positive sentiment."
```

```
[components.llm.task]
@llm_tasks = "spacy.Raw.v1"
field = "llm_reply"
[components.llm.task.examples]
@misc = "spacy.FewShotReader.v1"
path = "raw_examples.yml"
```

Summarization

A summarization task takes a document as input and generates a summary that is stored in an extension attribute.

spacy.Summarization.v1

The `spacy.Summarization.v1` task supports both zero-shot and few-shot prompting.

ARGUMENT	DESCRIPTION
`template`	Custom prompt template to send to LLM model. Defaults to summarization.v1.jinja. **TYPE:** `str`
`examples`	Optional function that generates examples for few-shot learning. Defaults to `None`. **TYPE:** `Optional[Callable[[], Iterable[Any]]]`
`parse_responses` (NEW)	Callable for parsing LLM responses for this task. Defaults to the internal parsing method for this task. **TYPE:** `Optional[TaskResponseParser[SummarizationTask]]`
`prompt_example_type` (NEW)	Type to use for fewshot examples. Defaults to `SummarizationExample`. **TYPE:** `Optional[Type[FewshotExample]]`
`max_n_words`	Maximum number of words to be used in summary. Note that this should not expected to work exactly. Defaults to `None`. **TYPE:** `Optional[int]`
`field`	Name of extension attribute to store summary in (i. e. the summary will be available in `doc._.{field}`). Defaults to `summary`. **TYPE:** `str`

The summarization task prompts the model for a concise summary of the provided text. It optionally allows to limit the response to a certain number of tokens - note that this requirement will be included in the prompt, but the task doesn’t perform a hard cut-off. It’s hence possible that your summary exceeds `max_n_words`.

To perform few-shot learning, you can write down a few examples in a separate file, and provide these to be injected into the prompt to the LLM. The default reader `spacy.FewShotReader.v1` supports `.yml`, `.yaml`, `.json` and `.jsonl`.

```
- text: >
    The United Nations, referred to informally as the UN, is an
    intergovernmental organization whose stated purposes are  to maintain
    international peace and security, develop friendly relations among nations,
    achieve international cooperation, and serve as a centre for harmonizing the
    actions of nations. It is the world's largest international organization.
    The UN is headquartered on international territory in New York City, and the
    organization has other offices in Geneva, Nairobi, Vienna, and The Hague,
    where the International Court of Justice is headquartered.\n\n The UN was
    established after World War II with the aim of preventing future world wars,
    and succeeded the League of  Nations, which was characterized as
    ineffective.
  summary:
    'The UN is an international organization that promotes global peace,
    cooperation, and harmony. Established after WWII, its purpose is to prevent
    future world wars.'
```

```
[components.llm.task]
@llm_tasks = "spacy.Summarization.v1"
max_n_words = 20
[components.llm.task.examples]
@misc = "spacy.FewShotReader.v1"
path = "summarization_examples.yml"
```

EL (Entity Linking)

The EL links recognized entities (see NER) to those in a knowledge base (KB). The EL task prompts the LLM to select the most likely candidate from the KB, whose structure can be arbitrary.

Note that the documents processed by the entity linking task are expected to have recognized entities in their `.ents` attribute. This can be achieved by either running the NER task, using a trained spaCy NER model or setting the entities manually prior to running the EL task.

In order to be able to pull data from the KB, an object implementing the `CandidateSelector` protocol has to be provided. This requires two functions: (1) `__call__()` to fetch candidate entities for entity mentions in the text (assumed to be available in `Doc.ents`) and (2) `get_entity_description()` to fetch descriptions for any given entity ID. Descriptions can be empty, but ideally provide more context for entities stored in the KB.

`spacy-llm` provides a `CandidateSelector` implementation (`spacy.CandidateSelector.v1`) that leverages a spaCy knowledge base - as used in an `entity_linking` component - to select candidates. This knowledge base can be loaded from an existing spaCy pipeline (note that the pipeline's EL component doesn't have to be trained) or from a separate .yaml file.

spacy.EntityLinker.v1

Supports zero- and few-shot prompting. Relies on a configurable component suggesting viable entities before letting the LLM pick the most likely candidate.

ARGUMENT	DESCRIPTION
`template`	Custom prompt template to send to LLM model. Defaults to entity_linker.v1.jinja </> . **TYPE:** `str`
`parse_responses`	Callable for parsing LLM responses for this task. Defaults to the internal parsing method for this task. **TYPE:** `Optional[TaskResponseParser[EntityLinkerTask]]`
`prompt_example_type`	Type to use for fewshot examples. Defaults to `ELExample`. **TYPE:** `Optional[Type[FewshotExample]]`
`examples`	Optional callable that reads a file containing task examples for few-shot learning. If `None` is passed, zero-shot learning will be used. Defaults to `None`. **TYPE:** `ExamplesConfigType`
`scorer`	Scorer function. Defaults to the metric used by spaCy to evaluate entity linking performance. **TYPE:** `Optional[Scorer]`

spacy.CandidateSelector.v1

`spacy.CandidateSelector.v1` is an implementation of the `CandidateSelector` protocol required by `spacy.EntityLinker.v1`. The built-in candidate selector method allows loading existing knowledge bases in several ways, e. g. loading from a spaCy pipeline with a (not necessarily trained) entity linking component, and loading from a file describing the knowlege base as a .yaml file. Either way the loaded data will be converted to a spaCy `InMemoryLookupKB` instance. The KB's selection capabilities are used to select the most likely entity candidates for the specified mentions.

ARGUMENT	DESCRIPTION
`kb_loader`	KB loader object. **TYPE:** `InMemoryLookupKBLoader`
`top_n`	Top-n candidates to include in the prompt. Defaults to 5. **TYPE:** `int`

spacy.KBObjectLoader.v1

Adheres to the `InMemoryLookupKBLoader` interface required by `spacy.CandidateSelector.v1`. Loads a knowledge base from an existing spaCy pipeline.

ARGUMENT	DESCRIPTION
`path`	Path to KB file. **TYPE:** `Union[str,Path]`
`nlp_path`	Path to serialized NLP pipeline. If None, path will be guessed. **TYPE:** `Optional[Union[Path, str]]`
`desc_path`	Path to file with descriptions for entities. **TYPE:** `int`
`ent_desc_reader`	Entity description reader. Defaults to an internal method expecting a CSV file without header row, with ";" as delimiters, and with two columns - one for the entitys' IDs, one for their descriptions. **TYPE:** `Optional[EntDescReader]`

spacy.KBFileLoader.v1

Adheres to the `InMemoryLookupKBLoader` interface required by `spacy.CandidateSelector.v1`. Loads a knowledge base from a knowledge base file. The KB .yaml file has to stick to the following format:

```
entities:
  # The key should be whatever ID identifies this entity uniquely in your knowledge ba
  ID1:
      name: "..."
      desc: "..."
  ID2:
      ...
# Data on aliases in your knowledge base - e. g. "Apple" for the entity "Apple Inc.".
aliases:
  - alias: "..."
    # List of all entities that this alias refers to.
    entities: ["ID1", "ID2", ...]
    # Optional: prior probabilities that this alias refers to the n-th entity in the "
    probabilities: [0.5, 0.2, ...]
  - alias: "..."
    entities: [...]
    probabilities: [...]
  ...
```

See here </> for a toy example of how such a KB file might look like.

ARGUMENT	DESCRIPTION
`path`	Path to KB file. **TYPE:** `Union[str,Path]`

NER

The NER task identifies non-overlapping entities in text.

spacy.NER.v3

Version 3 is fundamentally different to v1 and v2, as it implements Chain-of-Thought prompting, based on the PromptNER paper by Ashok and Lipton (2023). On an internal use-case, we have found this implementation to obtain significant better accuracy - with an increase of F-score of up to 15 percentage points.

When no examples are specified, the v3 implementation will use a dummy example in the prompt. Technically this means that the task will always perform few-shot prompting under the hood.

ARGUMENT	DESCRIPTION
`template`	Custom prompt template to send to LLM model. Defaults to ner.v3.jinja </> . **TYPE:** `str`
`examples`	Optional function that generates examples for few-shot learning. Defaults to `None` . **TYPE:** `Optional[Callable[[], Iterable[Any]]]`
`parse_responses` (NEW)	Callable for parsing LLM responses for this task. Defaults to the internal parsing method for this task. **TYPE:** `Optional[TaskResponseParser[NERTask]]`
`prompt_example_type` (NEW)	Type to use for fewshot examples. Defaults to `NERExample` . **TYPE:** `Optional[Type[FewshotExample]]`
`scorer`	Scorer function that evaluates the task performance on provided examples. Defaults to the metric used by spaCy. **TYPE:** `Optional[Scorer]`
`labels`	List of labels or str of comma-separated list of labels. **TYPE:** `Union[List[str], str]`
`label_definitions`	Optional dict mapping a label to a description of that label. These descriptions are added to the prompt to help instruct the LLM on what to extract. Defaults to `None` . **TYPE:** `Optional[Dict[str, str]]`
`description` (NEW)	A description of what to recognize or not recognize as entities. **TYPE:** `str`
`normalizer`	Function that normalizes the labels as returned by the LLM. If `None` , defaults to `spacy.LowercaseNormalizer.v1` . Defaults to `None` . **TYPE:** `Optional[Callable[[str], str]]`
`alignment_mode`	Alignment mode in case the LLM returns entities that do not align with token boundaries. Options are `"strict"` , `"contract"` or `"expand"` . Defaults to `"contract"` . **TYPE:** `str`
`case_sensitive_matching`	Whether to search without case sensitivity. Defaults to `False` . **TYPE:** `bool`

Note that the `single_match` parameter, used in v1 and v2, is not supported anymore, as the CoT parsing algorithm takes care of this automatically.

New to v3 is the fact that you can provide an explicit description of what entities should look like. You can use this feature in addition to `label_definitions`.

```
[components.llm.task]
@llm_tasks = "spacy.NER.v3"
labels = ["DISH", "INGREDIENT", "EQUIPMENT"]
description = Entities are the names food dishes,
    ingredients, and any kind of cooking equipment.
    Adjectives, verbs, adverbs are not entities.
    Pronouns are not entities.

[components.llm.task.label_definitions]
DISH = "Known food dishes, e.g. Lobster Ravioli, garlic bread"
INGREDIENT = "Individual parts of a food dish, including herbs and spices."
EQUIPMENT = "Any kind of cooking equipment. e.g. oven, cooking pot, grill"
```

To perform few-shot learning, you can write down a few examples in a separate file, and provide these to be injected into the prompt to the LLM. The default reader `spacy.FewShotReader.v1` supports `.yml`, `.yaml`, `.json` and `.jsonl`.

While not required, this task works best when both positive and negative examples are provided. The format is different than the files required for v1 and v2, as additional fields such as `is_entity` and `reason` should now be provided.

```
[
  {
    "text": "You can't get a great chocolate flavor with carob.",
    "spans": [
      {
        "text": "chocolate",
        "is_entity": false,
        "label": "==NONE==",
        "reason": "is a flavor in this context, not an ingredient"
      },
      {
        "text": "carob",
        "is_entity": true,
        "label": "INGREDIENT",
        "reason": "is an ingredient to add chocolate flavor"
      }
    ]
  },
  ...
]
```

```
[components.llm.task.examples]
@misc = "spacy.FewShotReader.v1"
path = "${paths.examples}"
```

For a fully working example, see this usage example </> .

spacy.NER.v2

This version supports explicitly defining the provided labels with custom descriptions, and further supports zero-shot and few-shot prompting just like v1.

ARGUMENT	DESCRIPTION
`template` (NEW)	Custom prompt template to send to LLM model. Defaults to ner.v2.jinja </> . **TYPE:** `str`
`examples`	Optional function that generates examples for few-shot learning. Defaults to `None` . **TYPE:** `Optional[Callable[[], Iterable[Any]]]`
`parse_responses` (NEW)	Callable for parsing LLM responses for this task. Defaults to the internal parsing method for this task. **TYPE:** `Optional[TaskResponseParser[NERTask]]`
`prompt_example_type` (NEW)	Type to use for fewshot examples. Defaults to `NERExample` . **TYPE:** `Optional[Type[FewshotExample]]`
`scorer` (NEW)	Scorer function that evaluates the task performance on provided examples. Defaults to the metric used by spaCy. **TYPE:** `Optional[Scorer]`
`labels`	List of labels or str of comma-separated list of labels. **TYPE:** `Union[List[str], str]`
`label_definitions` (NEW)	Optional dict mapping a label to a description of that label. These descriptions are added to the prompt to help instruct the LLM on what to extract. Defaults to `None` . **TYPE:** `Optional[Dict[str, str]]`
`normalizer`	Function that normalizes the labels as returned by the LLM. If `None` , defaults to `spacy.LowercaseNormalizer.v1` . Defaults to `None` . **TYPE:** `Optional[Callable[[str], str]]`
`alignment_mode`	Alignment mode in case the LLM returns entities that do not align with token boundaries. Options are `"strict"` , `"contract"` or `"expand"` . Defaults to `"contract"` . **TYPE:** `str`
`case_sensitive_matching`	Whether to search without case sensitivity. Defaults to `False` . **TYPE:** `bool`
`single_match`	Whether to match an entity in the LLM's response only once (the first hit) or multiple times. Defaults to `False` . **TYPE:** `bool`

The parameters `alignment_mode`, `case_sensitive_matching` and `single_match` are identical to the v1 implementation. The format of few-shot examples are also the same.

New to v2 is the fact that you can write definitions for each label and provide them via the `label_definitions` argument. This lets you tell the LLM exactly what you're looking for rather than relying on the LLM to interpret its task given just the label name. Label descriptions are freeform so you can write whatever you want here, but a brief description along with some examples and counter examples seems to work quite well.

```
[components.llm.task]
@llm_tasks = "spacy.NER.v2"
labels = PERSON,SPORTS_TEAM

[components.llm.task.label_definitions]
PERSON = "Extract any named individual in the text."
SPORTS_TEAM = "Extract the names of any professional sports team. e.g. Golden State Wa
```

For a fully working example, see this usage example </> .

spacy.NER.v1

The original version of the built-in NER task supports both zero-shot and few-shot prompting.

ARGUMENT	DESCRIPTION
`examples`	Optional function that generates examples for few-shot learning. Defaults to `None`. **TYPE:** `Optional[Callable[[], Iterable[Any]]]`
`parse_responses` (NEW)	Callable for parsing LLM responses for this task. Defaults to the internal parsing method for this task. **TYPE:** `Optional[TaskResponseParser[NERTask]]`
`prompt_example_type` (NEW)	Type to use for fewshot examples. Defaults to `NERExample`. **TYPE:** `Optional[Type[FewshotExample]]`
`scorer` (NEW)	Scorer function that evaluates the task performance on provided examples. Defaults to the metric used by spaCy. **TYPE:** `Optional[Scorer]`
`labels`	Comma-separated list of labels. **TYPE:** `str`
`normalizer`	Function that normalizes the labels as returned by the LLM. If `None`, defaults to `spacy.LowercaseNormalizer.v1`. **TYPE:** `Optional[Callable[[str], str]]`
`alignment_mode`	Alignment mode in case the LLM returns entities that do not align with token boundaries. Options are `"strict"`, `"contract"` or `"expand"`. Defaults to `"contract"`. **TYPE:** `str`
`case_sensitive_matching`	Whether to search without case sensitivity. Defaults to `False`. **TYPE:** `bool`
`single_match`	Whether to match an entity in the LLM's response only once (the first hit) or multiple times. Defaults to `False`. **TYPE:** `bool`

The NER task implementation doesn't currently ask the LLM for specific offsets, but simply expects a list of strings that represent the enties in the document. This means that a form of string matching is required. This can be configured by the following parameters:

- The `single_match` parameter is typically set to `False` to allow for multiple matches. For instance, the response from the LLM might only mention the entity "Paris" once, but you'd still want to mark it every time it occurs in the document.

- The case-sensitive matching is typically set to `False` to be robust against case variances in the LLM's output.
- The `alignment_mode` argument is used to match entities as returned by the LLM to the tokens from the original `Doc` - specifically it's used as argument in the call to `doc.char_span()` ≣ . The `"strict"` mode will only keep spans that strictly adhere to the given token boundaries. `"contract"` will only keep those tokens that are fully within the given range, e.g. reducing `"New Y"` to `"New"`. Finally, `"expand"` will expand the span to the next token boundaries, e.g. expanding `"New Y"` out to `"New York"`.

To perform few-shot learning, you can write down a few examples in a separate file, and provide these to be injected into the prompt to the LLM. The default reader `spacy.FewShotReader.v1` supports `.yml`, `.yaml`, `.json` and `.jsonl`.

```
- text: Jack and Jill went up the hill.
  entities:
    PERSON:
      - Jack
      - Jill
    LOCATION:
      - hill
- text: Jack fell down and broke his crown.
  entities:
    PERSON:
      - Jack
```

```
[components.llm.task.examples]
@misc = "spacy.FewShotReader.v1"
path = "ner_examples.yml"
```

SpanCat

The SpanCat task identifies potentially overlapping entities in text.

spacy.SpanCat.v3

The built-in SpanCat v3 task is a simple adaptation of the NER v3 task to support overlapping entities and store its annotations in `doc.spans`.

ARGUMENT	DESCRIPTION
`template`	Custom prompt template to send to LLM model. Defaults to `spancat.v3.jinja` </> . **TYPE:** `str`
`examples`	Optional function that generates examples for few-shot learning. Defaults to `None` . **TYPE:** `Optional[Callable[[], Iterable[Any]]]`
`parse_responses` (NEW)	Callable for parsing LLM responses for this task. Defaults to the internal parsing method for this task. **TYPE:** `Optional[TaskResponseParser[SpanCatTask]]`
`prompt_example_type` (NEW)	Type to use for fewshot examples. Defaults to `SpanCatExample` . **TYPE:** `Optional[Type[FewshotExample]]`
`scorer` (NEW)	Scorer function that evaluates the task performance on provided examples. Defaults to the metric used by spaCy. **TYPE:** `Optional[Scorer]`
`labels`	List of labels or str of comma-separated list of labels. **TYPE:** `Union[List[str], str]`
`label_definitions`	Optional dict mapping a label to a description of that label. These descriptions are added to the prompt to help instruct the LLM on what to extract. Defaults to `None` . **TYPE:** `Optional[Dict[str, str]]`
`description` (NEW)	A description of what to recognize or not recognize as entities. **TYPE:** `str`
`spans_key`	Key of the `Doc.spans` dict to save the spans under. Defaults to `"sc"` . **TYPE:** `str`
`normalizer`	Function that normalizes the labels as returned by the LLM. If `None` , defaults to `spacy.LowercaseNormalizer.v1` . **TYPE:** `Optional[Callable[[str], str]]`
`alignment_mode`	Alignment mode in case the LLM returns entities that do not align with token boundaries. Options are `"strict"` , `"contract"` or `"expand"` . Defaults to `"contract"` . **TYPE:** `str`
`case_sensitive_matching`	Whether to search without case sensitivity. Defaults to `False`

`case_sensitive_matching`	Whether to search without case sensitivity. Defaults to `False`. **TYPE:** `bool`

Note that the `single_match` parameter, used in v1 and v2, is not supported anymore, as the CoT parsing algorithm takes care of this automatically.

spacy.SpanCat.v2

The built-in SpanCat v2 task is a simple adaptation of the NER v2 task to support overlapping entities and store its annotations in `doc.spans`.

ARGUMENT	DESCRIPTION
`template` (NEW)	Custom prompt template to send to LLM model. Defaults to `spancat.v2.jinja` </> . **TYPE:** `str`
`examples`	Optional function that generates examples for few-shot learning. Defaults to `None` . **TYPE:** `Optional[Callable[[], Iterable[Any]]]`
`parse_responses` (NEW)	Callable for parsing LLM responses for this task. Defaults to the internal parsing method for this task. **TYPE:** `Optional[TaskResponseParser[SpanCatTask]]`
`prompt_example_type` (NEW)	Type to use for fewshot examples. Defaults to `SpanCatExample` . **TYPE:** `Optional[Type[FewshotExample]]`
`scorer` (NEW)	Scorer function that evaluates the task performance on provided examples. Defaults to the metric used by spaCy. **TYPE:** `Optional[Scorer]`
`labels`	List of labels or str of comma-separated list of labels. **TYPE:** `Union[List[str], str]`
`label_definitions` (NEW)	Optional dict mapping a label to a description of that label. These descriptions are added to the prompt to help instruct the LLM on what to extract. Defaults to `None` . **TYPE:** `Optional[Dict[str, str]]`
`spans_key`	Key of the `Doc.spans` dict to save the spans under. Defaults to `"sc"` . **TYPE:** `str`
`normalizer`	Function that normalizes the labels as returned by the LLM. If `None` , defaults to `spacy.LowercaseNormalizer.v1` . **TYPE:** `Optional[Callable[[str], str]]`
`alignment_mode`	Alignment mode in case the LLM returns entities that do not align with token boundaries. Options are `"strict"` , `"contract"` or `"expand"` . Defaults to `"contract"` . **TYPE:** `str`
`case_sensitive_matching`	Whether to search without case sensitivity. Defaults to `False` . **TYPE:** `bool`
`single_match`	Whether to match an entity in the LLM's response only once (the first hit) or

`single_match`	Whether to match an entity in the LLM's response only once (the first hit) or multiple times. Defaults to `False`. TYPE: `bool`

Except for the `spans_key` parameter, the SpanCat v2 task reuses the configuration from the NER v2 task. Refer to its documentation for more insight.

spacy.SpanCat.v1

The original version of the built-in SpanCat task is a simple adaptation of the v1 NER task to support overlapping entities and store its annotations in `doc.spans`.

ARGUMENT	DESCRIPTION
`examples`	Optional function that generates examples for few-shot learning. Defaults to `None`. **TYPE:** `Optional[Callable[[], Iterable[Any]]]`
`parse_responses` (NEW)	Callable for parsing LLM responses for this task. Defaults to the internal parsing method for this task. **TYPE:** `Optional[TaskResponseParser[SpanCatTask]]`
`prompt_example_type` (NEW)	Type to use for fewshot examples. Defaults to `SpanCatExample`. **TYPE:** `Optional[Type[FewshotExample]]`
`scorer` (NEW)	Scorer function that evaluates the task performance on provided examples. Defaults to the metric used by spaCy. **TYPE:** `Optional[Scorer]`
`labels`	Comma-separated list of labels. **TYPE:** `str`
`spans_key`	Key of the `Doc.spans` dict to save the spans under. Defaults to `"sc"`. **TYPE:** `str`
`normalizer`	Function that normalizes the labels as returned by the LLM. If `None`, defaults to `spacy.LowercaseNormalizer.v1`. **TYPE:** `Optional[Callable[[str], str]]`
`alignment_mode`	Alignment mode in case the LLM returns entities that do not align with token boundaries. Options are `"strict"`, `"contract"` or `"expand"`. Defaults to `"contract"`. **TYPE:** `str`
`case_sensitive_matching`	Whether to search without case sensitivity. Defaults to `False`. **TYPE:** `bool`
`single_match`	Whether to match an entity in the LLM's response only once (the first hit) or multiple times. Defaults to `False`. **TYPE:** `bool`

Except for the `spans_key` parameter, the SpanCat v1 task reuses the configuration from the NER v1 task. Refer to its documentation for more insight.

TextCat

The TextCat task labels documents with relevant categories.

spacy.TextCat.v3

On top of the functionality from v2, version 3 of the built-in TextCat tasks allows setting definitions of labels. Those definitions are included in the prompt.

ARGUMENT	DESCRIPTION
`template`	Custom prompt template to send to LLM model. Defaults to `textcat.v3.jinja` </> . **TYPE:** `str`
`examples`	Optional function that generates examples for few-shot learning. Defaults to `None` . **TYPE:** `Optional[Callable[[], Iterable[Any]]]`
`parse_responses` (NEW)	Callable for parsing LLM responses for this task. Defaults to the internal parsing method for this task. **TYPE:** `Optional[TaskResponseParser[SpanCatTask]]`
`prompt_example_type` (NEW)	Type to use for fewshot examples. Defaults to `TextCatExample` . **TYPE:** `Optional[Type[FewshotExample]]`
`scorer` (NEW)	Scorer function that evaluates the task performance on provided examples. Defaults to the metric used by spaCy. **TYPE:** `Optional[Scorer]`
`labels`	List of labels or str of comma-separated list of labels. **TYPE:** `Union[List[str], str]`
`label_definitions` (NEW)	Dictionary of label definitions. Included in the prompt, if set. Defaults to `None` . **TYPE:** `Optional[Dict[str, str]]`
`normalizer`	Function that normalizes the labels as returned by the LLM. If `None` , falls back to `spacy.LowercaseNormalizer.v1` . Defaults to `None` . **TYPE:** `Optional[Callable[[str], str]]`
`exclusive_classes`	If set to `True` , only one label per document should be valid. If set to `False` , one document can have multiple labels. Defaults to `False` . **TYPE:** `bool`
`allow_none`	When set to `True` , allows the LLM to not return any of the given label. The resulting dict in `doc.cats` will have `0.0` scores for all labels. Defaults to `True` . **TYPE:** `bool`
`verbose`	If set to `True` , warnings will be generated when the LLM returns invalid responses. Defaults to `False` . **TYPE:** `bool`

The formatting of few-shot examples is the same as those for the v1 implementation.

spacy.TextCat.v2

V2 includes all v1 functionality, with an improved prompt template.

ARGUMENT	DESCRIPTION
`template` (NEW)	Custom prompt template to send to LLM model. Defaults to `textcat.v2.jinja` </> . **TYPE:** `str`
`examples`	Optional function that generates examples for few-shot learning. Defaults to `None` . **TYPE:** `Optional[Callable[[], Iterable[Any]]]`
`parse_responses` (NEW)	Callable for parsing LLM responses for this task. Defaults to the internal parsing method for this task. **TYPE:** `Optional[TaskResponseParser[SpanCatTask]]`
`prompt_example_type` (NEW)	Type to use for fewshot examples. Defaults to `TextCatExample` . **TYPE:** `Optional[Type[FewshotExample]]`
`scorer` (NEW)	Scorer function that evaluates the task performance on provided examples. Defaults to the metric used by spaCy. **TYPE:** `Optional[Scorer]`
`labels`	List of labels or str of comma-separated list of labels. **TYPE:** `Union[List[str], str]`
`normalizer`	Function that normalizes the labels as returned by the LLM. If `None` , falls back to `spacy.LowercaseNormalizer.v1` . **TYPE:** `Optional[Callable[[str], str]]`
`exclusive_classes`	If set to `True` , only one label per document should be valid. If set to `False` , one document can have multiple labels. Defaults to `False` . **TYPE:** `bool`
`allow_none`	When set to `True` , allows the LLM to not return any of the given label. The resulting dict in `doc.cats` will have `0.0` scores for all labels. Defaults to `True` . **TYPE:** `bool`
`verbose`	If set to `True` , warnings will be generated when the LLM returns invalid responses. Defaults to `False` . **TYPE:** `bool`

The formatting of few-shot examples is the same as those for the v1 implementation.

spacy.TextCat.v1

Version 1 of the built-in TextCat task supports both zero-shot and few-shot prompting.

ARGUMENT	DESCRIPTION
`examples`	Optional function that generates examples for few-shot learning. Deafults to `None`. **TYPE:** `Optional[Callable[[], Iterable[Any]]]`
`parse_responses` (NEW)	Callable for parsing LLM responses for this task. Defaults to the internal parsing method for this task. **TYPE:** `Optional[TaskResponseParser[SpanCatTask]]`
`prompt_example_type` (NEW)	Type to use for fewshot examples. Defaults to `TextCatExample`. **TYPE:** `Optional[Type[FewshotExample]]`
`scorer` (NEW)	Scorer function that evaluates the task performance on provided examples. Defaults to the metric used by spaCy. **TYPE:** `Optional[Scorer]`
`labels`	Comma-separated list of labels. **TYPE:** `str`
`normalizer`	Function that normalizes the labels as returned by the LLM. If `None`, falls back to `spacy.LowercaseNormalizer.v1`. **TYPE:** `Optional[Callable[[str], str]]`
`exclusive_classes`	If set to `True`, only one label per document should be valid. If set to `False`, one document can have multiple labels. Defaults to `False`. **TYPE:** `bool`
`allow_none`	When set to `True`, allows the LLM to not return any of the given label. The resulting dict in `doc.cats` will have `0.0` scores for all labels. Defaults to `True`. **TYPE:** `bool`
`verbose`	If set to `True`, warnings will be generated when the LLM returns invalid responses. Defaults to `False`. **TYPE:** `bool`

To perform few-shot learning, you can write down a few examples in a separate file, and provide these to be injected into the prompt to the LLM. The default reader `spacy.FewShotReader.v1` supports `.yml`, `.yaml`, `.json` and `.jsonl`.

```
[
  {
    "text": "You look great!",
    "answer": "Compliment"
  },
  {
    "text": "You are not very clever at all.",
    "answer": "Insult"
  }
]
```

```
[components.llm.task.examples]
@misc = "spacy.FewShotReader.v1"
path = "textcat_examples.json"
```

If you want to perform few-shot learning with a binary classifier (i. e. a text either should or should not be assigned to a given class), you can provide positive and negative examples with answers of "POS" or "NEG". "POS" means that this example should be assigned the class label defined in the configuration, "NEG" means it shouldn't. E. g. for spam classification:

```
[
  {
    "text": "You won the lottery! Wire a fee of 200$ to be able to withdraw your winni
    "answer": "POS"
  },
  {
    "text": "Your order #123456789 has arrived",
    "answer": "NEG"
  }
]
```

REL

The REL task extracts relations between named entities.

spacy.REL.v1

The built-in REL task supports both zero-shot and few-shot prompting. It relies on an upstream NER component for entities extraction.

ARGUMENT	DESCRIPTION
`template`	Custom prompt template to send to LLM model. Defaults to `rel.v3.jinja` </> . **TYPE:** `str`
`examples`	Optional function that generates examples for few-shot learning. Defaults to `None` . **TYPE:** `Optional[Callable[[], Iterable[Any]]]`
`parse_responses` (NEW)	Callable for parsing LLM responses for this task. Defaults to the internal parsing method for this task. **TYPE:** `Optional[TaskResponseParser[RELTask]]`
`prompt_example_type` (NEW)	Type to use for fewshot examples. Defaults to `RELExample` . **TYPE:** `Optional[Type[FewshotExample]]`
`scorer` (NEW)	Scorer function that evaluates the task performance on provided examples. Defaults to the metric used by spaCy. **TYPE:** `Optional[Scorer]`
`labels`	List of labels or str of comma-separated list of labels. **TYPE:** `Union[List[str], str]`
`label_definitions`	Dictionary providing a description for each relation label. Defaults to `None` . **TYPE:** `Optional[Dict[str, str]]`
`normalizer`	Function that normalizes the labels as returned by the LLM. If `None` , falls back to `spacy.LowercaseNormalizer.v1` . Defaults to `None` . **TYPE:** `Optional[Callable[[str], str]]`
`verbose`	If set to `True` , warnings will be generated when the LLM returns invalid responses. Defaults to `False` . **TYPE:** `bool`

To perform few-shot learning, you can write down a few examples in a separate file, and provide these to be injected into the prompt to the LLM. The default reader `spacy.FewShotReader.v1` supports `.yml` , `.yaml` , `.json` and `.jsonl` .

```
{"text": "Laura bought a house in Boston with her husband Mark.", "ents": [{"start_cha
```

```
{"text": "Michael travelled through South America by bike.", "ents": [{"start_char": 0
```

```
[components.llm.task]
@llm_tasks = "spacy.REL.v1"
labels = ["LivesIn", "Visits"]

[components.llm.task.examples]
@misc = "spacy.FewShotReader.v1"
path = "rel_examples.jsonl"
```

Note: the REL task relies on pre-extracted entities to make its prediction. Hence, you'll need to add a component that populates `doc.ents` with recognized spans to your spaCy pipeline and put it *before* the REL component.

For a fully working example, see this usage example </> .

Lemma

The Lemma task lemmatizes the provided text and updates the `lemma_` attribute in the doc's tokens accordingly.

spacy.Lemma.v1

This task supports both zero-shot and few-shot prompting.

ARGUMENT	DESCRIPTION
`template`	Custom prompt template to send to LLM model. Defaults to lemma.v1.jinja </> . TYPE: `str`
`examples`	Optional function that generates examples for few-shot learning. Defaults to `None` . TYPE: `Optional[Callable[[], Iterable[Any]]]`
`parse_responses` (NEW)	Callable for parsing LLM responses for this task. Defaults to the internal parsing method for this task. TYPE: `Optional[TaskResponseParser[LemmaTask]]`
`prompt_example_type` (NEW)	Type to use for fewshot examples. Defaults to `LemmaExample` . TYPE: `Optional[Type[FewshotExample]]`
`scorer` (NEW)	Scorer function that evaluates the task performance on provided examples. Defaults to the metric used by spaCy. TYPE: `Optional[Scorer]`

The task prompts the LLM to lemmatize the passed text and return the lemmatized version as a list of tokens and their corresponding lemma. E. g. the text `I'm buying ice cream for my friends` should invoke the response

```
I: I
'm: be
buying: buy
ice: ice
cream: cream
for: for
my: my
friends: friend
.: .
```

If for any given text/doc instance the number of lemmas returned by the LLM doesn’t match the number of tokens from the pipeline’s tokenizer, no lemmas are stored in the corresponding doc’s tokens. Otherwise the tokens `.lemma_` property is updated with the lemma suggested by the LLM.

To perform few-shot learning, you can write down a few examples in a separate file, and provide these to be injected into the prompt to the LLM. The default reader `spacy.FewShotReader.v1` supports `.yml`, `.yaml`, `.json` and `.jsonl`.

```yaml
- text: I'm buying ice cream.
  lemmas:
    - 'I': 'I'
    - "'m": 'be'
    - 'buying': 'buy'
    - 'ice': 'ice'
    - 'cream': 'cream'
    - '.': '.'

- text: I've watered the plants.
  lemmas:
    - 'I': 'I'
    - "'ve": 'have'
    - 'watered': 'water'
    - 'the': 'the'
    - 'plants': 'plant'
    - '.': '.'
```

```ini
[components.llm.task]
@llm_tasks = "spacy.Lemma.v1"
[components.llm.task.examples]
@misc = "spacy.FewShotReader.v1"
path = "lemma_examples.yml"
```

Sentiment

Performs sentiment analysis on provided texts. Scores between 0 and 1 are stored in `Doc._.sentiment` - the higher, the more positive. Note in cases of parsing issues (e. g. in case of unexpected LLM responses) the value might be `None`.

spacy.Sentiment.v1

This task supports both zero-shot and few-shot prompting.

ARGUMENT	DESCRIPTION
`template`	Custom prompt template to send to LLM model. Defaults to sentiment.v1.jinja. **TYPE:** `str`
`examples`	Optional function that generates examples for few-shot learning. Defaults to `None`. **TYPE:** `Optional[Callable[[], Iterable[Any]]]`
`parse_responses` (NEW)	Callable for parsing LLM responses for this task. Defaults to the internal parsing method for this task. **TYPE:** `Optional[TaskResponseParser[SentimentTask]]`
`prompt_example_type` (NEW)	Type to use for fewshot examples. Defaults to `SentimentExample`. **TYPE:** `Optional[Type[FewshotExample]]`
`scorer` (NEW)	Scorer function that evaluates the task performance on provided examples. Defaults to the metric used by spaCy. **TYPE:** `Optional[Scorer]`
`field`	Name of extension attribute to store summary in (i. e. the summary will be available in `doc._.{field}`). Defaults to `sentiment`. **TYPE:** `str`

To perform few-shot learning, you can write down a few examples in a separate file, and provide these to be injected into the prompt to the LLM. The default reader `spacy.FewShotReader.v1` supports `.yml`, `.yaml`, `.json` and `.jsonl`.

```
- text: 'This is horrifying.'
  score: 0
- text: 'This is underwhelming.'
  score: 0.25
- text: 'This is ok.'
  score: 0.5
- text: "I'm looking forward to this!"
  score: 1.0
```

```
[components.llm.task]
@llm_tasks = "spacy.Sentiment.v1"
[components.llm.task.examples]
@misc = "spacy.FewShotReader.v1"
path = "sentiment_examples.yml"
```

NoOp

This task is only useful for testing - it tells the LLM to do nothing, and does not set any fields on the `docs`.

spacy.NoOp.v1

This task needs no further configuration.

Models

A *model* defines which LLM model to query, and how to query it. It can be a simple function taking a collection of prompts (consistent with the output type of `task.generate_prompts()`) and returning a collection of responses (consistent with the expected input of `parse_responses`). Generally speaking, it's a function of type `Callable[[Iterable[Iterable[Any]]], Iterable[Iterable[Any]]]`, but specific implementations can have other signatures, like `Callable[[Iterable[Iterable[str]]], Iterable[Iterable[str]]]`.

Note: the model signature expects a nested iterable so it's able to deal with sharded docs. Unsharded docs (i. e. those produced by (nonsharding tasks)[/api/large-language-models#task-nonsharding]) are reshaped to fit the expected data structure.

Models via REST API

These models all take the same parameters, but note that the `config` should contain provider-specific keys and values, as it will be passed onwards to the provider's API.

ARGUMENT	DESCRIPTION
`name`	Model name, i. e. any supported variant for this particular model. Default depends on the specific model (cf. below) **TYPE:** `str`
`config`	Further configuration passed on to the model. Default depends on the specific model (cf. below). **TYPE:** `Dict[Any, Any]`
`strict`	If `True`, raises an error if the LLM API returns a malformed response. Otherwise, return the error responses as is. Defaults to `True`. **TYPE:** `bool`
`max_tries`	Max. number of tries for API request. Defaults to `5`. **TYPE:** `int`
`max_request_time`	Max. time (in seconds) to wait for request to terminate before raising an exception. Defaults to `30.0`. **TYPE:** `float`
`interval`	Time interval (in seconds) for API retries in seconds. Defaults to `1.0`. **TYPE:** `float`
`endpoint`	Endpoint URL. Defaults to the provider's standard URL, if available (which is not the case for providers with exclusively custom deployments, such as Azure) **TYPE:** `Optional[str]`

Currently, these models are provided as part of the core library:

MODEL	PROVIDER	SUPPORTED NAMES	DEFAULT NAME
`spacy.GPT-4.v1`	OpenAI	`["gpt-4", "gpt-4-0314", "gpt-4-32k", "gpt-4-32k-0314"]`	`"gpt-4"`
`spacy.GPT-4.v2`	OpenAI	`["gpt-4", "gpt-4-0314", "gpt-4-32k", "gpt-4-32k-0314"]`	`"gpt-4"`
`spacy.GPT-4.v3`	OpenAI	All names of GPT-4 models offered by OpenAI	`"gpt-4"`
`spacy.GPT-3-5.v1`	OpenAI	`["gpt-3.5-turbo", "gpt-3.5-turbo-16k", "gpt-3.5-turbo-0613", "gpt-3.5-turbo-0613-16k", "gpt-3.5-turbo-instruct"]`	`"gpt-3.5-turbo`
`spacy.GPT-3-5.v2`	OpenAI	`["gpt-3.5-turbo", "gpt-3.5-turbo-16k", "gpt-3.5-turbo-0613", "gpt-3.5-turbo-0613-16k", "gpt-3.5-turbo-instruct"]`	`"gpt-3.5-turbo`
`spacy.GPT-3-5.v3`	OpenAI	All names of GPT-3.5 models offered by OpenAI	`"gpt-3.5-turbo`
`spacy.Davinci.v1`	OpenAI	`["davinci"]`	`"davinci"`
`spacy.Davinci.v2`	OpenAI	`["davinci"]`	`"davinci"`
`spacy.Text-Davinci.v1`	OpenAI	`["text-davinci-003", "text-davinci-002"]`	`"text-davinci-`
`spacy.Text-Davinci.v2`	OpenAI	`["text-davinci-003", "text-davinci-002"]`	`"text-davinci-`
`spacy.Code-Davinci.v1`	OpenAI	`["code-davinci-002"]`	`"code-davinci-`
`spacy.Code-Davinci.v2`	OpenAI	`["code-davinci-002"]`	`"code-davinci-`
`spacy.Curie.v1`	OpenAI	`["curie"]`	`"curie"`
`spacy.Curie.v2`	OpenAI	`["curie"]`	`"curie"`
`spacy.Text-Curie.v1`	OpenAI	`["text-curie-001"]`	`"text-curie-001`

`spacy.Text-Curie.v2`	OpenAI	`["text-curie-001"]`	`"text-curie-001`
`spacy.Babbage.v1`	OpenAI	`["babbage"]`	`"babbage"`
`spacy.Babbage.v2`	OpenAI	`["babbage"]`	`"babbage"`
`spacy.Text-Babbage.v1`	OpenAI	`["text-babbage-001"]`	`"text-babbage-0`
`spacy.Text-Babbage.v2`	OpenAI	`["text-babbage-001"]`	`"text-babbage-0`
`spacy.Ada.v1`	OpenAI	`["ada"]`	`"ada"`
`spacy.Ada.v2`	OpenAI	`["ada"]`	`"ada"`
`spacy.Text-Ada.v1`	OpenAI	`["text-ada-001"]`	`"text-ada-001"`
`spacy.Text-Ada.v2`	OpenAI	`["text-ada-001"]`	`"text-ada-001"`
`spacy.Azure.v1`	Microsoft, OpenAI	Arbitrary values	No default
`spacy.Command.v1`	Cohere	`["command", "command-light", "command-light-nightly", "command-nightly"]`	`"command"`
`spacy.Claude-2-1.v1`	Anthropic	`["claude-2-1"]`	`"claude-2-1"`
`spacy.Claude-2.v1`	Anthropic	`["claude-2", "claude-2-100k"]`	`"claude-2"`
`spacy.Claude-1.v1`	Anthropic	`["claude-1", "claude-1-100k"]`	`"claude-1"`
`spacy.Claude-1-0.v1`	Anthropic	`["claude-1.0"]`	`"claude-1.0"`
`spacy.Claude-1-2.v1`	Anthropic	`["claude-1.2"]`	`"claude-1.2"`
`spacy.Claude-1-3.v1`	Anthropic	`["claude-1.3", "claude-1.3-100k"]`	`"claude-1.3"`
`spacy.Claude-instant-1.v1`	Anthropic	`["claude-instant-1", "claude-instant-1-100k"]`	`"claude-instant`
`spacy.Claude-instant-1-1.v1`	Anthropic	`["claude-instant-1.1", "claude-instant-1.1-100k"]`	`"claude-instant`

`spacy.PaLM.v1`	Google	`["chat-bison-001", "text-bison-001"]`	`"text-bison-001`

To use these models, make sure that you've set the relevant API keys as environment variables.

⚠️ **A note on `spacy.Azure.v1`**. Working with Azure OpenAI is slightly different than working with models from other providers:

- In Azure LLMs have to be made available by creating a *deployment* of a given model (e. g. GPT-3.5). This deployment can have an arbitrary name. The `name` argument, which everywhere else denotes the model name (e. g. `claude-1.0`, `gpt-3.5`), here refers to the *deployment name*.
- Deployed Azure OpenAI models are reachable via a resource-specific base URL, usually of the form `https://{resource}.openai.azure.com`. Hence the URL has to be specified via the `base_url` argument.
- Azure further expects the *API version* to be specified. The default value for this, via the `api_version` argument, is currently `2023-05-15` but may be updated in the future.
- Finally, since we can't infer information about the model from the deployment name, `spacy-llm` requires the `model_type` to be set to either `"completions"` or `"chat"`, depending on whether the deployed model is a completion or chat model.

API Keys

Note that when using hosted services, you have to ensure that the proper API keys are set as environment variables as described by the corresponding provider's documentation.

E. g. when using OpenAI, you have to get an API key from openai.com, and ensure that the keys are set as environmental variables:

```
export OPENAI_API_KEY="sk-..."
export OPENAI_API_ORG="org-..."
```

For Cohere:

```
export CO_API_KEY="..."
```

For Anthropic:

```
export ANTHROPIC_API_KEY="..."
```

For PaLM:

```
export PALM_API_KEY="..."
```

Models via HuggingFace

These models all take the same parameters:

ARGUMENT	DESCRIPTION
`name`	Model name, i. e. any supported variant for this particular model. **TYPE:** `str`
`config_init`	Further configuration passed on to the construction of the model with `transformers.pipeline()`. Defaults to `{}`. **TYPE:** `Dict[str, Any]`
`config_run`	Further configuration used during model inference. Defaults to `{}`. **TYPE:** `Dict[str, Any]`

Currently, these models are provided as part of the core library:

MODEL	PROVIDER	SUPPORTED NAMES	HF DIRECTORY
`spacy.Dolly.v1`	Databricks	`["dolly-v2-3b", "dolly-v2-7b", "dolly-v2-12b"]`	https://huggingface.co/databricks
`spacy.Falcon.v1`	TII	`["falcon-rw-1b", "falcon-7b", "falcon-7b-instruct", "falcon-40b-instruct"]`	https://huggingface.co/tiiuae
`spacy.Llama2.v1`	Meta AI	`["Llama-2-7b-hf", "Llama-2-13b-hf", "Llama-2-70b-hf"]`	https://huggingface.co/meta-llama
`spacy.Mistral.v1`	Mistral AI	`["Mistral-7B-v0.1", "Mistral-7B-Instruct-v0.1"]`	https://huggingface.co/mistralai
`spacy.StableLM.v1`	Stability AI	`["stablelm-base-alpha-3b", "stablelm-base-alpha-7b", "stablelm-tuned-alpha-3b", "stablelm-tuned-alpha-7b"]`	https://huggingface.co/stabilityai
`spacy.OpenLLaMA.v1`	OpenLM Research	`["open_llama_3b", "open_llama_7b", "open_llama_7b_v2", "open_llama_13b"]`	https://huggingface.co/openlm-research

Note that Hugging Face will download the model the first time you use it - you can define the cached directory by setting the environmental variable `HF_HOME`.

Installation with HuggingFace

To use models from HuggingFace, ideally you have a GPU enabled and have installed `transformers`, `torch` and CUDA in your virtual environment. This allows you to have the setting `device=cuda:0` in your config, which ensures that the model is loaded entirely on the GPU (and fails otherwise).

You can do so with

```
python -m pip install "spacy-llm[transformers]" "transformers[sentencepiece]"
```

If you don’t have access to a GPU, you can install `accelerate` and set `device_map=auto` instead, but be aware that this may result in some layers getting distributed to the CPU or even the hard drive, which may ultimately result in extremely slow queries.

```
python -m pip install "accelerate>=0.16.0,<1.0"
```

LangChain models

To use LangChain </> for the API retrieval part, make sure you have installed it first:

```
python -m pip install "langchain==0.0.191"
# Or install with spacy-llm directly
python -m pip install "spacy-llm[extras]"
```

Note that LangChain currently only supports Python 3.9 and beyond.

LangChain models in `spacy-llm` work slightly differently. `langchain`’s models are parsed automatically, each LLM class in `langchain` has one entry in `spacy-llm`’s registry. As `langchain`’s design has one class per API and not per model, this results in registry entries like `langchain.OpenAI.v1` - i. e. there is one registry entry per API and not per model (family), as for the REST- and HuggingFace-based entries.

The name of the model to be used has to be passed in via the `name` attribute.

ARGUMENT	DESCRIPTION
`name`	The name of a mdodel supported by LangChain for this API. **TYPE:** `str`
`config`	Configuration passed on to the LangChain model. Defaults to `{}`. **TYPE:** `Dict[Any, Any]`
`query`	Function that executes the prompts. If `None`, defaults to `spacy.CallLangChain.v1`. **TYPE:** `Optional[Callable[["langchain.llms.BaseLLM", Iterable[Any]], Iterable[Any]]]`

The default `query` (`spacy.CallLangChain.v1`) executes the prompts by running `model(text)` for each given textual prompt.

Cache

Interacting with LLMs, either through an external API or a local instance, is costly. Since developing an NLP pipeline generally means a lot of exploration and prototyping, `spacy-llm` implements a built-in cache to avoid reprocessing the same documents at each run that keeps batches of documents stored on disk.

ARGUMENT	DESCRIPTION
`path`	Cache directory. If `None`, no caching is performed, and this component will act as a NoOp. Defaults to `None`. **TYPE:** `Optional[Union[str,Path]]`
`batch_size`	Number of docs in one batch (file). Once a batch is full, it will be peristed to disk. Defaults to 64. **TYPE:** `int`
`max_batches_in_mem`	Max. number of batches to hold in memory. Allows you to limit the effect on your memory if you're handling a lot of docs. Defaults to 4. **TYPE:** `int`

When retrieving a document, the `BatchCache` will first figure out what batch the document belongs to. If the batch isn't in memory it will try to load the batch from disk and then move it into memory.

Note that since the cache is generated by a registered function, you can also provide your own registered function returning your own cache implementation. If you wish to do so, ensure that your cache object adheres to the `Protocol` defined in `spacy_llm.ty.Cache`.

Various functions

spacy.FewShotReader.v1

This function is registered in spaCy's `misc` registry, and reads in examples from a `.yml`, `.yaml`, `.json` or `.jsonl` file. It uses `srsly` </> to read in these files and parses them depending on the file extension.

ARGUMENT	DESCRIPTION
`path`	Path to an examples file with suffix `.yml`, `.yaml`, `.json` or `.jsonl`. **TYPE:** `Union[str, Path]`

spacy.FileReader.v1

This function is registered in spaCy's `misc` registry, and reads a file provided to the `path` to return a `str` representation of its contents. This function is typically used to read Jinja files containing the prompt template.

ARGUMENT	DESCRIPTION
`path`	Path to the file to be read. **TYPE:** `Union[str, Path]`

Normalizer functions

These functions provide simple normalizations for string comparisons, e.g. between a list of specified labels and a label given in the raw text of the LLM response. They are registered in spaCy's `misc` registry and have the signature `Callable[[str], str]`.

- `spacy.StripNormalizer.v1` : only apply `text.strip()`
- `spacy.LowercaseNormalizer.v1` : applies `text.strip().lower()` to compare strings in a case-insensitive way.

</> SUGGEST EDITS