```
from dataclasses import dataclass
import time
from typing import Any, Dict, List, Literal, Optional, Union

import shortuuid
from pydantic import BaseModel, Field


class ErrorResponse(BaseModel):
    """Error responses."""

    object: str = "error"
    message: str
    code: int


class ModelPermission(BaseModel):
    """Model permissions."""

    id: str = Field(default_factory=lambda: f"modelperm-{shortuuid.random()}")
    object: str = "model_permission"
    created: int = Field(default_factory=lambda: int(time.time()))
    allow_create_engine: bool = False
    allow_sampling: bool = True
    allow_logprobs: bool = True
    allow_search_indices: bool = True
```

```
    allow_view: bool = True
    allow_fine_tuning: bool = False
    organization: str = "*"
    group: Optional[str] = None
    is_blocking: bool = False


class ModelCard(BaseModel):
    """Model cards."""

    id: str
    object: str = "model"
    created: int = Field(default_factory=lambda: int(time.time()))
    owned_by: str = "swarms"
    root: Optional[str] = None
    parent: Optional[str] = None
    permission: List[ModelPermission] = []


class ModelList(BaseModel):
    """Model list consists of model cards."""

    object: str = "list"
    data: List[ModelCard] = []
```

```
class UsageInfo(BaseModel):
    """Usage information."""

    prompt_tokens: int = 0
    total_tokens: int = 0
    completion_tokens: Optional[int] = 0


class ChatCompletionRequestQos(BaseModel):
    """Chat completion request."""

    model: str
    messages: Union[str, List[Dict[str, str]]]
    temperature: Optional[float] = 0.7
    top_p: Optional[float] = 1.0
    n: Optional[int] = 1
    max_tokens: Optional[int] = 512
    stop: Optional[bool] = False
    stream: Optional[bool] = False
    presence_penalty: Optional[float] = 0.0
    frequency_penalty: Optional[float] = 0.0
    user: Optional[str] = None
    user_id: Optional[str] = None
    # additional argument of swarms
    repetition_penalty: Optional[float] = 1.0
    session_id: Optional[int] = -1
```

```
    ignore_eos: Optional[bool] = False


class ChatCompletionRequest(BaseModel):
    """Chat completion request."""

    model: str
    # yapf: disable
    messages: Union[str, List[Dict[str, str]]] = Field(examples=[[{'role': 'user', 'content': 'hi'}]])  # noqa
    temperature: Optional[float] = 0.7
    top_p: Optional[float] = 1.0
    n: Optional[int] = 1
    max_tokens: Optional[int] = 512
    stop: Optional[Union[str, List[str]]] = Field(default=None, examples=[None])  # noqa
    # yapf: enable
    stream: Optional[bool] = False
    presence_penalty: Optional[float] = 0.0
    frequency_penalty: Optional[float] = 0.0
    user: Optional[str] = None
    # additional argument of swarms
    repetition_penalty: Optional[float] = 1.0
    session_id: Optional[int] = -1
    ignore_eos: Optional[bool] = False


class ChatMessage(BaseModel):
```

```
    """Chat messages."""

    role: str
    content: str


class ChatCompletionResponseChoice(BaseModel):
    """Chat completion response choices."""

    index: int
    message: ChatMessage
    finish_reason: Optional[Literal["stop", "length"]] = None


class ChatCompletionResponse(BaseModel):
    """Chat completion response."""

    id: str = Field(default_factory=lambda: f"chatcmpl-{shortuuid.random()}")
    object: str = "chat.completion"
    created: int = Field(default_factory=lambda: int(time.time()))
    model: str
    choices: List[ChatCompletionResponseChoice]
    usage: UsageInfo


class DeltaMessage(BaseModel):
```

```
    """Delta messages."""

    role: Optional[str] = None
    content: Optional[str] = None


class ChatCompletionResponseStreamChoice(BaseModel):
    """Chat completion response stream choice."""

    index: int
    delta: DeltaMessage
    finish_reason: Optional[Literal["stop", "length"]] = None


class ChatCompletionStreamResponse(BaseModel):
    """Chat completion stream response."""

    id: str = Field(default_factory=lambda: f"chatcmpl-{shortuuid.random()}")
    object: str = "chat.completion.chunk"
    created: int = Field(default_factory=lambda: int(time.time()))
    model: str
    choices: List[ChatCompletionResponseStreamChoice]


class CompletionRequest(BaseModel):
    """Completion request."""
```

```
    model: str
    prompt: Union[str, List[Any]]
    suffix: Optional[str] = None
    temperature: Optional[float] = 0.7
    n: Optional[int] = 1
    max_tokens: Optional[int] = 16
    stop: Optional[Union[str, List[str]]] = Field(default=None, examples=[None])
    stream: Optional[bool] = False
    top_p: Optional[float] = 1.0
    logprobs: Optional[int] = None
    echo: Optional[bool] = False
    presence_penalty: Optional[float] = 0.0
    frequency_penalty: Optional[float] = 0.0
    user: Optional[str] = None
    # additional argument of swarms
    repetition_penalty: Optional[float] = 1.0
    session_id: Optional[int] = -1
    ignore_eos: Optional[bool] = False
    top_k: Optional[int] = 40  # for opencompass


class CompletionRequestQos(BaseModel):
    """Completion request."""

    model: str
```

```
    prompt: Union[str, List[Any]]
    suffix: Optional[str] = None
    temperature: Optional[float] = 0.7
    n: Optional[int] = 1
    max_tokens: Optional[int] = 16
    stop: Optional[Union[str, List[str]]] = None
    stream: Optional[bool] = False
    top_p: Optional[float] = 1.0
    logprobs: Optional[int] = None
    echo: Optional[bool] = False
    presence_penalty: Optional[float] = 0.0
    frequency_penalty: Optional[float] = 0.0
    user: Optional[str] = None
    # additional argument of swarms
    top_k: int = 40
    repetition_penalty: Optional[float] = 1.0
    session_id: Optional[int] = -1
    ignore_eos: Optional[bool] = False
    user_id: Optional[str] = None


class CompletionResponseChoice(BaseModel):
    """Completion response choices."""

    index: int
    text: str
```

```
    logprobs: Optional[int] = None
    finish_reason: Optional[Literal["stop", "length"]] = None


class CompletionResponse(BaseModel):
    """Completion response."""

    id: str = Field(default_factory=lambda: f"cmpl-{shortuuid.random()}")
    object: str = "text_completion"
    created: int = Field(default_factory=lambda: int(time.time()))
    model: str
    choices: List[CompletionResponseChoice]
    usage: UsageInfo


class CompletionResponseStreamChoice(BaseModel):
    """Completion response stream choice."""

    index: int
    text: str
    logprobs: Optional[float] = None
    finish_reason: Optional[Literal["stop", "length"]] = None


class CompletionStreamResponse(BaseModel):
    """Completion stream response."""
```

```
    id: str = Field(default_factory=lambda: f"cmpl-{shortuuid.random()}")
    object: str = "text_completion"
    created: int = Field(default_factory=lambda: int(time.time()))
    model: str
    choices: List[CompletionResponseStreamChoice]


class EmbeddingsRequest(BaseModel):
    """Embedding request."""

    model: str = None
    input: Union[str, List[str]]
    user: Optional[str] = None


class EmbeddingsResponse(BaseModel):
    """Embedding response."""

    object: str = "list"
    data: List[Dict[str, Any]]
    model: str
    usage: UsageInfo


class EncodeRequest(BaseModel):
```

```
    """Encode request."""

    input: Union[str, List[str]]
    do_preprocess: Optional[bool] = False
    add_bos: Optional[bool] = True


class EncodeResponse(BaseModel):
    """Encode response."""

    input_ids: Union[List[int], List[List[int]]]
    length: Union[int, List[int]]


class GenerateRequest(BaseModel):
    """Generate request."""

    prompt: Union[str, List[Dict[str, str]]]
    session_id: int = -1
    interactive_mode: bool = False
    stream: bool = False
    stop: Optional[Union[str, List[str]]] = Field(default=None, examples=[None])
    request_output_len: int = 512
    top_p: float = 0.8
    top_k: int = 40
    temperature: float = 0.8
```

```
    repetition_penalty: float = 1.0
    ignore_eos: bool = False
    cancel: Optional[bool] = False  # cancel a responding request


class GenerateRequestQos(BaseModel):
    """Generate request."""

    prompt: Union[str, List[Dict[str, str]]]
    session_id: int = -1
    interactive_mode: bool = False
    stream: bool = False
    stop: bool = False
    request_output_len: int = 512
    top_p: float = 0.8
    top_k: int = 40
    temperature: float = 0.8
    repetition_penalty: float = 1.0
    ignore_eos: bool = False
    user_id: Optional[str] = None


class GenerateResponse(BaseModel):
    """Generate response."""

    text: str
```

```
    tokens: int
    finish_reason: Optional[Literal["stop", "length"]] = None


@dataclass
class GenerationConfig:
    """generation parameters used by inference engines.

    Args:
        n (int): Define how many chat completion choices to generate for each
            input message
        max_new_tokens (int): The maximum number of tokens that can be
            generated in the chat completion
        top_p (float): An alternative to sampling with temperature, called
            nucleus sampling, where the model considers the results of the
            tokens with top_p probability mass
        top_k (int): An alternative to sampling with temperature, where
            the model considers the top_k tokens with the highest probability
        temperature (float): Sampling temperature
        repetition_penalty (float): Penalty to prevent the model from
            generating repeated words or phrases. A value larger than
            1 discourages repetition
        ignore_eos (bool): Indicator to ignore the eos_token_id or not
        random_seed (int): Seed used when sampling a token
        stop_words (List[str]): Words that stop generating further tokens
        bad_words (List[str]): Words that the engine will never generate
```

```
    min_new_tokens (int): The minimum numbers of tokens to generate,
        ignoring the number of tokens in the prompt.
"""


n: int = 1
max_new_tokens: int = 512
top_p: float = 1.0
top_k: int = 1
temperature: float = 0.8
repetition_penalty: float = 1.0
ignore_eos: bool = False
random_seed: int = None
stop_words: List[str] = None
bad_words: List[str] = None
min_new_tokens: int = None
```