

```
import uuid
```

```
import time
```

```
from typing import List, Literal, Optional, Union
```

```
from pydantic import BaseModel, Field
```

```
class ModelCard(BaseModel):
```

```
    """
```

A Pydantic model representing a model card, which provides metadata about a machine learning model.

It includes fields like model ID, owner, and creation time.

```
    """
```

```
    id: str
```

```
    object: str = "model"
```

```
    created: int = Field(default_factory=lambda: int(time.time()))
```

```
    owned_by: str = "owner"
```

```
    root: Optional[str] = None
```

```
    parent: Optional[str] = None
```

```
    permission: Optional[list] = None
```

```
class ModelList(BaseModel):
```

```
    object: str = "list"
```

```
    data: List[ModelCard] = []
```

```
class ImageUrl(BaseModel):
```

```
    url: str
```

```
class TextContent(BaseModel):
```

```
    type: Literal["text"]
```

```
    text: str
```

```
class ImageUrlContent(BaseModel):
```

```
    type: Literal["image_url"]
```

```
    image_url: ImageUrl
```

```
ContentItem = Union[TextContent, ImageUrlContent]
```

```
class ChatMessageInput(BaseModel):
```

```
    role: str = Field(
```

```
        ...,
```

```
        description="The role of the message sender. Could be 'user', 'assistant', or 'system'.",
```

```
    )
```

```
    content: Union[str, List[ContentItem]]
```

```
class ChatMessageResponse(BaseModel):
```

```
    role: str = Field(
```

```
        ...,
```

```
        description="The role of the message sender. Could be 'user', 'assistant', or 'system'.",
```

```
    )
```

```
    content: str = None
```

```
class DeltaMessage(BaseModel):
```

```
    role: Optional[Literal["user", "assistant", "system"]] = None
```

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

```
class ChatCompletionRequest(BaseModel):
```

```
    model: str = "gpt-4o"
```

```
    messages: List[ChatMessageInput]
```

```
    temperature: Optional[float] = 0.8
```

```
    top_p: Optional[float] = 0.8
```

```
    max_tokens: Optional[int] = 4000
```

```
    stream: Optional[bool] = False
```

```
    repetition_penalty: Optional[float] = 1.0
```

```
    echo: Optional[bool] = False
```

```
class ChatCompletionResponseChoice(BaseModel):
```

```
index: int = Field(..., description="The index of the choice.")
input: str = Field(..., description="The input message.")
message: ChatMessageResponse = Field(
    ..., description="The output message."
)
```

```
class ChatCompletionResponseStreamChoice(BaseModel):
```

```
    index: int
```

```
    delta: DeltaMessage
```

```
class UsageInfo(BaseModel):
```

```
    prompt_tokens: int = 0
```

```
    total_tokens: int = 0
```

```
    completion_tokens: Optional[int] = 0
```

```
class ChatCompletionResponse(BaseModel):
```

```
    model: str
```

```
    object: Literal["chat.completion", "chat.completion.chunk"]
```

```
    choices: List[
```

```
        Union[
```

```
            ChatCompletionResponseChoice,
```

```
            ChatCompletionResponseStreamChoice,
```

```
        ]
```

```
]
```

```
created: Optional[int] = Field(  
    default_factory=lambda: int(time.time()))  
)
```

```
class AgentChatCompletionResponse(BaseModel):
```

```
    id: Optional[str] = Field(  
        f"agent-{uuid.uuid4().hex}",  
        description="The ID of the agent that generated the completion response.",  
    )
```

```
    agent_name: Optional[str] = Field(  
        ...,  
        description="The name of the agent that generated the completion response.",  
    )
```

```
    object: Optional[  
        Literal["chat.completion", "chat.completion.chunk"]  
    ] = None
```

```
    choices: Optional[ChatCompletionResponseChoice] = None
```

```
    created: Optional[int] = Field(  
        default_factory=lambda: int(time.time()))  
)
```

```
    # full_usage: Optional[UsageInfo]
```