

Act as a teacher who is creating dense class materials to teach computer science professionals how to use AI; these materials should be short, yet effective. Your task is to provide a comprehensive guide on local Ollama prompts in python for a new learner. You can reference Ollama's online libraries and models for further background. For each part of the python AI code, please explain what the parameters for the function/method are intended for. You should use a definitive tone with for your delivery.

You can use this example:

```
<example>
response = chat(
    model=MODEL_NAME,
    messages=[
        {"role": "system", "content": prompt},
        {"role": "user", "content": expected},
    ],
    options={"temperature": TEMP},
)
</example>
```

<explaination>

You can reference this for what kind of parameters should be explained. In this example, you should explain the "roles", and the "options". Furthermore, you should specify the difference between system and user roles.

</explaination>

Local AI Integration with Ollama: A Technical Guide for Developers

1. Core Architecture Overview

Ollama provides a local inference engine for running open-source LLMs. The Python `ollama` package interfaces with the Ollama server (running locally via Docker or

direct installation) to execute models without cloud dependencies.

```
python

import ollama

# Basic completion
response = ollama.generate(
    model='llama3.2:latest', # Model identifier
    prompt='Explain quantum entanglement', # Input text
    options={
        'temperature': 0.7, # Creativity: 0.0-2.0
        'num_predict': 512, # Max output tokens
        'top_p': 0.9, # Nucleus sampling
        'repeat_penalty': 1.1 # Reduce repetition
    }
)
```

Parameter Breakdown:

- `model` : String identifying the model (pull from Ollama library:
`ollama pull llama3.2`)
- `prompt` : Raw text input for completion tasks
- `options` : Dictionary controlling inference behavior

2. Chat Completion with Role Management

```
python

response = ollama.chat(
    model='mistral:7b-instruct',
    messages=[
        {
            'role': 'system',
            'content': 'You are a senior Python architect. Provide concise, production-ready code.'
        },
        {
            'role': 'user',
            'content': 'Implement async file parsing with error handling'
        }
    ]
)
```

```

},
{
    'role': 'assistant',
    'content': 'I\'ll create a robust parser using asyncio and...'
},
{
    'role': 'user',
    'content': 'Now add compression support'
}
],
options={
    'temperature': 0.3,
    'top_k': 40,           # Limit token selection pool
    'seed': 42,            # Deterministic outputs
    'stream': False        # Return full response vs chunks
}
)

```

Role Definitions:

| Role | Purpose | Context Window Usage |
|-----------|---|----------------------------------|
| system | Sets behavior, constraints, personality | Persistent across entire session |
| user | Human/program inputs, queries, requests | Added sequentially |
| assistant | Previous AI responses for continuity | Maintains conversation state |

Critical Notes:

- System prompts are **not supported by all models** (check model cards)
- Assistant messages enable multi-turn conversations with memory
- Each message consumes tokens from the model's context window (e.g., 4096 for Llama 3.2)

3. Advanced Configuration Parameters

```
python
```

```
# Production configuration example
response = ollama.chat(
    model='codellama:13b',
    messages=[...],
    format='json', # Force JSON output schema
    options={
        # Creativity & Diversity
        'temperature': 0.8,          # 0.0 (deterministic) → 2.0 (chaotic)
        'top_p': 0.95,              # Cumulative probability cutoff
        'top_k': 50,                # Consider top N tokens only

        # Generation Control
        'num_predict': 1024,         # Hard stop after N tokens
        'stop': ['\n```, '<|end|>'], # Sequence triggers stop
        'repeat_penalty': 1.2,       # Penalize repeated phrases

        # Performance
        'num_ctx': 4096,            # Context window size
        'num_batch': 512,            # Parallel processing batch
        'num_thread': 8,             # CPU threads for inference

        # Reproducibility
        'seed': 12345,              # Fixed seed for deterministic outputs
        'tfs_z': 1.0,                # Tail-free sampling parameter
        'mirostat': 2,               # 0=disabled, 1=v1, 2=v2 adaptive perplexity
    }
)
```

4. Streaming for Real-time Applications

```
python
```

```
# Progressive output for UX
stream = ollama.chat(
    model='llama3.2:latest',
    messages=[{'role': 'user', 'content': 'Write a Kafka consumer'},
    stream=True, # Enable chunked response
    options={'temperature': 0.5}
)
```

```

for chunk in stream:
    if chunk[ 'done']: break
    print(chunk[ 'message'][ 'content'], end=' ', flush=True)
    # Chunk structure: { 'message': { 'content': '...'}, 'done': False}

```

Streaming Benefits:

- Reduced perceived latency
 - Token-by-token processing visibility
 - Memory efficiency for long outputs
-

5. Model Management & Embeddings

```

python

# List available models
models = ollama.list() # Returns local models with details

# Pull new models
ollama.pull('llama3.2:3b') # Downloads from OLLAMA Library

# Generate embeddings for RAG
embedding = ollama.embeddings(
    model='nomic-embed-text',
    prompt='Document text for vectorization'
) # Returns float32 vector

```

6. Production Best Practices

Configuration Strategy:

```

python

PROD_CONFIG = {
    'temperature': 0.3,      # Lower for factual tasks
    'top_p': 0.9,           # Balance creativity/coherence
    'num_predict': 512,     # Prevent runaway generation
    'repeat_penalty': 1.1   # Essential for long-form content
}

```

```
}

CREATIVE_CONFIG = {
    'temperature': 1.2,      # Higher for brainstorming
    'top_k': 100,           # Wider token selection
    'mirostat': 2           # Adaptive creativity control
}
```

Error Handling:

```
python

try:
    response = ollama.chat(
        model=MODEL,
        messages=messages,
        options={'request_timeout': 30.0} # Server timeout
    )
except ollama.ResponseError as e:
    if 'no such model' in str(e):
        ollama.pull(MODEL)
        # Retry logic here
except ConnectionError:
    # Ensure Ollama server is running: `ollama serve`
```

7. Quick Reference: Parameter Matrix

| Parameter | Typical Range | Effect | Use Case |
|----------------|---------------|------------------------|----------------------------------|
| temperature | 0.0-2.0 | Randomness | 0.2 for code, 1.0 for creative |
| top_p | 0.5-1.0 | Diversity vs focus | 0.9 for balanced output |
| top_k | 1-100 | Vocabulary restriction | 40 for focused, 100 for diverse |
| repeat_penalty | 1.0-1.5 | Anti-repetition | 1.1 for long documents |
| num_predict | 1-4096 | Output length limit | 256 for summaries, 1024 for docs |
| seed | Any integer | Reproducibility | Debugging, testing |

Next Steps