

Giải thích hàm load_config() - Từ A đến Z cho người mới

Mục đích của hàm này

Hàm `load_config()` giống như một "**người phục vụ thông minh**" - nó sẽ:

1. **Đầu tiên**: Tìm file cấu hình `config.json`
2. **Nếu tìm thấy**: Đọc settings từ file đó
3. **Nếu không tìm thấy**: Sử dụng settings mặc định

Phân tích từng dòng code

Dòng 1-2: Khai báo hàm

python

```
def load_config():  
    """Tải cấu hình từ file config.json hoặc trả về mặc định."""
```

- Tạo một hàm tên `load_config`
- Không cần tham số đầu vào
- Docstring giải thích chức năng

Dòng 3: Tìm đường dẫn file config

python

```
config_path = os.path.join(os.path.dirname(__file__), "..", "config.json")
```

Giải thích chi tiết:

- `__file__`: Đường dẫn file hiện tại (utils.py)
- `os.path.dirname(__file__)`: Lấy thư mục chứa file hiện tại
- `".."`: Lùi lên 1 cấp thư mục cha
- `"config.json"`: Tên file cấu hình
- `os.path.join()`: Nối các phần đường dẫn lại

Ví dụ thực tế:

Nếu `utils.py` ở: `/home/user/project/src/utils.py`

Thì `config_path` sẽ là: `/home/user/project/config.json`

Dòng 4: Cấu hình mặc định

python

```
default_config = {"project_root": os.path.abspath(os.path.join(os.path.dirname(__file__), "..")
```

Giải thích:

- Tạo dictionary chứa cấu hình mặc định
- `os.path.abspath()`: Chuyển đường dẫn thành đường dẫn tuyệt đối
- Lấy thư mục cha của file hiện tại làm `project_root`

Ví dụ:

python

```
# Nếu utils.py ở /home/user/project/src/utils.py
default_config = {
    "project_root": "/home/user/project"
}
```

Dòng 5-9: Thử đọc file config

python

```
try:
    with open(config_path, "r", encoding="utf-8") as f:
        return {**default_config, **json.load(f)}
except FileNotFoundError:
    print(f"⚠ Không tìm thấy config.json, sử dụng giá trị mặc định: {default_config}")
    return default_config
```

Giải thích từng phần:

1. `try:` - Thử thực hiện code, nếu lỗi thì nhảy sang `except`
2. `with open(config_path, "r", encoding="utf-8") as f:`
 - Mở file config với mode đọc ("r")
 - `encoding="utf-8"`: Đọc file với mã hóa UTF-8 (hỗ trợ tiếng Việt)
 - `with`: Tự động đóng file sau khi xong
3. `return {**default_config, **json.load(f)}`
 - `json.load(f)`: Đọc nội dung JSON từ file
 - `**default_config`: Trả các key-value từ default_config
 - `**json.load(f)`: Trả các key-value từ file config
 - `{...}`: Merge 2 dictionary lại (file config sẽ ghi đè default nếu trùng key)
4. `except FileNotFoundError:`
 - Bắt lỗi khi không tìm thấy file
 - In thông báo warning
 - Trả về cấu hình mặc định

Ví dụ cụ thể

Trường hợp 1: Có file config.json

json

```
// Nội dung file config.json
{
    "model_name": "phobert-base",
    "batch_size": 32,
    "learning_rate": 0.001
}
```

Kết quả:

python

```
result = {
    "project_root": "/home/user/project",    # Từ default_config
    "model_name": "phobert-base",           # Từ config.json
    "batch_size": 32,                       # Từ config.json
    "learning_rate": 0.001                  # Từ config.json
}
```

Trường hợp 2: Không có file config.json

Output console:

```
⚠ Không tìm thấy config.json, sử dụng giá trị mặc định: {'project_root': '/home/user/project'}
```

Kết quả:

```
python
```

```
result = {  
    "project_root": "/home/user/project"    # Chỉ có default_config  
}
```

Trường hợp 3: File config ghi đè default

```
json
```

```
// config.json  
{  
    "project_root": "/custom/path",  
    "model_name": "phobert-large"  
}
```

Kết quả:

```
python
```

```
result = {  
    "project_root": "/custom/path",    # Ghi đè default_config  
    "model_name": "phobert-large"     # Từ config.json  
}
```

Tại sao cần hàm này?

1. Flexibility (Linh hoạt)

```
python
```

```
# Không cần hardcode settings trong code
```

```
# BAD:
```

```
model_path = "/home/user/project/models/best.pt"
```

```
# GOOD:
```

```
config = load_config()
```

```
model_path = os.path.join(config["project_root"], "models", "best.pt")
```

2. Environment-specific (Theo môi trường)

```
python
```

```
# Development
```

```
config_dev.json: {"project_root": "/home/dev/project", "debug": true}
```

```
# Production
```

```
config_prod.json: {"project_root": "/opt/app", "debug": false}
```

3. Error Resilience (Chống lỗi)

```
python
```

```
# Nếu quên tạo config.json → vẫn chạy được với default
```

```
# Không bị crash app
```

Cách sử dụng trong thực tế

1. Tạo file config.json (optional)

```
json
```

```
{  
    "project_root": "/my/custom/path",  
    "model_name": "vinai/phobert-base",  
    "max_length": 256,  
    "batch_size": 16,  
    "device": "cuda",  
    "log_level": "INFO"  
}
```

2. Sử dụng trong code

python

Load config

```
config = load_config()
```

Sử dụng các settings

```
model_path = os.path.join(config["project_root"], "models", "best.pt")
```

```
batch_size = config.get("batch_size", 32) # Default 32 nếu không có
```

```
device = config.get("device", "cpu") # Default cpu nếu không có
```

```
print(f"Using device: {device}")
```

```
print(f"Batch size: {batch_size}")
```

```
print(f"Model path: {model_path}")
```

Pattern Dictionary Merge (***)

Đây là Python syntax để merge dictionaries:

python

```
dict1 = {"a": 1, "b": 2}
```

```
dict2 = {"b": 3, "c": 4}
```

*# Merge với ** operator*

```
result = {**dict1, **dict2}
```

```
print(result) # {"a": 1, "b": 3, "c": 4}
```

dict2 ghi đè dict1 khi trùng key

Trong hàm load_config:

python

```
default_config = {"project_root": "/default"}
```

```
file_config = {"project_root": "/custom", "batch_size": 32}
```






```
result = {**default_config, **file_config}
```

```
# {"project_root": "/custom", "batch_size": 32}
```

file_config ghi đè default_config

Tổng kết

Hàm `load_config()` là một **utility function** thông minh giúp:

-  Load settings từ file JSON
-  Fallback về default nếu không có file
-  Merge settings linh hoạt
-  Handle errors gracefully
-  Hỗ trợ đường dẫn relative và absolute

Nguyên tắc: "Cấu hình linh hoạt, fallback an toàn, không crash app!"