Requirements

The framework must:

1. **Support Categorization**: Identify interface types (e.g., LAN, WAN, CORE, INTERNET) and roles (e.g., uplink, trunk, MPLS) compatible with tools like Nautoobot, Kentik, and scripts.
2. **LLDP Neighbor Population**: Allow population of data from Link Layer Discovery Protocol (LLDP) or Cisco Discovery Protocol (CDP) neighbors.
3. **Human and Machine Readability**: Be understandable by engineers and parseable by machines.
4. **Manual Suffix Option**: Include a delimiter (e.g., : for scripted fields, || for manual content) for additional notes (e.g., circuit references not yet in Nautoobot).
5. **Dual Usage**: Serve engineers for operational tasks and network management systems for filtering, alerts, and reports.
6. **Field-Specific Information**: Include interface type, role, connected equipment, bandwidth (for WAN), and circuit references.
7. **Simplicity**: Avoid complexity that requires constant documentation reference.
8. **Regex Compatibility**: Enable extraction of fields (e.g., bandwidth) and classification by tools like Kentik.
9. **Alert Suppression**: Allow monitoring tools to ignore certain links (e.g., maintenance or unmonitored ports).
10. **Automation Support**: Enable scripts to populate descriptions based on neighbor data.

Input Examples

Inputs are the raw data or scenarios that need to be formatted into the standardized description. These are typically derived from network configurations or operational needs:

- **Example 1**: A LAN switch port connected to an access point at location US-TX-F-124-46-SL-01 on interface G10/0.
- **Example 2**: A LAN switch connected to US-TX-F-124-46-SL-01 on Port-channel100, VLAN 100.
- **Example 3**: A router on a point-to-point WAN circuit connected to US-TX-F-124-46-RA-01 on G10/0.
- **Example 4**: A core router on a 500 Mbps Megaport point-to-point NNI circuit to EU-GB-LON-C-LONPWG-PE1 on xe-0/0/0.300, with circuit reference ABC420938 and virtual circuit SLKJDLDJ.

- **Example 5**: A WAN CE-facing interface with a 50 Mbps circuit to Chevron CE ALMRA01 on Gi-0/0/0, carrier Beeline, circuit reference A00_VPLS_5200b02.

Expected Output Examples

The outputs are the formatted interface descriptions based on the standardized format (fields delimited by :), reflecting the requirements:

- **Output 1**: LAN : AP : US-TX-F-124-46-SL-01 G10/0
  - *Explanation*: Type is LAN, role is AP (access point), connected device and interface are specified.
- **Output 2**: LAN : SW : US-TX-F-124-46-SL-01 Po100.100
  - *Explanation*: Type is LAN, role is SW (switch), connected device and Port-channel100 VLAN 100 are included.
- **Output 3**: WAN : P2P : US-TX-F-124-46-RA-01 G10/0
  - *Explanation*: Type is WAN, role is P2P (point-to-point), connected device and interface are specified.
- **Output 4**: CORE : NNI : EU-GB-LON-C-LONPWG-PE1 xe-0/0/0.300 : 500m : Cogent : ABC420938 : EVC SLKJDLDJ
  - *Explanation*: Type is CORE, role is NNI, connected device/interface, bandwidth (500m), carrier (Cogent), circuit reference, and virtual circuit are included.
- **Output 5**: CORE : UNI : ALMRA01 Gi0/0/0 : 50m : Beeline : A00_VPLS_5200b02
  - *Explanation*: Type is CORE, role is UNI, connected device/interface, bandwidth (50m), carrier (Beeline), and circuit reference are included.

Explanation

- **Format Structure**: The description uses a colon (:) as the delimiter between fields (type, role, connected device/interface, bandwidth, carrier, circuit reference). If a field contains a :, it must be replaced (e.g., with a space).
- **Type and Role**: The type (LAN, WAN, CORE, INTERNET) and role (e.g., AP, P2P, NNI) define the interface's purpose and are critical for categorization and automation.
- **Connected Device**: Specifies the directly connected equipment and interface (e.g., US-TX-F-124-46-SL-01 G10/0).
- **Additional Fields**: For WAN/CORE, bandwidth (e.g., 500m for 500 Mbps), carrier, and circuit references provide operational details.
- **Maintenance Mode**: Prefixing with u (e.g., uLAN) indicates the interface is unmonitored (e.g., for maintenance), suppressing alerts.
- **Flexibility**: The manual suffix (e.g., || additional notes) allows extra information not yet automated.

SpaCy

Pros:

- **Efficiency**: SpaCy is lightweight and optimized for production use, making it faster for inference and suitable for real-time or resource-constrained environments (e.g., browser with Pyodide).
- **Custom NER Training**: SpaCy's built-in named entity recognition (NER) can be easily trained on the specific patterns from your document (e.g., LAN : AP : US-TX-F-124-46-SL-01 G10/0) with a relatively small dataset.
- **Rule-Based Support**: You can combine SpaCy's statistical models with rule-based patterns or regex (e.g., ^CORE : (UNI|NNI)) to handle structured formats effectively.
- **Ease of Integration**: Already integrated into your interface_parser.py script, and its async compatibility aligns with the provided loop structure.
- **Small Footprint**: The trained model size is manageable (e.g., ~10-50 MB), which is ideal for deployment in limited environments.

Cons:

- **Limited Contextual Understanding**: SpaCy's NER relies on local context and may struggle with complex, long-range dependencies or ambiguous natural language (e.g., inferring "WAN" from a circuit description).
- **Training Data Dependency**: Requires a well-annotated dataset, and performance depends heavily on the quality and quantity of examples (e.g., the TRAIN_DATA from the previous response).
- **Less Advanced Semantics**: Lacks the deep contextual understanding of Transformers, which could miss nuanced relationships in unstructured text.

Best Use Case:

SpaCy is ideal if your input is semi-structured or follows predictable patterns (as seen in the document's examples), and you need a lightweight, fast solution with custom training on a small dataset (e.g., 50-100 examples).

Transformer-Based Models (e.g., BERT, RoBERTa)

Pros:

- **Superior Contextual Understanding**: Transformers excel at capturing long-range dependencies and contextual nuances, making them better for unstructured or ambiguous text (e.g., inferring "WAN" or "maintenance" from narrative descriptions).

- **Pretrained Knowledge**: Models like BERT or RoBERTa come pretrained on vast corpora, providing a strong starting point for fine-tuning on your domain-specific data.
- **Higher Accuracy**: With fine-tuning, Transformers can achieve higher precision and recall for entity recognition, especially for overlapping or complex entities.
- **Transfer Learning**: Can leverage pre-trained models from Hugging Face (e.g., bert-base-uncased) and fine-tune with your data, reducing the need for extensive manual annotation.

Cons:

- **Resource Intensive**: Transformers require significant computational resources (e.g., GPU for training, larger memory footprint ~400-1000 MB), which may not suit browser deployment or low-power devices.
- **Complexity**: Fine-tuning requires more expertise and infrastructure (e.g., Hugging Face Transformers, PyTorch/TensorFlow), and integration into an async loop may need additional effort.
- **Slower Inference**: Inference is slower than SpaCy, which could impact real-time performance.
- **Overkill for Structured Data**: If the input is highly structured (e.g., LAN : AP : ...), the advanced capabilities might not be fully utilized.

Best Use Case:

Transformers are best if your input includes varied, unstructured natural language (e.g., free-text operational notes) or if you need state-of-the-art accuracy and have the resources for training and deployment.

Comparison for Your Requirement

- **Data Characteristics**: The document provides semi-structured examples (e.g., LAN : AP : ...) and some narrative text (e.g., "A router on a 500 Mbps..."). The task involves recognizing specific entities with clear delimiters, suggesting a structured parsing task.
- **Accuracy Needs**: High accuracy is important for automation (e.g., Kentik, Nautoobot), but the entity boundaries are well-defined, reducing the need for deep contextual analysis.
- **Deployment**: The async loop and Pyodide compatibility suggest a browser or lightweight environment, favoring SpaCy.
- **Training Effort**: With a small dataset (e.g., 50-100 examples), SpaCy can be trained effectively, while Transformers require more data and fine-tuning effort.

Recommendation

For your current requirement, **SpaCy** is the best choice because:

- The input data is semi-structured with predictable patterns, which SpaCy can handle well with custom NER training.
- The lightweight nature aligns with the async loop and potential browser deployment.
- The training process is straightforward with the provided examples, and you can augment it with rule-based checks (e.g., regex for ^LAN : u).

When to Consider Transformers

Switch to a Transformer-based model if:

- You encounter significant unstructured text requiring deep contextual understanding.
- You have access to a larger annotated dataset (e.g., 500+ examples) and computational resources for fine-tuning.
- You need to handle ambiguous or overlapping entities that SpaCy struggles with.

Next Steps with SpaCy

- Expand the TRAIN_DATA with more examples from the document (e.g., CORE:MGMT, INTERNET:ISP).
- Train and evaluate the model as outlined in the previous response.
- Add regex validation post-NER to ensure compliance with the document's format (e.g., check for proper delimiters).

Prerequisites

1. **Install Poetry**:
   - If not installed, follow the official instructions: Poetry Installation.
   - Verify with poetry --version.
2. **Python Environment**:
   - Ensure Python 3.9+ is installed (python --version).
3. **Dependencies**:
   - Use Poetry to manage dependencies.

Step-by-Step Usage

1. Set Up the Project

Create a new directory for your project:

```
mkdir interface_parser_project
cd interface_parser_project
```

Initialize a Poetry project:

```
poetry init
```

Add SpaCy and related dependencies to pyproject.toml:

```
[tool.poetry.dependencies]
python = "^3.9"
spacy = "^3.7.0"
spacy-lookups-data = "^1.0.5"

[tool.poetry.dev-dependencies]

[build-system]
requires = ["poetry-core>=1.0.0"]
build-backend = "poetry.core.masonry.api"
```

Install dependencies:

```
poetry install
```

Activate the Poetry shell:

```
poetry shell
```

Download the base SpaCy model:

```
python -m spacy download en_core_web_sm
```

2. Create and Save the Training Script

- Create a file named `train_ner.py` in the project directory with the training code from the previous response:

```
import spacy
from spacy.training.example import Example
import random

# Training data from the document
TRAIN_DATA = [
    ("A LAN switch port connected to an access point at US-TX-F-124-46-SL-01
on G10/0",
     {"entities": [(2, 5, "INTERFACE_TYPE"), (34, 36, "ROLE"), (47, 70,
"DEVICE")]}),
    ("A router on a 500 Mbps Megaport point-to-point NNI circuit to EU-GB-
LON-C-LONPWG-PE1 on xe-0/0/0.300 with circuit ABC420938",
     {"entities": [(9, 12, "INTERFACE_TYPE"), (44, 47, "ROLE"), (61, 94,
"DEVICE"), (13, 20, "BANDWIDTH"), (21, 28, "CARRIER"), (99, 108,
"CIRCUIT_REF")]}),
    ("A WAN CE-facing interface with a 50 Mbps circuit to Chevron CE ALMRA01
on Gi-0/0/0, carrier Beeline, circuit A00_VPLS_5200b02",
     {"entities": [(2, 5, "INTERFACE_TYPE"), (7, 10, "ROLE"), (47, 63,
"DEVICE"), (25, 32, "BANDWIDTH"), (71, 78, "CARRIER"), (87, 103,
"CIRCUIT_REF")]}),
    ("LAN : AP : US-TX-F-124-46-SL-01 G10/0",
     {"entities": [(0, 3, "INTERFACE_TYPE"), (6, 8, "ROLE"), (10, 33,
"DEVICE")]}),
    ("WAN : MPLS : 100m : Verizon : 329847298472",
     {"entities": [(0, 3, "INTERFACE_TYPE"), (6, 10, "ROLE"), (12, 16,
"BANDWIDTH"), (17, 24, "CARRIER"), (25, 36, "CIRCUIT_REF")]})
]

# Load base model
nlp = spacy.load("en_core_web_sm")

# Add new entity labels
ner = nlp.get_pipe("ner")
for label in ["INTERFACE_TYPE", "ROLE", "DEVICE", "BANDWIDTH", "CARRIER",
"CIRCUIT_REF"]:
    ner.add_label(label)

# Disable other pipes during training
other_pipes = [pipe for pipe in nlp.pipe_names if pipe != "ner"]
with nlp.disable_pipes(*other_pipes):
    optimizer = nlp.begin_training()
    for _ in range(20):
        random.shuffle(TRAIN_DATA)
        losses = {}
        for text, annotations in TRAIN_DATA:
            doc = nlp.make_doc(text)
            example = Example.from_dict(doc, annotations)
            nlp.update([example], drop=0.5, sgd=optimizer, losses=losses)
        print(losses)

# Save the trained model
nlp.to_disk("interface_ner_model")
print("Model saved to interface_ner_model")
```

3. Train the Model

- Run the training script within the Poetry environment:

```
python train_ner.py
```

Output will show training losses, and the model will be saved as interface_ner_model in the project directory.

4. Update and Use interface_parser.py

- Replace the interface_parser.py content with the updated version that uses the trained model. Create or overwrite `interface_parser.py` with:

```
import asyncio
import platform
import spacy
import re

# Load the trained model
nlp = spacy.load("interface_ner_model")

# Define patterns and roles based on requirements
INTERFACE_TYPES = ["LAN", "WAN", "CORE", "INTERNET"]
LAN_ROLES = ["AP", "MV", "MT", "MA", "MM", "UU", "UX"]
WAN_ROLES = ["WAN", "MPLS", "P2P", "P2MP", "SAT"]
CORE_ROLES = ["UNI", "NNI", "MGMT", "XNI"]
DELIMITER = ":"
MAINTENANCE_PREFIX = "u"

async def setup():
    pass

def extract_entities(text):
    doc = nlp(text)
    entities = {}
    for ent in doc.ents:
        if ent.label_ in ["INTERFACE_TYPE", "ROLE", "DEVICE", "BANDWIDTH",
"CARRIER", "CIRCUIT_REF"]:
            entities[ent.label_.lower().replace("_", "")] = ent.text
    return entities

def generate_description(entities, maintenance=False):
    parts = [entities.get("type", "")]
    if maintenance and parts[0]:
        parts[0] = MAINTENANCE_PREFIX + parts[0]
    parts.append(entities.get("role", ""))
    parts.append(entities.get("device", ""))
    if "bandwidth" in entities:
        parts.append(entities["bandwidth"])
    if "carrier" in entities:
        parts.append(entities["carrier"])
    if "circuit_ref" in entities:
        parts.append(entities["circuit_ref"])
    return DELIMITER.join(filter(None, parts))

async def update_loop():
    inputs = [
        "A LAN switch port connected to an access point at US-TX-F-124-46-
SL-01 on G10/0",
        "A router on a 500 Mbps Megaport point-to-point NNI circuit to EU-
GB-LON-C-LONPWG-PE1 on xe-0/0/0.300 with circuit ABC420938",
        "A WAN CE-facing interface with a 50 Mbps circuit to Chevron CE
ALMRA01 on Gi-0/0/0, carrier Beeline, circuit A00_VPLS_5200b02"
    ]

    for input_text in inputs:
        entities = extract_entities(input_text)
        desc = generate_description(entities)
        print(f"Generated Description: {desc}")
        maint_desc = generate_description(entities, maintenance=True)
        print(f"Maintenance Description: {maint_desc}")
        await asyncio.sleep(0.1)

if platform.system() == "Emscripten":
    asyncio.ensure_future(main())
else:
    if __name__ == "__main__":
        asyncio.run(main())

async def main():
    setup()
    while True:
        await update_loop()
        await asyncio.sleep(1.0 / 60)
```

5. Run the Parser

- Execute the parser script:

```
python interface_parser.py
```

Expected output will look like

```
Generated Description: LAN:AP:US-TX-F-124-46-SL-01 G10/0
Maintenance Description: uLAN:AP:US-TX-F-124-46-SL-01 G10/0
Generated Description: WAN:NNI:EU-GB-LON-C-LONPWG-PE1 xe-0/0/0.300:500
Mbps:Megaport:ABC420938
Maintenance Description: uWAN:NNI:EU-GB-LON-C-LONPWG-PE1 xe-0/0/0.300:500
Mbps:Megaport:ABC420938
Generated Description: WAN:CE:Chevron CE ALMRA01 on Gi-0/0/0:50
Mbps:Beeline:A00_VPLS_5200b02
Maintenance Description: uWAN:CE:Chevron CE ALMRA01 on Gi-0/0/0:50
Mbps:Beeline:A00_VPLS_5200b02
```

- - Note: Output depends on the trained model's accuracy. Adjust TRAIN_DATA if entities are misidentified.

6. Test and Iterate

- Add more examples to TRAIN_DATA in train_ner.py (e.g., from other document pages) and retrain if needed.
- Test with new inputs by modifying the inputs list in update_loop.
- Debug by checking entities output to ensure all required fields are extracted.

7. Deployment (Optional)

- For browser use with Pyodide, ensure the model is bundled (convert to .js using Pyodide’s tools) and adjust the script to load it appropriately.
- For local use, the current setup works within the Poetry environment.

To convert the provided code into a Python framework, we’ll structure it using a modular design with a package-based approach. This will improve maintainability, reusability, and scalability. We’ll create a framework called interface_parser_framework with separate modules for training, entity extraction, and description generation, adhering to Python best practices. The framework will use Poetry for dependency management and integrate with SpaCy for NLP tasks.

Project Structure

```
interface_parser_framework/
│
├── interface_parser_framework/
```

```
│   ├── __init__.py
│   ├── config.py
│   ├── models.py
│   ├── extractor.py
│   ├── generator.py
│   ├── utils.py
│   └── logging.py         # New logging configuration
│
├── tests/
│   ├── __init__.py
│   ├── test_extractor.py
│   └── test_generator.py
│
├── pyproject.toml
├── README.md
└── setup.py
```

Step-by-Step Implementation

1. Create logging.py

Add a new module to handle logging configuration and provide a logger instance.

interface_parser_framework/logging.py

```
import logging
import os
from datetime import datetime

# Configure logging
log_dir = "logs"
os.makedirs(log_dir, exist_ok=True)
log_file = os.path.join(log_dir,
f"interface_parser_{datetime.now().strftime('%Y%m%d_%H%M%S')}.log")

logging.basicConfig(
    level=logging.INFO,
    format="%(asctime)s - %(name)s - %(levelname)s - %(message)s",
    handlers=[
        logging.FileHandler(log_file),
        logging.StreamHandler()  # Optional: logs to console
    ]
)

logger = logging.getLogger(__name__)
```

```python
def get_logger(name):
    """Return a logger instance with the specified name."""
    return logging.getLogger(name)
```

Explanation:

- Creates a logs directory if it doesn’t exist.
- Generates a timestamped log file (e.g., interface_parser_20250801_1126.log).
- Sets up logging with INFO level, including timestamp, logger name, level, and message.
- Provides a get_logger function to access the logger elsewhere.

2. Update Existing Modules

Incorporate logging into each module using the get_logger function.

`interface_parser_framework/__init__.py`

```python
from .extractor import extract_entities
from .generator import generate_description
from .logging import get_logger

logger = get_logger(__name__)

__all__ = ["extract_entities", "generate_description"]
```

`interface_parser_framework/config.py`

```python
from .logging import get_logger

logger = get_logger(__name__)

INTERFACE_TYPES = ["LAN", "WAN", "CORE", "INTERNET"]
LAN_ROLES = ["AP", "MV", "MT", "MA", "MM", "UU", "UX"]
WAN_ROLES = ["WAN", "MPLS", "P2P", "P2MP", "SAT"]
CORE_ROLES = ["UNI", "NNI", "MGMT", "XNI"]
DELIMITER = ":"
MAINTENANCE_PREFIX = "u"
MODEL_PATH = "interface_ner_model"
```

`interface_parser_framework/models.py`

```
import spacy
from spacy.training.example import Example
import random
from .config import MODEL_PATH
from .logging import get_logger

logger = get_logger(__name__)

def train_model(train_data):
    """Train a custom NER model with the provided data."""
    logger.info("Starting model training")
    nlp = spacy.load("en_core_web_sm")
    ner = nlp.get_pipe("ner")
    for label in ["INTERFACE_TYPE", "ROLE", "DEVICE", "BANDWIDTH",
"CARRIER", "CIRCUIT_REF"]:
        ner.add_label(label)

    other_pipes = [pipe for pipe in nlp.pipe_names if pipe != "ner"]
    with nlp.disable_pipes(*other_pipes):
        optimizer = nlp.begin_training()
        for _ in range(20):
            random.shuffle(train_data)
            losses = {}
            for text, annotations in train_data:
                doc = nlp.make_doc(text)
                example = Example.from_dict(doc, annotations)
                nlp.update([example], drop=0.5, sgd=optimizer,
losses=losses)
            logger.debug(f"Training iteration losses: {losses}")
    nlp.to_disk(MODEL_PATH)
    logger.info(f"Model saved to {MODEL_PATH}")
    return nlp

def load_model():
    """Load the trained NER model."""
    logger.info("Loading trained model")
    return spacy.load(MODEL_PATH)
```

`interface_parser_framework/extractor.py`

```
import spacy
from .models import load_model
from .config import INTERFACE_TYPES, LAN_ROLES, WAN_ROLES, CORE_ROLES
from .logging import get_logger
```

```python
logger = get_logger(__name__)

nlp = load_model()

def extract_entities(text):
    """Extract entities from the input text using the trained model."""
    logger.debug(f"Extracting entities from text: {text}")
    doc = nlp(text)
    entities = {}
    for ent in doc.ents:
        if ent.label_ in ["INTERFACE_TYPE", "ROLE", "DEVICE", "BANDWIDTH",
"CARRIER", "CIRCUIT_REF"]:
            entities[ent.label_.lower().replace("_", "")] = ent.text
            logger.debug(f"Extracted entity: {ent.label_} = {ent.text}")
    if not entities:
        logger.warning("No entities extracted from text")
    return entities
```

`interface_parser_framework/generator.py`

```python
from .config import DELIMITER, MAINTENANCE_PREFIX
from .logging import get_logger

logger = get_logger(__name__)

def generate_description(entities, maintenance=False):
    """Generate a formatted interface description from extracted
entities."""
    logger.debug(f"Generating description with entities: {entities},
maintenance: {maintenance}")
    parts = [entities.get("type", "")]
    if maintenance and parts[0]:
        parts[0] = MAINTENANCE_PREFIX + parts[0]
    parts.append(entities.get("role", ""))
    parts.append(entities.get("device", ""))
    if "bandwidth" in entities:
        parts.append(entities["bandwidth"])
    if "carrier" in entities:
        parts.append(entities["carrier"])
    if "circuit_ref" in entities:
        parts.append(entities["circuit_ref"])
    desc = DELIMITER.join(filter(None, parts))
    logger.info(f"Generated description: {desc}")
    return desc
```

`interface_parser_framework/utils.py`

```python
import asyncio
from .extractor import extract_entities
from .generator import generate_description
from .logging import get_logger

logger = get_logger(__name__)

async def async_process(inputs):
    """Process inputs asynchronously and yield results."""
    logger.info("Starting asynchronous processing of inputs")
    for input_text in inputs:
        logger.debug(f"Processing input: {input_text}")
        entities = extract_entities(input_text)
        desc = generate_description(entities)
        maint_desc = generate_description(entities, maintenance=True)
        yield desc, maint_desc
        await asyncio.sleep(0.1)
```

`main.py`

```python
import asyncio
import platform
from interface_parser_framework import extract_entities,
generate_description, utils
from interface_parser_framework.logging import get_logger

logger = get_logger(__name__)

async def main():
    logger.info("Starting main application")
    inputs = [
        "A LAN switch port connected to an access point at US-TX-F-124-46-
SL-01 on G10/0",
        "A router on a 500 Mbps Megaport point-to-point NNI circuit to EU-
GB-LON-C-LONPWG-PE1 on xe-0/0/0.300 with circuit ABC420938",
        "A WAN CE-facing interface with a 50 Mbps circuit to Chevron CE
ALMRA01 on Gi-0/0/0, carrier Beeline, circuit A00_VPLS_5200b02"
    ]

    async for desc, maint_desc in utils.async_process(inputs):
        logger.info(f"Generated Description: {desc}")
        logger.info(f"Maintenance Description: {maint_desc}")
```

```
if platform.system() == "Emscripten":
    asyncio.ensure_future(main())
else:
    if __name__ == "__main__":
        asyncio.run(main())
```

`train_model.py`

```
from interface_parser_framework.models import train_model
from interface_parser_framework.logging import get_logger

logger = get_logger(__name__)

TRAIN_DATA = [
    ("A LAN switch port connected to an access point at US-TX-F-124-46-SL-01 
on G10/0",
     {"entities": [(2, 5, "INTERFACE_TYPE"), (34, 36, "ROLE"), (47, 70, 
"DEVICE")]}),
    # Add more examples as needed
]

if __name__ == "__main__":
    logger.info("Starting model training process")
    train_model(TRAIN_DATA)
```

3. Update Tests

Ensure tests log appropriately.

`tests/test_extractor.py`

```
from interface_parser_framework import extract_entities
from interface_parser_framework.logging import get_logger

logger = get_logger(__name__)

def test_extract_entities():
    logger.info("Running test_extract_entities")
    text = "A LAN switch port connected to an access point at US-TX-F-124-
46-SL-01 on G10/0"
    entities = extract_entities(text)
    assert "type" in entities and entities["type"] == "LAN"
    assert "role" in entities and entities["role"] == "AP"
    assert "device" in entities
    logger.info("test_extract_entities passed")
```

`tests/test_generator.py`

```
from interface_parser_framework import generate_description
from interface_parser_framework.logging import get_logger

logger = get_logger(__name__)

def test_generate_description():
    logger.info("Running test_generate_description")
    entities = {"type": "LAN", "role": "AP", "device": "US-TX-F-124-46-SL-01 G10/0"}
    desc = generate_description(entities)
    assert desc == "LAN:AP:US-TX-F-124-46-SL-01 G10/0"
    maint_desc = generate_description(entities, maintenance=True)
    assert maint_desc == "uLAN:AP:US-TX-F-124-46-SL-01 G10/0"
    logger.info("test_generate_description passed")
```

4.. Run the Project

- Train the model:

```
python train_model.py
```

Run the main script:

```
python main.py
```

Check the log file in the logs directory (e.g., logs/interface_parser_20250801_1126.log).

```
2025-08-01 11:26:00,123 - interface_parser_framework.models - INFO - Starting model training
2025-08-01 11:26:01,456 - interface_parser_framework.models - DEBUG - Training iteration losses: {'ner': 0.123}
2025-08-01 11:26:01,789 - interface_parser_framework.models - INFO - Model saved to interface_ner_model
2025-08-01 11:26:02,123 - interface_parser_framework - INFO - Starting main application
2025-08-01 11:26:02,456 - interface_parser_framework.utils - INFO - Starting
```

```
asynchronous processing of inputs
2025-08-01 11:26:02,789 - interface_parser_framework.utils - DEBUG -
Processing input: A LAN switch port connected to an access point at US-TX-F-
124-46-SL-01 on G10/0
2025-08-01 11:26:02,890 - interface_parser_framework.extractor - DEBUG -
Extracting entities from text: A LAN switch port connected to an access
point at US-TX-F-124-46-SL-01 on G10/0
2025-08-01 11:26:02,991 - interface_parser_framework.extractor - DEBUG -
Extracted entity: INTERFACE_TYPE = LAN
2025-08-01 11:26:02,992 - interface_parser_framework.generator - INFO -
Generated description: LAN:AP:US-TX-F-124-46-SL-01 G10/0
...
```

Notes

- **Log Levels**: Use INFO for key events, DEBUG for detailed steps, and WARNING/ERROR for issues.
- **Customization**: Adjust log levels or add file rotation (e.g., using RotatingFileHandler) in logging.py if needed.

How to train the model

To train the SpaCy NER model in the interface_parser_framework using input data stored in CSV files, you'll need to read the CSV file, parse its contents into a format compatible with SpaCy's training data (text and entity annotations), and then use the train_model function. Below are the steps to achieve this, assuming the CSV file contains text inputs and their corresponding entity annotations.

Assumptions

- The CSV file has columns: text (the input description) and entities (a string representation of entity annotations, e.g., [(2, 5, "INTERFACE_TYPE"), (34, 36, "ROLE"), ...]).
- The CSV file is named training_data.csv and located in the project directory.

Updated Project Structure

No change to the structure; we'll modify train_model.py to handle CSV input.

Steps to Train the Model with CSV Files

1. Prepare the CSV File

Create a training_data.csv file with the following format:

```
text,entities
"A LAN switch port connected to an access point at US-TX-F-124-46-SL-01 on G10/0","[(2, 5, \"INTERFACE_TYPE\"), (34, 36, \"ROLE\"), (47, 70, \"DEVICE\")]"
"A router on a 500 Mbps Megaport point-to-point NNI circuit to EU-GB-LON-C-LONPWG-PE1 on xe-0/0/0.300 with circuit ABC420938","[(9, 12, \"INTERFACE_TYPE\"), (44, 47, \"ROLE\"), (61, 94, \"DEVICE\"), (13, 20, \"BANDWIDTH\"), (21, 28, \"CARRIER\"), (99, 108, \"CIRCUIT_REF\")]"
"A core router connected to EU-GB-LON-C-LONPWG-PE1 xe-0/0/0.300 on a 50m Megaport NNI circuit with ABC420938","[(2, 6, \"INTERFACE_TYPE\"), (53, 56, \"ROLE\"), (21, 44, \"DEVICE\"), (47, 50, \"BANDWIDTH\"), (51, 58, \"CARRIER\"), (63, 72, \"CIRCUIT_REF\")]"
```

- **Notes**:
  - Use double quotes around the entities string and escape inner quotes with ".
  - Adjust character positions (start, end) based on the exact text in each row.
  - You can generate this CSV manually or programmatically from annotated data.

2. Update train_model.py

Modify train_model.py to read the CSV file and train the model. Install pandas for CSV handling:
Update pyproject.toml:

```
[tool.poetry.dependencies]
python = "^3.9"
spacy = "^3.7.0"
spacy-lookups-data = "^1.0.5"
pandas = "^2.0.0"  # Add pandas for CSV parsing

[tool.poetry.dev-dependencies]
pytest = "^7.0.0"
```

```
poetry install
```

- Update train_model.py:

`train_model.py`

```python
import pandas as pd
from interface_parser_framework.models import train_model
from interface_parser_framework.logging import get_logger
import ast  # To safely evaluate the entities string

logger = get_logger(__name__)

def load_training_data(csv_path="training_data.csv"):
    """Load training data from a CSV file."""
    logger.info(f"Loading training data from {csv_path}")
    df = pd.read_csv(csv_path)
    train_data = []
    for _, row in df.iterrows():
        text = row["text"]
        entities_str = row["entities"]
        # Safely evaluate the string representation of the entities list
        entities = ast.literal_eval(entities_str)
        annotations = {"entities": entities}
        train_data.append((text, annotations))
    logger.info(f"Loaded {len(train_data)} training examples")
    return train_data

if __name__ == "__main__":
    logger.info("Starting model training process with CSV input")
    train_data = load_training_data()
    train_model(train_data)
```

- **Explanation**:
  - pandas.read_csv reads the CSV file into a DataFrame.
  - ast.literal_eval safely converts the entities string (e.g., "[(2, 5, "INTERFACE_TYPE")]") into a Python list of tuples.
  - The resulting train_data is a list of (text, annotations) tuples compatible with SpaCy's training.

3. Ensure models.py Supports New Entities

If your CSV includes new entity types (e.g., MAINTENANCE), ensure models.py includes them (already updated in the previous response with MAINTENANCE).

`interface_parser_framework/models.py (partial)`

```python
# ... (previous imports and logger setup)

def train_model(train_data):
    """Train a custom NER model with the provided data."""
    logger.info("Starting model training")
    nlp = spacy.load("en_core_web_sm")
    ner = nlp.get_pipe("ner")
    for label in ["INTERFACE_TYPE", "ROLE", "DEVICE", "BANDWIDTH",
"CARRIER", "CIRCUIT_REF", "MAINTENANCE"]:
        ner.add_label(label)

    # ... (rest of the function remains the same)
```

4. Retrain the Model

- Ensure training_data.csv is in the project directory.
  Run the training script:

```
python train_model.py
```

- **Output**: The logger will indicate the number of examples loaded and training progress. The model will be saved to interface_ner_model.

5. Test the Updated Model

- Run main.py to verify the model processes new inputs from the CSV:

```
python main.py
```

Obsereve the logs

6. Add More Data

- To include additional CSV files or rows:
  - Append new rows to training_data.csv or create additional files (e.g., training_data_2.csv).
  - Update load_training_data to handle multiple files:

E2E

Step-by-Step Project Setup

1. Initialize the Project

- Create a new directory and navigate into it

```
mkdir interface_parser_framework
cd interface_parser_framework
git init
```

Edit pyproject.toml to include dependencies

```
[tool.poetry]
name = "interface_parser_framework"
version = "0.1.0"
description = "A framework for parsing and generating network interface
descriptions using SpaCy."
authors = ["Your Name <your.email@example.com>"]
readme = "README.md"
packages = [{include = "interface_parser_framework"}]

[tool.poetry.dependencies]
python = "^3.9"
spacy = "^3.7.0"
spacy-lookups-data = "^1.0.5"
pandas = "^2.0.0"

[tool.poetry.dev-dependencies]
pytest = "^7.0.0"

[build-system]
requires = ["poetry-core>=1.0.0"]
build-backend = "poetry.core.masonry.api"
```

```
poetry install
poetry shell
python -m spacy download en_core_web_sm
```

3. Create Project Structure

Create the following directory structure manually or with a script:

```
interface_parser_framework/
|
├── interface_parser_framework/
|   ├── __init__.py
|   ├── config.py
|   ├── models.py
|   ├── extractor.py
```

```
│   ├── generator.py
│   ├── utils.py
│   └── logging.py
│
├── tests/
│   ├── __init__.py
│   ├── test_extractor.py
│   └── test_generator.py
│
├── training_data.csv
├── train_model.py
├── main.py
├── pyproject.toml
├── README.md
└── setup.py
```

Create empty `__init__.py` files in interface_parser_framework/ and tests/ to mark them as packages

4. Implement the Code

`interface_parser_framework/__init__.py`

```python
from .extractor import extract_entities
from .generator import generate_description
from .logging import get_logger

logger = get_logger(__name__)

__all__ = ["extract_entities", "generate_description"]
```

`interface_parser_framework/config.py`

```python
from .logging import get_logger

logger = get_logger(__name__)

INTERFACE_TYPES = ["LAN", "WAN", "CORE", "INTERNET"]
LAN_ROLES = ["AP", "MV", "MT", "MA", "MM", "UU", "UX"]
WAN_ROLES = ["WAN", "MPLS", "P2P", "P2MP", "SAT"]
CORE_ROLES = ["UNI", "NNI", "MGMT", "XNI"]
DELIMITER = ":"
```

```
MAINTENANCE_PREFIX = "u"
MODEL_PATH = "interface_ner_model"
```

`interface_parser_framework/models.py`

```


import spacy
from spacy.training.example import Example
import random
from .config import MODEL_PATH
from .logging import get_logger

logger = get_logger(__name__)

def train_model(train_data):
    logger.info("Starting model training")
    nlp = spacy.load("en_core_web_sm")
    ner = nlp.get_pipe("ner")
    for label in ["INTERFACE_TYPE", "ROLE", "DEVICE", "BANDWIDTH",
"CARRIER", "CIRCUIT_REF", "MAINTENANCE"]:
        ner.add_label(label)

    other_pipes = [pipe for pipe in nlp.pipe_names if pipe != "ner"]
    with nlp.disable_pipes(*other_pipes):
        optimizer = nlp.begin_training()
        for _ in range(20):
            random.shuffle(train_data)
            losses = {}
            for text, annotations in train_data:
                doc = nlp.make_doc(text)
                example = Example.from_dict(doc, annotations)
                nlp.update([example], drop=0.5, sgd=optimizer,
losses=losses)
            logger.debug(f"Training iteration losses: {losses}")
    nlp.to_disk(MODEL_PATH)
    logger.info(f"Model saved to {MODEL_PATH}")
    return nlp

def load_model():
    logger.info("Loading trained model")
    return spacy.load(MODEL_PATH)
```

`interface_parser_framework/extractor.py`

```
import spacy
from .models import load_model
from .config import INTERFACE_TYPES, LAN_ROLES, WAN_ROLES, CORE_ROLES
from .logging import get_logger

logger = get_logger(__name__)

nlp = load_model()

def extract_entities(text):
    logger.debug(f"Extracting entities from text: {text}")
    doc = nlp(text)
    entities = {}
    for ent in doc.ents:
        if ent.label_ in ["INTERFACE_TYPE", "ROLE", "DEVICE", "BANDWIDTH",
"CARRIER", "CIRCUIT_REF", "MAINTENANCE"]:
            entities[ent.label_.lower().replace("_", "")] = ent.text
            logger.debug(f"Extracted entity: {ent.label_} = {ent.text}")
    if not entities:
        logger.warning("No entities extracted from text")
    return entities
```

`interface_parser_framework/generator.py`

```
from .config import DELIMITER, MAINTENANCE_PREFIX
from .logging import get_logger

logger = get_logger(__name__)

def generate_description(entities, maintenance=False):
    logger.debug(f"Generating description with entities: {entities},
maintenance: {maintenance}")
    parts = [entities.get("type", "")]
    if maintenance and parts[0]:
        parts[0] = MAINTENANCE_PREFIX + parts[0]
    parts.append(entities.get("role", ""))
    parts.append(entities.get("device", ""))
    if "bandwidth" in entities:
        parts.append(entities["bandwidth"])
    if "carrier" in entities:
        parts.append(entities["carrier"])
    if "circuit_ref" in entities:
        parts.append(entities["circuit_ref"])
    desc = DELIMITER.join(filter(None, parts))
    logger.info(f"Generated description: {desc}")
    return desc
```

`interface_parser_framework/utils.py`

```
import asyncio
from .extractor import extract_entities
from .generator import generate_description
from .logging import get_logger

logger = get_logger(__name__)

async def async_process(inputs):
    logger.info("Starting asynchronous processing of inputs")
    for input_text in inputs:
        logger.debug(f"Processing input: {input_text}")
        entities = extract_entities(input_text)
        desc = generate_description(entities)
        maint_desc = generate_description(entities, maintenance=True)
        yield desc, maint_desc
        await asyncio.sleep(0.1)
```

`interface_parser_framework/logging.py`

```
import logging
import os
from datetime import datetime

log_dir = "logs"
os.makedirs(log_dir, exist_ok=True)
log_file = os.path.join(log_dir,
f"interface_parser_{datetime.now().strftime('%Y%m%d_%H%M%S')}.log")

logging.basicConfig(
    level=logging.INFO,
    format="%(asctime)s - %(name)s - %(levelname)s - %(message)s",
    handlers=[
        logging.FileHandler(log_file),
        logging.StreamHandler()
    ]
)
```

```
logger = logging.getLogger(__name__)

def get_logger(name):
    return logging.getLogger(name)
```

`training_data.csv`

```
text,entities
"A LAN switch port connected to an access point at US-TX-F-124-46-SL-01 on
G10/0","[(2, 5, \"INTERFACE_TYPE\"), (34, 36, \"ROLE\"), (47, 70,
\"DEVICE\")]"
"A router on a 500 Mbps Megaport point-to-point NNI circuit to EU-GB-LON-C-
LONPWG-PE1 on xe-0/0/0.300 with circuit ABC420938","[(9, 12,
\"INTERFACE_TYPE\"), (44, 47, \"ROLE\"), (61, 94, \"DEVICE\"), (13, 20,
\"BANDWIDTH\"), (21, 28, \"CARRIER\"), (99, 108, \"CIRCUIT_REF\")]"
"A core router connected to EU-GB-LON-C-LONPWG-PE1 xe-0/0/0.300 on a 50m
Megaport NNI circuit with ABC420938","[(2, 6, \"INTERFACE_TYPE\"), (53, 56,
\"ROLE\"), (21, 44, \"DEVICE\"), (47, 50, \"BANDWIDTH\"), (51, 58,
\"CARRIER\"), (63, 72, \"CIRCUIT_REF\")]"
"An INTERNET connection to Cogent ISP on a 1g circuit with reference
X0908204","[(3, 11, \"INTERFACE_TYPE\"), (24, 30, \"ROLE\"), (12, 18,
\"CARRIER\"), (31, 33, \"BANDWIDTH\"), (48, 56, \"CIRCUIT_REF\")]"
"A LAN port taken down for maintenance at US-TX-F-124-46-SL-01 G10/0","[(2,
5, \"INTERFACE_TYPE\"), (27, 38, \"DEVICE\"), (21, 26, \"MAINTENANCE\")]"
```

`train_model.py`

```python
import pandas as pd
from interface_parser_framework.models import train_model
from interface_parser_framework.logging import get_logger
import ast

logger = get_logger(__name__)

def load_training_data(csv_path="training_data.csv"):
    logger.info(f"Loading training data from {csv_path}")
    df = pd.read_csv(csv_path)
    train_data = []
    for _, row in df.iterrows():
        text = row["text"]
        entities_str = row["entities"]
        entities = ast.literal_eval(entities_str)
        annotations = {"entities": entities}
        train_data.append((text, annotations))
    logger.info(f"Loaded {len(train_data)} training examples")
    return train_data

if __name__ == "__main__":
    logger.info("Starting model training process with CSV input")
    train_data = load_training_data()
    train_model(train_data)
```

`main.py`

```
import asyncio
import platform
from interface_parser_framework import extract_entities,
generate_description, utils
from interface_parser_framework.logging import get_logger

logger = get_logger(__name__)

async def main():
    logger.info("Starting main application")
    inputs = [
        "A LAN switch port connected to an access point at US-TX-F-124-46-
SL-01 on G10/0",
        "A core router connected to EU-GB-LON-C-LONPWG-PE1 xe-0/0/0.300 on a
50m Megaport NNI circuit with ABC420938",
        "An INTERNET connection to Cogent ISP on a 1g circuit with reference
X0908204"
    ]

    async for desc, maint_desc in utils.async_process(inputs):
        logger.info(f"Generated Description: {desc}")
        logger.info(f"Maintenance Description: {maint_desc}")

if platform.system() == "Emscripten":
    asyncio.ensure_future(main())
else:
    if __name__ == "__main__":
        asyncio.run(main())
```

`tests/test_extractor.py`

```
from interface_parser_framework import extract_entities
from interface_parser_framework.logging import get_logger

logger = get_logger(__name__)

def test_extract_entities():
    logger.info("Running test_extract_entities")
    text = "A LAN switch port connected to an access point at US-TX-F-124-
46-SL-01 on G10/0"
    entities = extract_entities(text)
    assert "type" in entities and entities["type"] == "LAN"
    assert "role" in entities and entities["role"] == "AP"
    assert "device" in entities
    logger.info("test_extract_entities passed")
```

`tests/test_generator.py`

```
from interface_parser_framework import generate_description
from interface_parser_framework.logging import get_logger

logger = get_logger(__name__)

def test_generate_description():
    logger.info("Running test_generate_description")
    entities = {"type": "LAN", "role": "AP", "device": "US-TX-F-124-46-SL-01
G10/0"}
    desc = generate_description(entities)
    assert desc == "LAN:AP:US-TX-F-124-46-SL-01 G10/0"
    maint_desc = generate_description(entities, maintenance=True)
    assert maint_desc == "uLAN:AP:US-TX-F-124-46-SL-01 G10/0"
    logger.info("test_generate_description passed")
```

`setup.py`

```
from setuptools import setup, find_packages

setup(
    name="interface_parser_framework",
    version="0.1.0",
    packages=find_packages(),
    install_requires=[
        "spacy>=3.7.0",
        "spacy-lookups-data>=1.0.5",
        "pandas>=2.0.0",
    ],
)
```

`README.md`

~~~
# Interface Parser Framework

A Python framework for parsing and generating network interface descriptions
using SpaCy.

## Installation
```bash
poetry install
python -m spacy download en_core_web_sm
~~~

Usage

1. Train the model:

```
python train_model.py
```

2. Run the application:

```
python main.py
```

Adding New Data

Update `training_data.csv` with new examples and rerun `train_model.py`.

Testing

```
poetry run pytest
```

License

MIT

~~~
#### 5. **Run the Project**
- Train the model:
  ```bash
  python train_model.py
~~~

- Expected output: Model training logs and a saved `interface_ner_model`.
- Run the main script:

```
python main.py
```

Expected output (in console and `logs/interface_parser_20250801_1144.log)`:

```
2025-08-01 11:44:00,123 - interface_parser_framework - INFO - Starting main
application
2025-08-01 11:44:00,124 - interface_parser_framework.utils - INFO - Starting
asynchronous processing of inputs
2025-08-01 11:44:00,125 - interface_parser_framework.utils - DEBUG -
Processing input: A LAN switch port connected to an access point at US-TX-F-
124-46-SL-01 on G10/0
2025-08-01 11:44:00,126 - interface_parser_framework.generator - INFO -
Generated description: LAN:AP:US-TX-F-124-46-SL-01 G10/0
2025-08-01 11:44:00,127 - interface_parser_framework.generator - INFO -
Generated description: uLAN:AP:US-TX-F-124-46-SL-01 G10/0
...
```

- Run tests:

```
poetry run pytest
```

6. Enhancements and Maintenance

- **Add More Data**: Append rows to `training_data.csv` and retrain.
- **Debugging**: Check `logs/` for issues.
- **Deployment**: For Pyodide, bundle the model using Pyodide tools.

Notes

- **Time**: This setup should take 30-40 minutes as of 11:44 AM IST on August 01, 2025.
- **Scalability**: The framework supports adding more CSV files or modules as needed.
- **Backup**: Save interface_ner_model before retraining.

This end-to-end project provides a robust foundation for your interface parsing needs.