Chapter 6

Pipelines in Hugging Face Diffusers

Introduction

Pipelines are a fundamental component of the Hugging Face Diffusers library, enabling the smooth coordination of various **natural language processing** (**NLP**) tasks within a single workflow. This chapter concentrates on understanding, building, customizing, and deploying NLP pipelines. With detailed examples, case studies, and best practices, you will learn how to develop robust pipelines that combine multiple models, handle complex tasks, and scale effectively for production settings.

Structure

This chapter covers the following topics:

* Understanding pipelines and their role in NLP workflows
* Building and customizing NLP pipelines
* Practical applications of pipelines
* Integrating multiple models into a single pipeline
* Managing and scaling pipelines for production use

Objectives

By the end of this chapter, readers will be able to understand the role of pipelines in NLP by familiarizing themselves with the concept and importance of pipelines within the Hugging Face Diffusers library for streamlining NLP workflows. They will be able to build and customize pipelines, gaining practical knowledge to create and change custom NLP pipelines tailored to specific tasks and requirements. Readers will also learn how to integrate schedulers into pipelines, adapting them for various schedulers to improve training and inference processes for diverse NLP applications. Additionally, they will master strategies for deploying pipelines in production, focusing on scaling and deploying NLP pipelines in real-world, production-level environments to ensure performance and reliability. Finally, they will be able to apply pipelines to practical use cases by exploring case studies that demonstrate how pipelines can be used to address complex NLP challenges and deliver impactful solutions across industries.

Understanding pipelines and their role in NLP workflows

Pipelines are vital to modern NLP workflows, providing a structured way to manage a series of operations, from raw data preprocessing to final deployment. These modular systems allow developers to focus on individual components while ensuring scalability, reproducibility, and efficient task execution. By encapsulating processes like tokenization, modeling, and post-processing, pipelines serve as the foundation for building robust NLP systems, supporting various applications from text classification to entity recognition. [1]

Definition and importance

Pipelines simplify the complexities of NLP by integrating various tasks into a unified workflow. Their modular nature allows for easy debugging, updating, and scaling of individual components without disrupting the entire system. For instance, a pipeline for sentiment analysis might include tokenization, feature extraction, model inference, and output post-processing, all of which are seamlessly connected. This structure enables consistent results and ensures that workflows are still adaptable to changing data or task requirements. [2]

Key components

The success of any NLP pipeline relies on its core components, which work together to turn raw text into useful insights. Each part plays a crucial role in ensuring the pipeline runs smoothly, providing accurate results for specific tasks. From data cleaning and preparation to deploying models in production, the following linked steps form the foundation of a strong NLP system.[1]:

1. **Data preprocessing**: Data preprocessing is the first and most crucial step in any NLP pipeline. It involves cleaning and standardizing raw text data to ensure it is compatible with the following processes. Techniques include removing stop words, punctuation, and special characters, normalizing text (such as converting to lowercase), and managing missing values. For example, when working with Twitter data, preprocessing may involve removing hashtags and user mentions while retaining the relevant text. Effective preprocessing reduces noise and improves model performance by providing cleaner input data. [3]
2. **Tokenization**: Tokenization breaks text into smaller units, such as words, subwords, or characters, which machine learning models can process. For example, BERT uses WordPiece tokenization to handle out-of-vocabulary words by splitting them into subwords [4].

Tokenization ensures that the text representation matches the model's architecture. For example, the sentence *Natural language processing is exciting!* might be tokenized into the sequence: ["natural", "language", "processing", "is", "exciting", "!"]. Tokenization is especially important for tasks involving multilingual datasets or informal text.

1. **Feature extraction**: Feature extraction transforms tokens into numerical data that models can interpret. Common methods include embeddings like Word2Vec or contextual embeddings from BERT [5]. For example, the sentence *I love NLP* might be represented as a vector [0.5, 0.8, 0.3], capturing semantic relationships between words. Advanced feature extraction techniques enable models to understand linguistic details, thereby improving performance in tasks such as sentiment analysis and machine translation.
2. **Modeling**: The modeling stage applies **machine learning** (**ML**) or deep learning algorithms to solve specific NLP tasks. Pre-trained transformer models, such as RoBERTa or DistilBERT, can be fine-tuned for tasks like sentiment analysis or named entity recognition [6]. For example, a classifier might assign the sentence *This product is amazing* to the label *positive.* Modeling forms the core of the pipeline, using pre-trained architectures to achieve high accuracy with minimal data.
3. **Post-processing**: Post-processing improves model output to meet task-specific needs. For example, in a machine translation pipeline, it may involve detokenizing the translated text and correcting grammatical errors. Similarly, in a named entity recognition task, it could map entity IDs back to their original terms in the text. This step makes sure that the output is easy to read and matches practical use needs [7].
4. **Integration and deployment**: Integration and deployment involve embedding the pipeline into production environments. This step ensures that the pipeline interacts seamlessly with **application programming interfaces** (**APIs**), databases, or other systems. Techniques such as containerization (e.g., using Docker) and cloud-based deployment help scalability and reliability. For instance, a sentiment analysis pipeline deployed as an API might process thousands of user reviews in real-time, delivering actionable insights for businesses.

Building and customizing NLP pipelines

Building and customizing NLP pipelines is a vital skill for developing efficient, task-specific language processing solutions. Pipelines offer a structured and modular approach, enabling the adaptation and extension of existing frameworks to meet specific needs. Custom pipelines can be tailored to fit the details of different datasets, domains, or tasks, helping developers design workflows that enhance both accuracy and efficiency. This adaptability has made pipelines indispensable in applications such as social media analysis, customer service automation, and more [1].

Creating custom pipelines

Custom pipelines are designed to cater to specialized NLP tasks, offering flexibility for applications like analyzing informal language on social media or processing domain-specific texts. For instance, a pipeline for analyzing tweets might include preprocessing steps to manage unique Twitter elements such as hashtags, emojis, and slang. These pipelines allow developers to focus on solving specific problems without being constrained by the general-purpose nature of prebuilt solutions.

*Figure 6.1* illustrates the modular architecture of a typical NLP pipeline in Hugging Face Diffusers, from raw text input to final deployment, and the process spans from data preprocessing and tokenization through modeling, post-processing, and real-world deployment.:

A diagram of a data processing process

AI-generated content may be incorrect.

**Figure 6.1**: Core stages of an NLP pipeline using Hugging Face Diffusers

Integrating multiple NLP tasks

Modern NLP challenges often involve combining multiple tasks into a single pipeline. Merging tasks such as sentiment analysis, entity recognition, and text classification enables a more comprehensive understanding of text. For example, in customer service, a pipeline might decide the sentiment of a customer's query, extract entities such as product names, and classify the topic to direct the query to the proper support agent. This multi-task approach improves the pipeline's ability to manage complex, real-world situations effectively.

**Example**: A pipeline integrating sentiment analysis and named entity recognition.

The following code snippet proves how to integrate sentiment analysis and **named entity recognition** (**NER**) within a single workflow. This example uses the Hugging Face pipeline class to create an efficient and reusable framework for analyzing text.

`python

from transformers import pipeline

# Load pre-trained pipelines  
sentiment\_pipeline = pipeline("sentiment-analysis")  
ner\_pipeline = pipeline("ner")

# Sample text  
text = "Hugging Face tools are innovative and used at Zinnia AI."

# Perform sentiment analysis and entity recognition  
sentiment = sentiment\_pipeline(text)  
entities = ner\_pipeline(text)

print("Sentiment:", sentiment)  
print("Entities:", entities)

`

This example begins by importing the pipeline class from the Hugging Face library, which simplifies the creation of pre-trained NLP models for specific tasks. Two pipelines are instantiated - one for sentiment analysis and another for named entity recognition.

The sentiment\_pipeline analyzes the sentiment of the provided text ("Hugging Face tools are innovative and used at Zinnia AI"), predicting whether it is positive, negative, or neutral, along with a confidence score. Simultaneously, the ner\_pipeline detects entities within the text, such as proper nouns, organizations, or locations. For this input, the pipeline might identify "Hugging Face" and "Zinnia AI" as entities.

By combining the results of both pipelines, this workflow shows how multiple NLP tasks can be executed in parallel to provide a more comprehensive analysis of the input text. This integration is particularly valuable in scenarios requiring comprehensive text understanding, such as content categorization, customer feedback analysis, or automated report generation.

Practical applications of pipelines

Pipelines in NLP serve as a comprehensive framework for automating complex language processing tasks, enabling organizations to derive actionable insights with minimal effort. Their flexibility and modularity allow for seamless integration into diverse workflows, ranging from sentiment analysis to content moderation. This section explores a practical application - sentiment analysis - illustrating how pipelines facilitate real-time decision-making for brands by analyzing customer opinions on platforms such as social media.

Sentiment analysis pipelines are invaluable tools for monitoring customer sentiment across diverse channels, including social media, reviews, and feedback forms. By processing vast amounts of textual data, these pipelines allow brands to find trends, gauge public opinion, and respond proactively to emerging issues. For example, a company can detect a surge in negative feedback on social media and address the root cause before it escalates into a larger crisis [8]. Advanced sentiment analysis workflows use pre-trained models that assess polarity, whether a sentiment is positive, negative, or neutral, and return confidence scores, ensuring reliable insights.

Case Study: Sentiment analysis

The following implementation highlights a sentiment analysis pipeline designed to process customer feedback. This example demonstrates how brands can automate the evaluation of textual inputs, categorizing sentiments to inform strategic decision-making:

`python

from transformers import pipeline

# Initialize pipeline  
sentiment\_analysis = pipeline("sentiment-analysis")

# Analyze customer feedback  
feedback = ["Great product!", "Terrible customer service."]  
results = sentiment\_analysis(feedback)

for result in results:  
 print(f"Sentiment: {result['label']}, Confidence: {result['score']:.2f}")

`

This Python script proves the simplicity and effectiveness of the Hugging Face pipeline class for performing sentiment analysis. The process begins by importing the pipeline module, which serves as a high-level interface for using pre-trained NLP models. The sentiment-analysis argument specifies the task to be performed, initializing a pipeline pre-configured for sentiment classification.

The feedback list has two sample customer reviews - one positive ("Great product!") and one negative ("Terrible customer service."). These text inputs are passed to the pipeline, which analyzes each item in the list. Internally, the pipeline tokenizes the text, encodes it into a numerical format suitable for the model, processes it through the pre-trained sentiment classifier, and decodes the output.

For each review, the model produces two key outputs:

* **Label**: It shows the predicted sentiment category (e.g., "POSITIVE" or "NEGATIVE").
* **Score**: It is the confidence level of the prediction, ranging from 0 to 1.

The results are iterated through and printed, displaying the sentiment label and the corresponding confidence score for each input. For instance, the output might show:

`yaml

Sentiment: POSITIVE, Confidence: 0.99

Sentiment: NEGATIVE, Confidence: 0.85

`

This script exemplifies the practicality of NLP pipelines in real-world scenarios. Brands can deploy similar systems to watch and analyze customer sentiment on a scale, enabling them to act swiftly and strategically based on the insights derived from textual data. By automating this process, companies reduce manual effort, improve response times, and enhance customer satisfaction, aligning business strategies with public sentiment in real-time.

Integrating multiple models into a single pipeline

Integrating multiple models into a single pipeline is an advanced method to handle complex NLP challenges. By combining different models' abilities, these pipelines support comprehensive analysis and decision-making, greatly enhancing their scope and usefulness. In fields like customer service, healthcare, and financial analysis, integrated pipelines simplify workflows by automating tasks that would otherwise require manual effort or multiple separate systems. This section explores the practical uses of such integrations, showing how effective they can be in solving real-world problems quickly and efficiently.

Customer service automation

Integrated pipelines in customer service settings combine sentiment analysis, intent recognition, and entity extraction to deliver a holistic understanding of customer interactions. For instance, sentiment analysis finds the emotional tone of a customer query, while intent recognition finds the purpose behind the message (e.g., a refund request or product inquiry). Entity extraction then finds specific elements like order numbers, product names, or dates. Together, these tasks enable automated routing to the proper support agent or system, improving response times and customer satisfaction. Such systems are widely employed in AI-driven chatbots and virtual assistants [9].

Healthcare data processing

In the healthcare sector, integrated pipelines help process unstructured clinical text by combining medical entity recognition with summarization techniques. For example, an NLP pipeline might extract relevant entities such as drug names, symptoms, or medical conditions from a patient's clinical notes, followed by summarizing these findings into actionable insights for healthcare professionals. This approach enhances the efficiency of clinical workflows, allowing practitioners to focus on patient care instead of sifting through extensive medical records [10].

*Figure 6.2* shows the integration of multiple NLP models within a unified Hugging Face pipeline, enabling simultaneous sentiment analysis, entity recognition, and intent detection for enriched text understanding.

A diagram of a medical procedure

AI-generated content may be incorrect.

**Figure 6.2**: An integrated NLP pipeline performing multi-task inference using sentiment analysis, named entity recognition, and intent classification. This modular design enables comprehensive analysis of a single input and streamlines downstream decision-making in I

To illustrate the integration of multiple models, the following code combines sentiment analysis and NER tasks into a single pipeline. This example proves how text inputs can be simultaneously processed for emotional tone and entity identification:

python

from transformers import pipeline

# Load pipelines  
sentiment\_pipeline = pipeline("sentiment-analysis")  
ner\_pipeline = pipeline("ner")

# Example text  
text = "Zinnia Health provides excellent AI-driven care solutions."

# Process text with both pipelines  
sentiment = sentiment\_pipeline(text)  
entities = ner\_pipeline(text)

print("Sentiment:", sentiment)  
print("Entities:", entities)

This Python script highlights the integration of two pre-trained models within a unified pipeline to perform sentiment analysis and named entity recognition. The pipeline function from Hugging Face is used to load the respective models with pre-configured settings, such as:

* **Loading pipelines**: The script begins by initializing two separate pipelines: one for sentiment analysis (pipeline("sentiment-analysis")) and another for NER (pipeline("ner")). These pipelines encapsulate the complexity of model loading, tokenization, and inference, allowing for straightforward implementation.
* **Text input**: The example text, "Zinnia Health provides excellent AI-driven care solutions.", is processed through both pipelines to analyze its sentiment and extract entities. This input proves a typical real-world scenario where a single piece of text requires multi-layered analysis.
* **Processing**: The *sentiment analysis pipeline* determines the emotional tone of the text, outputting a label (e.g., "POSITIVE" or "NEGATIVE") and a confidence score indicating the model's certainty.

The NER pipeline finds entities within the text, such as "Zinnia Health" (an organization) and "AI-driven care solutions" (a concept or service). Each identified entity is paired with its category and positional indices in the input text.

* **Output interpretation**: The results are printed, displaying both the sentiment label and identified entities. For instance:

`css

Sentiment: [{'label': 'POSITIVE', 'score': 0.97}]

Entities: [{'entity': 'B-ORG', 'score': 0.95, 'index': 1, 'word': 'Zinnia Health'}]

`

This integration shows how pipelines can handle multiple NLP tasks at the same time, which means fewer processing steps are needed. Using pre-trained models speeds up deployment and cuts down on resource use, making them an ideal solution for applications that need complex analysis. This method is especially helpful when efficiency and accuracy are particularly important, such as in automated reporting, customer feedback analysis, and clinical data management.

Managing and scaling pipelines for production use

The transition from development to production is a critical phase in deploying NLP pipelines. Effective management and scaling strategies are essential to ensure that pipelines perform reliably under diverse conditions, manage high workloads, and maintain consistency across various deployment environments. This section examines deployment strategies, scaling techniques, and monitoring tools, providing practical guidance on managing NLP pipelines for real-world applications. By understanding and implementing these approaches, practitioners can optimize pipeline performance while minimizing operational overhead [11]; [12].

Deployment strategies

Deploying NLP pipelines in production requires robust and scalable solutions to ensure consistent performance and reliability. Strategies such as containerization, load balancing, and monitoring are essential for achieving these goals.

*Figure 6.3* illustrates a scalable, production-grade architecture for deploying NLP pipelines using containerization, load balancing, and monitoring tools to ensure high availability and operational efficiency:

A diagram of a software development

AI-generated content may be incorrect.

**Figure 6.3**: Architecture of a scalable NLP pipeline deployment. Client requests are routed through a load balancer to containerized services running Hugging Face pipelines, which are monitored in real-time using observability tools. This setup ensures consistent performance

Containerization

Containerization, using tools like Docker, allows pipelines to run in isolated, reproducible environments. By encapsulating code, dependencies, and configurations, containers cut discrepancies across development, testing, and production setups. For instance, deploying a sentiment analysis pipeline within a Docker container ensures that the same environment is kept across local machines and cloud servers. Docker Compose or Kubernetes can further streamline orchestration and scaling across multiple containers [13].

Load balancing

Load balancing distributes incoming requests across multiple servers, ensuring the best resource utilization and preventing bottlenecks. For example, an NLP pipeline managing real-time sentiment analysis on a high-traffic e-commerce site can use load balancers to spread tasks across multiple instances of the same pipeline. Tools like NGINX or **Amazon Web Services** (**AWS**) Elastic Load Balancer efficiently manage task distribution, keeping low latency and high availability [14].

Monitoring

Monitoring tools like Prometheus and Grafana offer real-time insights into pipeline performance. Metrics such as response time, **central processing unit** (**CPU**) usage, and memory consumption help find potential bottlenecks or failures. For instance, tracking latency trends in a named entity recognition pipeline might reveal periods of high demand, prompting adjustments in server allocation or resource scaling [15].

Scaling example

Scaling an NLP pipeline requires thoughtful design to ensure it can manage increasing workloads efficiently. The following code demonstrates the deployment of a simple sentiment analysis pipeline using Flask, which is suitable for scaling with container orchestration platforms such as Docker or Kubernetes:

`python

from flask import Flask, request, jsonify  
from transformers import pipeline

# Initialize Flask application and NLP pipeline

app = Flask(\_\_name\_\_)  
nlp\_pipeline = pipeline("sentiment-analysis")

@app.route('/analyze', methods=['POST'])  
def analyze():  
 data = request.get\_json()  
 text = data['text']  
 result = nlp\_pipeline(text)  
 return jsonify(result)

if \_\_name\_\_ == "\_\_main\_\_":  
 app.run(host="0.0.0.0", port=5000)

`

This script sets up a scalable NLP service using Flask as the web framework:

* **Pipeline initialization**: The pipeline function from Hugging Face loads a pre-trained sentiment analysis model. This lightweight setup enables rapid integration into a production environment.
* **Flask setup**: Flask provides a RESTful API interface for the pipeline. The /analyze endpoint accepts HTTP POST requests with **JavaScript Object Notation** (**JSON**) data holding the text to analyze.
* **Processing requests**: Incoming requests are parsed to extract the text field. The sentiment analysis pipeline processes this text, returning results such as the sentiment label (e.g., "POSITIVE") and confidence score.
* Deployment: The Flask application runs on host 0.0.0.0, allowing external access. Port 5000 is specified for easy integration with Docker containers or load balancers.

By deploying this service in a Docker container, practitioners can replicate the environment across multiple servers, enabling horizontal scaling for handling high traffic volumes.

To keep NLP systems resilient and responsive in production, pipelines should be built with scalable, containerized deployment and smart load balancing from the start.

*Figure 6.4* shows a production-grade NLP pipeline architecture that combines containerization (Docker), load balancing, and monitoring layers for scalable deployment. The system supports real-time sentiment analysis through a RESTful Flask API, with Kubernetes and observability tools like Prometheus and Grafana providing added support.

A diagram of a software pipeline

AI-generated content may be incorrect.

**Figure 6.4**: A production-grade NLP pipeline architecture

Conclusion

This chapter provided an in-depth exploration of pipelines within the Hugging Face Diffusers library, highlighting their pivotal role in orchestrating NLP workflows. From building and customizing pipelines to integrating multiple models for sophisticated tasks, we examined how pipelines streamline complex processes, enhance scalability, and ensure seamless production deployment. The practical examples and case studies underscore the versatility of pipelines, empowering readers to develop robust and efficient NLP solutions tailored to real-world challenges.

As we move into *Chapter 7, Schedulers in HF Diffusers*, the focus shifts to schedulers, essential tools for refining the training and inference stages of NLP models. Readers will learn how schedulers adjust parameters in real-time to enhance convergence, manage resources, and boost model performance. This understanding will lead to more efficient workflows and advanced techniques in NLP system development.

Exercises

Refer to the following section, which explores the practical exercises that deepen the understanding of pipeline management and scaling in production:

1. **Custom pipeline design**: Build a pipeline that combines sentiment analysis and text summarization. For instance, use Hugging Face's pre-trained models to analyze customer reviews, extract sentiment, and summarize key feedback points. Assess performance on a dataset like Yelp reviews, adjusting parameters for best outcomes [6].
2. **Multi-task pipeline**: Implement a pipeline integrating NER and sentiment analysis. This approach could process financial news articles, extracting entities like company names or stock symbols while assessing the article's sentiment to inform investment strategies [16].
3. **Scaling challenge**: Deploy the multi-task pipeline in a load-balanced Kubernetes setup. Launch multiple service instances, set up a load balancer to handle incoming traffic, and test scalability by simulating high-demand scenarios with tools like Apache JMeter [17].
4. **Pipeline performance comparison**: Compare execution times and accuracy between pre-built and custom pipelines. For example, evaluate the performance of Hugging Face's pre-built NER pipeline against a fine-tuned BERT model adapted for a specific dataset like CoNLL-2003. Analyze trade-offs in latency, memory usage, and prediction accuracy [18].



References

1. T. Wolf, L. Debut, V. Sanh, J. Chaumond, C. Delangue, A. Moi, P. Cistac, T. Rault, R. Louf, M. Funtowicz and J. Brew, "Transformers: State-of-the-Art Natural Language Processing," in Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations, 2020.
2. T. B. e. a. Brown, Language models are few-shot learners, NeurIPS, 2020.
3. C. D. Manning, P. Raghavan and H. Schutze, Introduction to Information Retrieval, Cambridge University Press, 2008.
4. J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, BERT: Pre-training of deep bidirectional transformers for language understanding, NAACL-HLT, 2019.
5. T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado and J. Dean, "Distributed Representations of Words and Phrases and Their Compositionality," Advances in Neural Information Processing Systems, vol. 26, pp. 3111, 3119, 2013.
6. C. Y. Liu and F. J. Och, "Automatic Evaluation of Machine Translation Quality Using Longest Common Subsequence and Skip-Bigram Statistics," in Proceedings of the 42nd Annual Meeting on Association for Computational Linguistics, 2019.
7. D. Jurafsky and J. H. Martin, Speech and Language Processing, Third ed., 2021.
8. B. Pang and L. Lee, "Opinion Mining and Sentiment Analysis," Foundations and Trends in Information Retrieval, vol. 2, pp. 1, 135, 2008.
9. S. Chopra, M. Auli and A. M. Rush, "Abstractive Sentence Summarization with Attentive Recurrent Neural Networks," in Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, 2016.
10. B. Shickel, P. J. Tighe, A. Bihorac, and P. Rashidi, "Deep EHR: A Survey of Recent Advances in Deep Learning Techniques for Electronic Health Record (EHR) Analysis," IEEE Journal of Biomedical and Health Informatics, vol. 22, pp. 1589, 1604, 2018.
11. I. Loshchilov and F. Hutter, "SGDR: Stochastic Gradient Descent with Warm Restarts," 2016.
12. [12] D. P. Kingma and J. L. Ba, "Adam: A Method for Stochastic Optimization," in Proceedings of the 3rd International Conference on Learning Representations (ICLR), 2015.
13. D. Merkel, "Docker: Lightweight Linux Containers for Consistent Development and Deployment," Linux Journal, vol. 2014, 2014.
14. D. Chandra, A. Gupta and S. Sharma, "Load Balancing Techniques for Web Applications," Journal of Network and Computer Applications, vol. 146, p. 102445, 2020.
15. K. Tzoumas, P. Boncz and A. Zeller, "Real-Time Monitoring of Large-Scale Systems with Prometheus," Journal of Systems and Software, vol. 134, p. 145. 158, 2017.
16. I. Chalkidis, M. Fergadiotis, P. Malakasiotis and I. Androutsopoulos, "LEGAL-BERT: Pretrained Transformers for Legal Text Mining," 2020.
17. A. Rodriguez, A. Tovar, and S. Riva, "Performance Testing with Apache JMeter," ITNOW, vol. 57, pp. 26, 27, 2015.
18. I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to Sequence Learning with Neural Networks," 2013.
19. M. Noyan, "Open-Source Text Generation & LLM Ecosystem at Hugging Face," 17 July 2023. [Online]. Available: https://huggingface.co/blog/os-llms.
20. A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, and I. Sutskever, "Language models are unsupervised multitask learners," 2019. [Online].
21. I. Goodfellow, Y. Bengio, and A. Courville, Deep Learning, MIT Press, 2016.
22. L. R. Rabiner, "A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition," in Proceedings of the IEEE, 1989.
23. L. N. Smith, "Cyclical Learning Rates for Training Neural Networks," in 2017 IEEE Winter Conference on Applications of Computer Vision (WACV), 2017.
24. A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," Advances in neural information processing systems, 2017.