NLP for Developers

Thanks to the recent advances in natural language processing (NLP), information can be extracted from large textual databases in a highly intuitive manner. But as a developer, it can be hard to wrap your head around how to not only build an NLP application prototype, but deploy it to production and make sure that it remains up to date.

This book is designed to help developers and data scientists understand the process of implementing an NLP system. It introduces foundational concepts in modern NLP in accessible language, and shows you the tools to design, build, and maintain an end-to-end application that's powered by NLP.

deepset

Automated question answering, intuitive natural language search interfaces, voice control, and much, much more: natural language processing (NLP) is here to make all of our lives easier. But while the tools to set up an NLP prototype for virtually any kind of use case are all out there, overseeing a project that implements an end-to-end system – from its inception to the final product that your users interact with – can seem like a daunting task for developers.

Our ebook "NLP for Product Managers" provided a friendly yet in-depth introduction to the topic of NLP in general, and to the task of managing a team looking to implement an NLP-based system in particular. We recommend checking it out if you want to get up to speed on concepts like language modeling or **Transformer models**.

This book, on the other hand, looks at the NLP implementation process from a practical, technical angle –focusing on the process of building and shipping the system itself.

If you're an ML engineer or a data scientist tasked with implementing an NLP-based system, you'll likely have lots of questions:

The **Transformer** is a neural network architecture that brought about a fundamental change in natural language processing. Transformer-based language models are much better (and faster!) than their predecessors at processing complex sentences. They have thus become the de facto standard for modern NLP applications.

- Do I need to understand all the intricacies of the Transformer model architecture?
- How do I build a prototype that can be scaled to a production-ready system?

- How can I compare different pipeline architectures and **hyperparameter settings** to find those that work best for me?
- How can I adapt an existing language model to my use case?
- How can I collaborate with back-end engineers on my team towards the final deployment of the NLP system?

If you have ever asked yourself any of these questions, then this book is for you.

Hyperparameters are the settings in a machine learning model that don't get determined during training, because they govern the training process itself. Examples are the model's learning rate, or the number of results that the model returns. They need to be set manually. In most cases, hyperparameters can have a significant impact on the model's performance, so it's important to try out different settings, preferably in a systematic manner.

As the company behind Haystack and deepset Cloud, we at deepset are determined to enable the adoption of NLP-based products wherever it is useful. We see it as one of our core tasks to educate and empower the different people in the applied NLP workflow. By giving developers the tools to set up their own systems, we hope to see more organizations benefit from the incredible advances that the field of NLP has undergone in the past five years.

In this book, we will go over the different processes that are essential to successfully setting up an applied NLP system in production. After a brief introduction to the topic of applied NLP, we will look at the entire deployment cycle from a high-level perspective, and finally dive into each aspect separately – from the pipeline design to the final deployment to production.

Contents

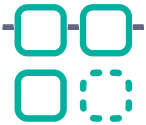
01 WHAT IS APPLIED NLP?

Unlike theoretical NLP, applied NLP focuses on providing developers with the tools they need to leverage pre-trained language models to benefit their organization in practice. People just learning about theoretical NLP are sometimes intimidated by the complexity of large, Transformer-based language models. These complicated fundamentals are less of a barrier in applied NLP: you will rarely be training a language model entirely from scratch.

Thanks to **model sharing**, you can go to centralized locations like the Hugging Face model hub, where tens of thousands of **pre-trained models** are freely available. You can also use an interface like OpenAI's API to access their language models without even leaving your IDE. Due to this ease of sharing, everyone can benefit from the huge leaps that research in NLP has made since the inception of the Transformer architecture for language modeling.

Pre-training and sharing models

Transformer models are big and powerful – and training them requires money and time. Thankfully, most NLP researchers engage in the resource-saving practice of model sharing: they upload their trained model's parameters to a publicly accessible platform like Hugging Face's model hub. From there, everyone can download the pre-trained language model and either use it directly, or subject it to further training passes to fine-tune it to a specific use case.



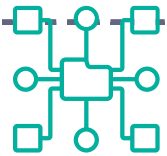
Frameworks with a focus on applied NLP therefore aim to assist their users with the following:

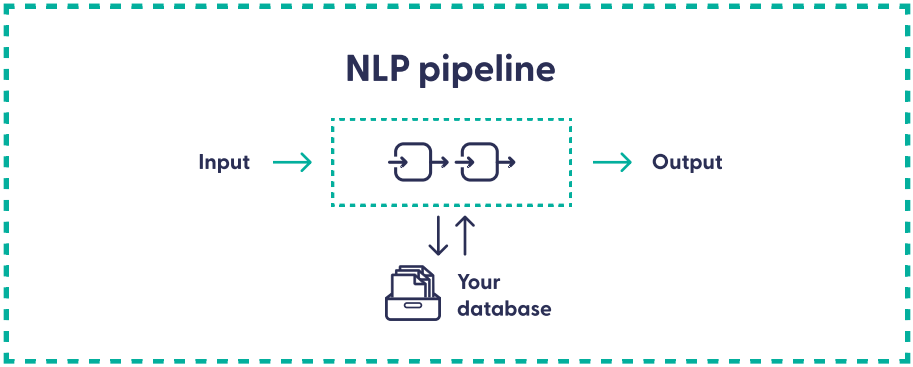
- Adapting large, pre-trained language models to individual use cases through tailored, curated data sets
- Providing the infrastructure for setting up a full-blown NLP system.

This infrastructure is usually provided in the form of pipelines. Pipelines are ready-made architectures with placeholders for language models. They handle the logic of inputting a natural-language query, transforming it in one or multiple steps, and outputting the result. The great benefit of pipelines is that they allow developers to compare different setups, build and test prototypes quickly, and later hand those prototypes over to a back-end engineer for final implementation.

Applied NLP frameworks

Many libraries and frameworks in NLP today focus on the implementation of NLP-powered products. Together with model sharing platforms, they're part of an ecosystem that makes it possible, even for non-NLP folks, to implement a production-ready system with Transformer-powered natural language abilities. Examples of such frameworks are spaCy, Haystack, and Hugging Face's Transformers.

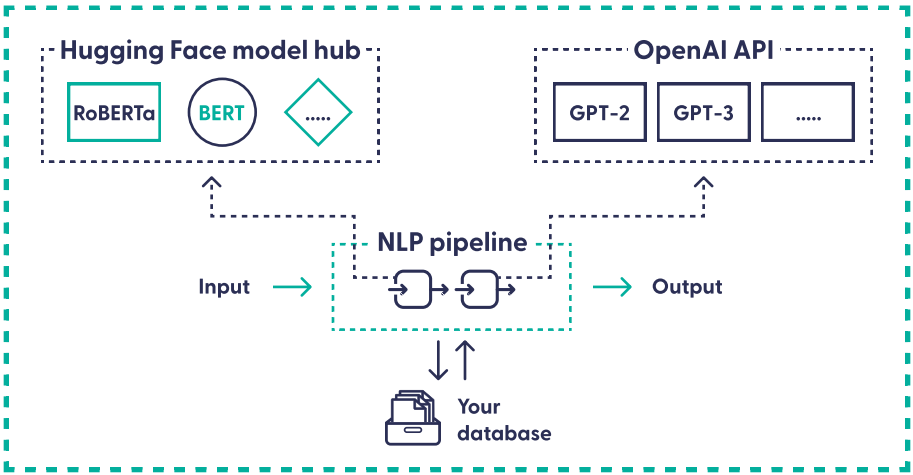




Pipelines also make it easy to connect to different kinds of databases, to set and tweak the models' hyperparameters, and to handle the types of high-power computing resources required when working with large, pre-trained Transformer models.

The concept of pipelines as containerized units is so useful that they are not only utilized for querying and inference. Besides query pipelines, most applied NLP systems implement pre-processing pipelines as well. These handle data transformation, any pre-processing steps like cleaning and splitting the text, and the storing of the transformed and pre-processed data in a database.

In addition to different pipeline designs for inference and pre-processing, applied NLP frameworks also offer interfaces to the various locations where pre-trained models are hosted. Thus, you can simply plug different models into your query pipeline and look at the results to find out what works best for you.



Not only can you use pre-trained language models out of the box, you can also use them as a basis for fine-tuning your own models. Fine-tuning refers to the task of feeding your model with additional data – either task- or domain-specific – to create a model that is customized to your own use case. That's why another mainstay of applied NLP is **data set management**.

Besides open-source libraries and frameworks, there are also managed solutions for implementing applied NLP projects. Such solutions provide a more approachable and user-friendly way to set up a working end-to-end system. The user interface makes it easy to implement and compare different system configurations, to collect feedback from end users early in the process, and to serve and deploy the final NLP application.

Data in NLP

As a sub-discipline of machine learning, and, more specifically, deep learning, modern NLP produces systems whose quality largely depends on the training data. That's why data-centric practices like creating high-quality data sets, regularly monitoring the data even after deployment, and performing qualitative error analyses are vital tools for the success of an applied NLP project.

To better understand the different ways in which applied NLP can benefit organizations in practice, let's have a look at the following three model use cases. Although, strictly speaking, these are hypothetical examples, they are inspired by real-world use cases that we've encountered over the years.

Use case 1: Building a QA system on top of a database of technical manuals

An aerospace manufacturer has amassed thousands of manuals for their products over the years. With that much textual information in private silos, it's hard for pilots in training to find the answers to their problems quickly (they can't just semantically scan the data set using internet search engines). Therefore, the company builds a question answering (QA) interface. Their setup uses a semantic retrieval model from the Hugging Face model hub, plus a QA model adapted to the technical domain.

Use case 2: Leveraging translation models to power multilingual chatbots

A chatbot manages help messages on an app with users in many different locations. It uses NLP to extract answers to the users' queries from a large collection of documents. Since the document corpus itself is monolingual, but users can ask questions in many languages, the company adds a translation layer on top of the chatbot pipeline.

Use case 3: Extracting information from business reports

A federal institution assesses financial risk factors for companies based on their business reports. They run each report through a pipeline for information extraction, which highlights the relevant sections using a model that's been fine-tuned to financial jargon. While low-confidence results still need to be checked manually, the system allows their officers to assess individual companies much faster.

Let's now have a look at the implementation cycle of an NLP system as a whole.

02 THE IMPLEMENTATION CYCLE IN APPLIED NLP

The NLP implementation process consists of two parts. In the prototyping phase, the developer sets up a working prototype pipeline and experiments with different configurations. In applied NLP, we usually opt for ***rapid prototyping***: a workflow in which a prototype system gets developed and deployed quickly, to make sure that we collect **user feedback** very early on in the process. This way, we can iterate through prototypes quickly, constantly improving and refining our system.

Once the cycle of prototyping, deployment and user feedback has produced a satisfactory system, the second phase, known as MLOps (machine learning operations), starts. In MLOps, the system is deployed and integrated into the final product. But it doesn't stop there. Rather, MLOps provides the framework for the regular monitoring, updating and improvement of a system in production. Because modern NLP is so driven by data, you need to make sure that your language models are regularly re-trained on textual data that captures your real-world use case.

Rather than following the sequential data science model of finalizing a system before deploying and sharing it with end users, applied NLP makes **user feedback** an essential element of the production process. In the old model, it could very well happen that after months of development, you would realize that your final product wasn't solving your users' actual problem. By involving an example group of end users early on, you can analyze their feedback to improve your system – for instance, by collecting and annotating new, more representative data to fine-tune your models.

Both the prototyping and the MLOps phase therefore heavily feature the testing and **evaluation of models**, both quantitatively and qualitatively. Only by periodically testing your system on real-world data and with real users can you make sure that it remains up to date.

Evaluating NLP models

Some NLP models are harder to evaluate than others. For example, if a sentiment classifier labels a text as positive while the correct label is negative, then we can safely say that it made a mistake. However, when it comes to generating or extracting text, it isn't always so easy to say what's wrong and what isn't. Your summarization model may output a summary that's vastly different from the "correct" text – but it can be just as good, or even better. That's why it's important to evaluate NLP models not only quantitatively, but also qualitatively – by having real-world users test them.

Let's take a bird's-eye view of the different phases of the implementation process, before we move on to look at each of the following topics in more detail.

Prototyping

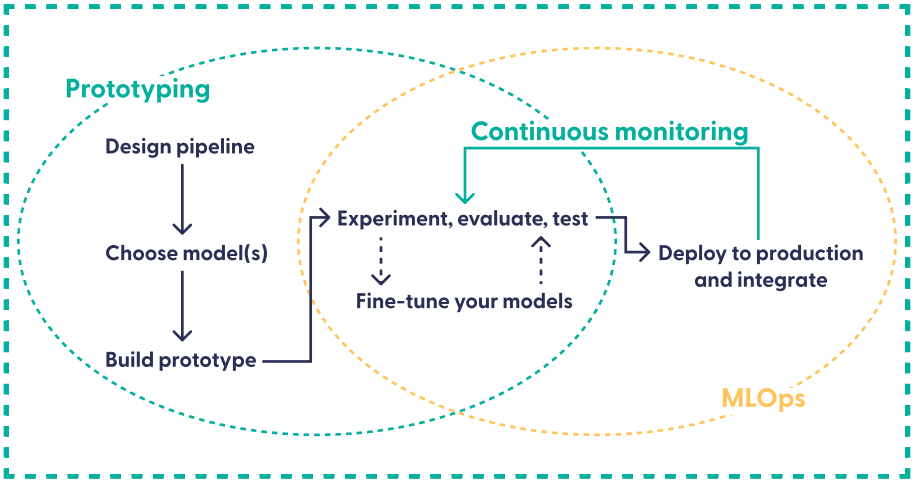
- **Design your pipeline.** Here, you think about what you want your system to achieve. Does it need to extract questions from a large collection of documents? Does it need to translate queries or outputs? Consider potential future use cases as well, so that the pipeline will be flexible and capable of scaling up with your project.
- **Choose the models.** The design decisions made in the previous step determine which kinds of language models are suitable for your system. In this phase, you'll want to visit the model hub and decide on some models that you want to test in your pipeline.

- **Build the prototype.** Set up the pipeline with your language models and connect it to your database. Now you already have a running system! If you're looking to evaluate different language models, then you'll likely build several pipeline prototypes at this stage.
- **Experiment, evaluate, and test.** This phase serves to find the best setup for your use case and the resources you have available. Is the inference fast enough? Accurate enough? What do your users think about it? You need to deploy a preliminary system, get it into the hands of your end users, and collect their feedback, in addition to evaluating the system on a representative evaluation data set. The more energy and resources you pour into this step, the better your system will become.
- **Fine-tune your models.** Collect real-world data for your use case, annotate it, and use it to fine-tune the models for higher accuracy. This step is followed by another iteration of experimentation, evaluation, and testing.

MLOps

- **Deployment to production and integration.** At this stage, you hand your prototype pipeline over to a back-end engineer, who will go on to deploy the system to production. It can then be integrated into the final product.
- **Monitoring your system.** After implementation, MLOps takes care of monitoring the product, periodically checking the system's performance against updated evaluation data sets, as well as collecting feedback from real users. It's important to make sure that your system remains up to date and doesn't decay. If you notice a dip in your system's quality, you'll likely want to go back to collecting and annotating new data, which will help you fine-tune your language models anew.

So, in a way, this final phase consists of periodically repeating the steps outlined earlier – only now, you're performing them on a system that's already been deployed to production.



The above diagram illustrates why, contrary to most people's perceptions, building an NLP system has little to do with training a Transformer language model from scratch. Rather, the hard work in applied NLP is about making sure that you have all the parts you need: high-quality, annotated data, and a framework that will help you with setting up pipelines, connecting to the resources where pre-trained models are stored, and fine-tuning those models on your own data.

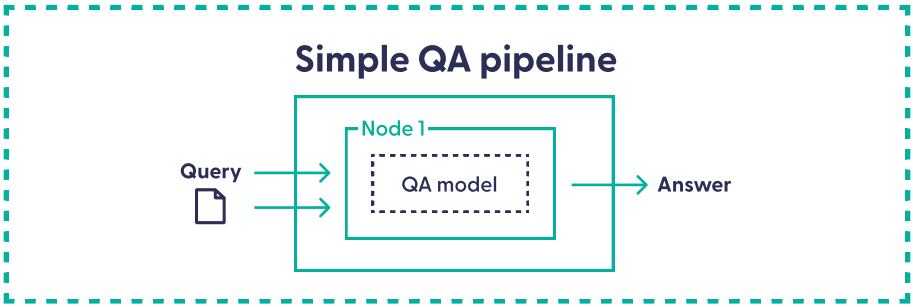
While the process in its entirety may seem a bit overwhelming, it helps to break it down into neatly defined steps. So in the next chapters, let's look at each of the various stages of the NLP implementation process in detail.

03 DESIGNING THE PIPELINE

As mentioned earlier, pipelines are a foundational paradigm in applied NLP. They allow developers to set up prototypes with complex language models quickly, and to share these prototypes with their back-end engineers. How, exactly, do pipelines accomplish that?

The pipeline paradigm

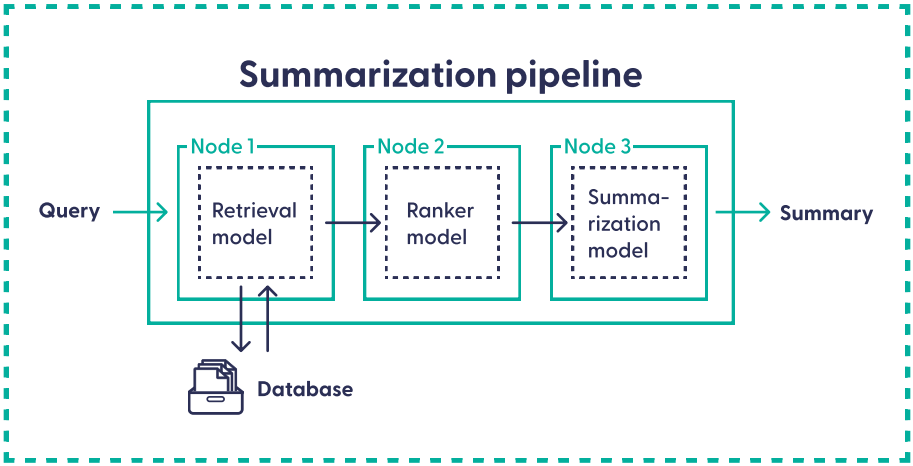
As we have established in the chapter 1, pipelines are system architectures that combine various NLP-powered nodes in a sequential order. In a query pipeline, each node holds a model that accomplishes a specific task. For an example of a very simple pipeline design, think of an extractive question answering (QA) pipeline. It consists of an input interface for queries, a node that contains the QA model, and an output interface that delivers the model's answers, which it extracts from a document.



What are the advantages of using pipelines?

Most NLP-based systems are much more complex, however, and that is where pipelines really shine. Think, for instance, of a summarization pipeline, which provides digests of several

documents retrieved from a large corpus in response to a natural language query.



1. They're easy to build

This system combines three nodes: one for retrieval, one for document ranking, and the summarization model proper. The pipeline takes care of routing the query to the first node, whose output then serves as input to the next node, and so forth. It also allows you to set all of the nodes' parameters – such as how many documents to return, the length of the summary, and whether or not to use a GPU – when you define the pipeline itself. By bundling all of these parameters in one place, the pipeline becomes much more manageable than if you had to juggle several Transformer models individually.

2. They're easy to reason about

That brings us to the second huge advantage: the pipeline paradigm allows you to reason about your system as a self-contained unit, just like the way a complex section of code can be abstracted away into a function or class and treated as a single item. When managing your summarization pipeline as

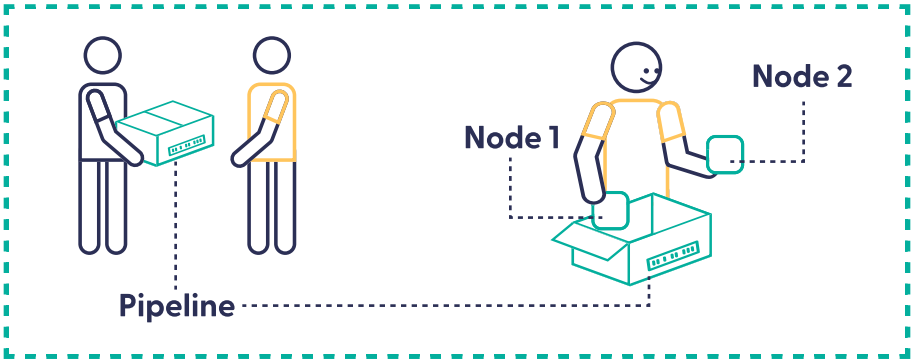
part of your final product, for example, you don't always need to remember that it uses a retriever, a ranker, and a summarizer, or which of those nodes connects to the database. All you need to know is that the pipeline receives a query, and, in response, outputs a summary based on your documents.

3. They're easy to share and document

Like a function or class, a pipeline is also easier to share with other people in your organization. Most applied NLP frameworks have a way of exporting pipelines in some basic file format. If you want to move your system to production, for example, all you need to do is export it to a file and hand it over to the respective engineers. This allows you to reproduce, reuse, and version-control any pipeline setup.

Black box versus individual nodes

But while it can be extremely useful to think about pipelines as self-contained units, it is often necessary to open the black box and look into the system components separately: during debugging, model evaluation, or testing. Pipelines make it possible to isolate each node and inspect its outputs, so that you remain in full control of your system's architecture.



Choosing the pipeline design

Your pipeline design should serve your use case. While many frameworks offer ready-made pipeline architectures, it's usually just as simple to set up your own system by connecting nodes within a default pipeline object. For example, you could use a custom design with two branches that lead to different retrieval models, or you could employ the architectural paradigm of a "wrapper" to implement a translation layer.

Apart from designing your system in the most efficient manner (by, for instance, only adding nodes that you strictly need), you should keep in mind that pipelines are unidirectional, and that the input structure of each node needs to match the output structure of the preceding one.

Once you have decided on a pipeline design, you can start thinking about which language models to populate your nodes with.

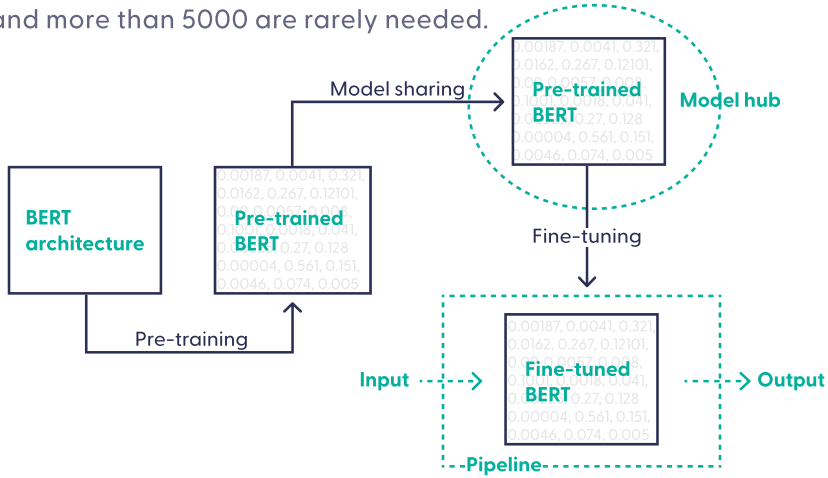
04 CHOOSING THE RIGHT MODELS

It is very rare that those deploying applied NLP systems will have to train their own models from scratch. Instead, working with Transformer-based language models in applied NLP is all about:

- Choosing the right pre-trained model for your use case
- **Fine-tuning** the model to the exact domain and task of your application, if needed
- Evaluating the model
- Monitoring and re-training the model periodically.

Fine-tuning

A pre-trained model is characterized by its weights – the parameters that have been set during training, and that determine the model's behavior during inference. When you fine-tune a model, the weights are initialized to the pre-trained values and you then adapt them further to your data by running additional training steps. Compared with pre-training, fine-tuning requires less (but usually annotated) data. Often, as few as 500 labeled data points suffice to adapt a model to the new target domain, and more than 5000 are rarely needed.



In this chapter, we'll look at the first step, which likely includes a visit to the model hub. The model hub is a bit like a department store. It helps to know in advance what you want, because it will make it easier to choose the models that you want to try out without wandering the aisles for hours. Those models can then be plugged into your pipeline, where you can evaluate them on your own data. At the end of this process, you'll be left with the language model that best fits your use case.

What do you need from your language model?

Of course, your project will dictate the main properties of your language model, such as the language itself and the specific task. For instance, if you're building a question answering model on top of texts in Turkish, then you'll want to look for a pre-trained Turkish QA model on the model hub. Or, if your system needs to translate between user inputs and the database, then you'll want to use a translation model.

But there are also other aspects to consider, some of which relate to the tradeoff between accuracy and speed. If you're looking for a very fast, lightweight model, then the accuracy of that model will likely be below the state of the art. The model

with the highest accuracy, on the other hand, will probably consume more resources to train and deploy, and will take a bit longer during inference. If you don't want to sacrifice one for the other, you could specifically look for **distilled language models** on the model hub, or even distill your own models later.

Model distillation describes the method of training a smaller model (the "student") to imitate the behavior of a larger model (the "teacher"). The result is a student model that is smaller than its teacher, but similar to it in terms of prediction quality. There are different model distillation techniques, some more complex (and more powerful) than others.

Note that there's no reason to limit yourself to one model. Thanks to fast prototyping in applied NLP, you'll be able to try and test many different models, before settling on the model to use in production.

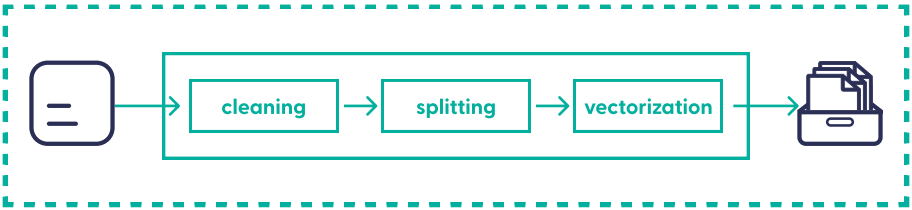
05 PROTOTYPING

After designing your pipeline architecture and choosing your model candidates, it's time to start prototyping. Fast prototyping, where you iterate through different system configurations by evaluating, testing and adapting pipelines quickly, is an essential element of the applied NLP implementation cycle.

A key advantage of using pipelines over setting up a system from scratch is that they allow for fast prototyping. All you need to do is instantiate the pipeline, add the models to it, and connect the pipeline to your underlying database. With the right applied NLP framework, the entire process won't require more than a few lines of code

Adding the documents

Text documents can come in many shapes and formats, and their initial form may not fit the language model you're going to work with. For instance, a question answering pipeline benefits from shorter documents. Here, too, it's best to use a pipeline in order to streamline the task of adding documents to your database. To distinguish it from the actual query pipeline, we refer to it as an ***indexing pipeline***.



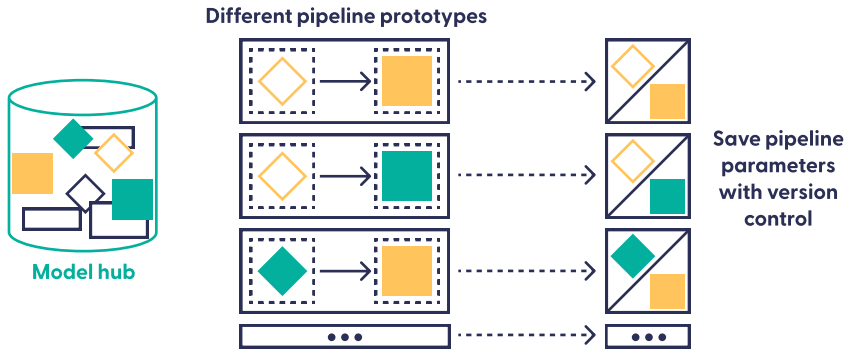
The indexing pipeline pre-processes your documents – for instance, by performing cleaning and splitting – before adding them to the database.

If you're working with a **Transformer-based retrieval model**, then you'll also need to run your documents through the model at indexing time. Depending on the model and the number of documents, that can take a while – but the good news is that the indexing pipeline only needs to run once, for a given set of pre-processing parameters.

Document retrieval describes the task of selecting documents from a large corpus in response to a query. When using a Transformer-based model for retrieval, the documents are represented as dense vectors (a sequence of computationally determined numbers) within the database. They can then be queried using questions in natural language.

Setting up the query pipeline

Set up the query pipeline by instantiating it together with the models that you want to try out. If you're working with several prototypes, make sure to keep them separate and identifiable. Ideally, do this using a version control system such as Git; alternatively, give them descriptive names and save their parameters in separate files.



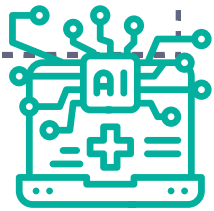
The pipeline is then connected to the indexed database. Now you can start querying your system – and, even better, let other people do the same.

06 DATA IN NLP I – TESTING AND EVALUATION

To understand the quality of your pipelines – and to be able to compare different systems – you'll need to evaluate and test them. For a quantitative analysis, a representative evaluation data set is run through the model to compute a number of performance metrics (see below). As a field that's closely related to **data-centric AI**, applied NLP places a high importance on the careful design of those data sets.

NLP systems aren't as straightforward to evaluate other machine learning models, because it's not always easy to define when language is "correct" (see infobox "Evaluating NLP systems"). That's why, to successfully evaluate an NLP system, it's necessary to include qualitative judgments in addition to the quantitative measurements.

The idea of improving deep learning models by concentrating on the data – rather than the model architecture – is at the core of a recent trend in machine learning dubbed "**data-centric AI**." Researchers in data-centric AI have come up with best practices regarding the annotation process, data set documentation, and "data in deployment" – that is, the ongoing need for monitoring and updating the training data for a system in production.



Quantitative evaluation

To begin quantitative evaluation you need to assemble a data set that accurately mirrors the real-world use cases your system is going to encounter in production. If you use data that is very differently distributed from the real data, or that only

captures certain problems and leaves out others, then the result of your evaluation won't be representative of your system's actual performance.

Annotation

How do you get your evaluation set of high-quality, representative data points? By annotating data from the real world (or at least a data set that is very similar). Annotation is a more complex task than it seems, due to the difficulty of putting real-world data into neatly separated categories, and the challenge of agreeing on annotation practices as a team. To make the task a bit easier, here are some tips:

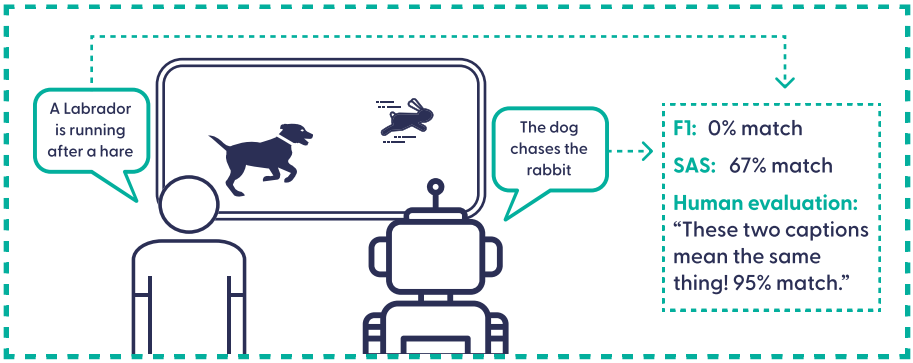
- Write ***annotation guidelines***. They're a crucial reference point for your team of annotators, and can later serve to document your data set and annotation process.
- Make sure to ***test the annotation guidelines*** on your data, to understand if you're really capturing all the use cases.
- If you're working with a team, train them and use measurements like an ***inter-annotator agreement*** to ensure that they're annotating the data in a consistent manner. It's also helpful to set up a ***communication channel between annotators***, where they can discuss certain edge cases, to further ensure the quality of the annotations.
- If you can, ***use an annotation platform***. This will make it easier to organize the data points and coordinate the work. Plus, a well-designed user interface will make it easier for your team to accomplish the task faster.

Once you have your evaluation data set in place, you can use it to compute a quantifiable metric of the quality of your system or systems.

Evaluating an NLP pipeline

When you evaluate an NLP pipeline, you can check the prediction quality either of the entire system or of individual nodes. The metrics you use then depend on the task that the nodes accomplish. For example, to evaluate a retrieval node, you'll be using a metric like recall, which measures the percentage of correctly retrieved documents.

When it comes to the evaluation of actual text-based nodes, you'll be looking at metrics like the F1 score, which measures the lexical overlap between the expected answer and the system's answer. However, the shortcomings of such metrics are apparent when we talk about semantic NLP systems, whose entire purpose is to abstract away from individual words and focus on the meaning of a text.



Two people could write vastly different summaries of the same text, and both could be equally appropriate. To capture that property, some frameworks have proposed Transformer-based metrics like the semantic answer similarity (SAS), which measure the semantic rather than the lexical congruence of two texts. However, as such methods are only slowly gaining wider traction, complementing your quantitative analysis with a qualitative one remains absolutely necessary.

Qualitative evaluation

Even if the data you use for your evaluation data set is relatively recent, nothing beats evaluating your system using real user feedback. Thanks to the easy prototyping offered by NLP pipelines, you'll be able to share one or more system prototypes with your audience early in the implementation cycle.

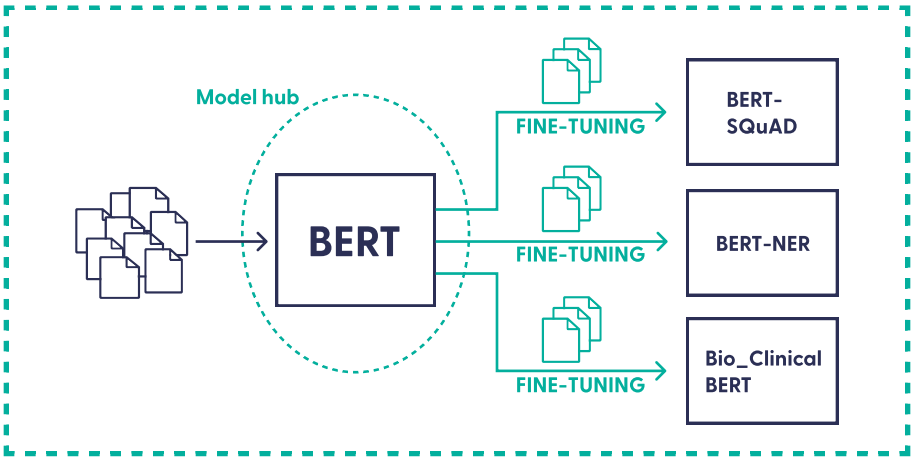
Share a simple browser-based prototype with a representative group of users and have them submit queries as they would to your final system. You can then ask them to evaluate the answers they received from the system. Aggregated, these judgments are valuable numbers that will help you decide which system to use in production, or which aspects of your system need improvement.

Testing your system with real-world users has a second benefit: it will alert you to discrepancies between your training or evaluation data set and the language used in the real world. Perhaps the demographic of your users has shifted, and they talk about different topics than they did when you collected your data. Perhaps they use words or abbreviations that your system has never seen before. You'll only know by investigating the data produced by your actual users. If it turns out that the language used by your users differs significantly from what your system has learned, then it's time to look into another data-driven technique: fine-tuning your language models.

07 DATA IN NLP II – FINE-TUNING

Fine-tuning may well be the most elegant technique in modern NLP. Instead of training a language model from scratch, you just download an existing one and train it further on your own data to accomplish a certain task, or to fit your specific domain better. While this step is often optional, almost every project benefits from fine-tuning, as a more customized language model fits your real-world application more snugly.

Large general-language Transformer models like BERT can be used as a basis for many different fine-tuning tasks, generating various fine-tuned models.



Fine-tuning versus domain adaptation

While the word “fine-tuning” is often used as an umbrella term, strictly speaking, there are two distinct ways of adapting a pre-trained language model: fine-tuning and domain adaptation. In ***domain adaptation***, you use data points with language that’s

specific to a particular area – finance or biology, for instance – to shift your general model towards an understanding of that domain-specific language. The result is still a general-purpose language model, which can be used as the starting point for task-specific fine-tuning.

Fine-tuning describes the practice of gearing a general language model toward a specific task like question answering, sentiment analysis, named entity recognition, or many others. The data sets used in fine-tuning differ in form from the general-purpose data sets. Both domain adaptation and fine-tuning get by with much less data than was required to generate the pre-trained model in the first place. That's because the pre-trained model already has accomplished the harder task of learning a general representation of the language, and only needs to apply that knowledge to a specific task.

Data, data, and more data

Creating a data set for fine-tuning requires as much care as assembling the evaluation data set. Understand that your model can only be as good as its training data – so if you put in the time and resources to create high-quality data sets, you will be rewarded with a powerful language model customized to your own application. The specific shape of your fine-tuning data depends on the task that you want to accomplish – for question answering, for example, it's common to use the **SQuAD** format.

The Stanford Question Answering Dataset (**SQuAD**) is a large collection of questions and corresponding answer passages often used for training question answering systems. SQuAD was originally designed for English, but versions of it now exist for many other languages, too.

08 MLOPS FOR NLP – DEPLOYMENT, INTEGRATION AND MONITORING

Once you've updated your data sets, fine-tuned your language models, and evaluated your prototypes in terms of both user feedback and evaluation metrics, it's finally time to deploy the system to production and release it to the world.

But, a bit like a garden that requires constant tending to, your ML models in production still need regular check-ins and care from their developer. So, rather than handing off your system to a back-end engineer and being done with it, MLOps requires that you as the model maintainer stay in the loop. Not only do you need to take care to deliver a deployable and maintainable system; because of the unique perishable nature of ML models, you also need to ensure that the downstream systems that use them remain up to date.

So, in this chapter, let's look at the steps involved in bringing your system to production – and maintaining it. Then we'll talk about how managed tools can assist you in streamlining a process that may, at times, seem quite overwhelming.

From prototypes to a system in production

While the underlying models are the same, the difference between running a prototype pipeline locally and deploying that same pipeline to production is like day and night. That's because a pipeline in production is just one component of a larger business application infrastructure. Besides the models, that infrastructure comprises servers, process management tools, and the data itself.

Such a complex setup has to satisfy many requirements. For example, it needs to:

- handle the scaling up and down of servers depending on system load
- handle parallel requests well
- handle authentication
- limit the rate of queries to prevent abuse of the system
- handle sensitive user data.

Applied NLP pipelines apply the principles of modularity and composability to NLP, in the same way composable stacks did this for other web apps a few years ago. Building your system out of modular components in a pipeline offers easily examined, granular control for use in installation, debugging, and analysis.

By packaging your composed system in a pipeline, you have created a transferable piece of software that handles everything from raw data to inference, and allows for smooth transition to a production environment.

Integrate your NLP system into your final product

In order for your pipeline to seamlessly integrate into the business application, it has to fit the overall API-driven application architecture. In the last decade using an HTTP RESTful API has become an industry standard for inter- and intra-application integration. This approach is widely used to instrument the common entry points to the application, where on a technical level the client requests and the application responses flow back and forth in a predictable manner.

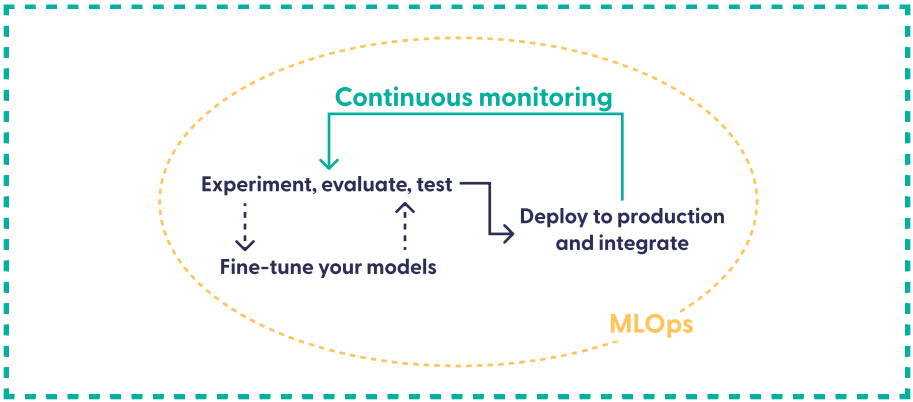
If your pipeline isn't wrapped in an API layer, it's going to be extremely hard for your peers in back-end and front-end app development to integrate it into the actual business application.

In many cases, the lack of a standardized, properly implemented API layer becomes a huge obstacle to actually using a pipeline in production.

Monitor your NLP application

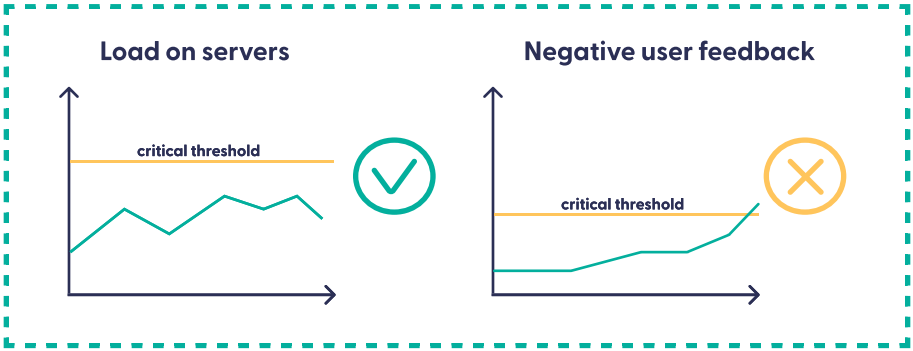
A central goal of MLOps is to account for the decay likely to occur in machine learning models. “Model decay” describes the phenomenon where ML models in production become outdated – and therefore less useful – over time. That’s why it’s so crucial to monitor your deployed system, and update it once it shows signs of decay.

As outlined in chapter 2, this part of the applied NLP process resembles a cycle: on a high level, you go back to evaluating and testing your models, and, optionally, to retraining and redeploying to production. This last step also likely involves annotation of additional data that matches the real world better than the data used for training the previous iteration of your model.



While some steps of the monitoring process – such as the user feedback and the retraining of models – involve manual work, others can and should be automated.

To be able to monitor your system, it's useful to devise metrics that can measure relevant aspects of your system's performance and quality. Those measurements can be calculated periodically, and their results can be integrated into some visualization, like a dashboard. In addition, you might want to define thresholds which, should your system reach them, will alert you to any critical states.



How about those aspects of the MLOps cycle that can't be fully automated? For example, both the selection of representative users and the evaluation of their feedback require some manual work, as does the collection of new training data and the retraining of models. Both tasks – feedback and retraining – are made considerably easier by managed platforms, which help your team organize the workflow, distribute the tasks, visualize workloads, and perform version control.

In addition to providing you with the tools to manage, monitor and maintain your system in production, managed tools also take a lot of work off the back-end engineer's shoulders – for example, by enabling automated scaling and authentication. And, of course, most managed solutions for serving ML systems in production include tools for monitoring those systems in a fully automated manner.

Building, implementing and maintaining an application that's powered by the latest NLP models might not be the easiest task for a developer – but thanks to model sharing, applied NLP frameworks, and hosted solutions with a powerful back end and an intuitive user interface, it is certainly doable. And it's worth it: providing a sleek, natural-language interface to unlock the information hidden in your data storage is a surefire way of building a satisfied user base.

Get started

At deepset, we think a lot about how to make one of the most exciting technological advances of our time available to a broader range of people. In our opinion, it's time to bring the impressive results of the last five years in NLP to every application and every service that uses natural language.

Haystack, our open-source applied NLP framework, lets you build your own natural language interfaces for your data. Have a look at the GitHub repository or the Haystack documentation page.

Check out our developer platform deepset Cloud for a fully managed solution that assists product teams in building their own NLP applications. deepset Cloud offers all the tools needed for fast prototyping, deployment, and collecting user feedback – all while implementing best practices from the MLOps cycle.

Finally, we'd be happy to welcome you in our online community on Discord, where you can chat to other developers building NLP products – or directly to our team.

09 LEARN MORE

deepset website

NLP for Product Managers ebook

When and How to Train Your Own Language Model

How to Evaluate a Question Answering System

Understanding Semantic Search

deepset Cloud documentation

Haystack website

Haystack annotation tool

Hugging Face model hub