NLP for Product Managers

Recently, natural language processing has become the focus of interest across many industries. Still, most practitioners find it challenging to understand the key concepts behind this powerful technology. In this ebook, we want to take a deep dive into how to apply modern NLP to your own use cases. We also show you how to use pre-built language models when developing an NLP application.

haystack
by deepset

You might have noticed some buzz lately around the acronym NLP. Short for natural language processing, it describes the understanding and production of human language by computers. Over the last five years, the field of NLP has undergone a massive transformation. Today, practically every business that handles textual data, whether internally or as part of its product, can benefit from the huge technical advances and open source culture in NLP.

As the company behind Haystack and deepset Cloud – whose aim is to make modular NLP accessible to any company working with textual data – we thought we'd share our experiences with using NLP in a business setting. This ebook is for you if you're looking to incorporate NLP technology into your own business applications and want to solidify your foundational background knowledge in order to discuss the topic efficiently with your team.

Perhaps you have a large collection of documents and want to make their content accessible at a glance – either company-internal, or to your users? Perhaps your customer-facing chatbot could benefit from a larger knowledge base and more flexible answering capabilities? We will address these and other use cases.

How this ebook is structured

This guide is divided into four chapters. In chapter 1, we provide an **overview of the current state of NLP**, before moving on to **specific industry applications** in chapter 2. Chapter 3 takes a look at the practical side of the **NLP implementation cycle**. Finally, in chapter 4, we'll leave you with a few **tips for a successful implementation**.

Who this ebook is for

Of course, this guide is not only for product managers, but for anyone looking for a comprehensive and timely overview of using NLP in business applications. So, without further ado, let's get started – by clarifying a few fundamental terms and their abbreviations.

Basic concepts

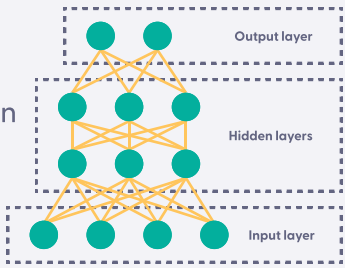
Machine learning

Machine learning (ML) describes a family of algorithms that learn patterns from domain-specific data to create a model of that domain. This learning process is called training. The trained model can then make reliable predictions about previously unseen data points.

Deep learning and neural networks

Deep learning (DL) is a subset of machine learning. DL forecasting algorithms all employ a type of artificial neural network (ANN or NN). Modern neural nets have millions or even billions of parameters and require lots of data during training. The word "deep" refers to the fact that these networks are typically realized as stacks of multiple "hidden" layers. During training, the data progresses through these layers sequentially.

An artificial neural network is a mathematical model for machine learning. Its structure is modeled after nerve cells in the brain, hence the name "neural." When exposed to training data, the network learns a representation of that data.



Deep neural networks – specifically, the Transformer architecture – excel at modeling unstructured input like natural language.

> Transformers are a group of neural networks that rely almost entirely on a technique known as attention. Attention models the underlying relationships between the words in a sequence, based on syntax and meaning. It's particularly good at capturing long-range dependencies: relationships between words that are far apart from each other in a given text passage.

Artificial intelligence

Whenever a machine tries to emulate intelligent behavior – be it through deep learning, machine learning more generally, or even rule-based systems – it can be categorized as artificial intelligence (AI).

Natural language processing (NLP) describes the art of teaching language to computers so that they can process, interpret, and produce human language. Like the term AI, NLP itself is not bound to a particular framework or technology, but rather a general description. Modern-day NLP uses ML and DL techniques heavily, but rule-based systems are still in use, too, for instance in chatbots. If you want to dive deeper into the topic of modern NLP, have a look at our curated list of NLP resources.

Contents

01 THE STATE OF NLP

In June 2017, a team of researchers at Google presented the Transformer: a novel neural network architecture for language modeling that was faster and seemed to understand natural language much better than its predecessors. From their inception, Transformer-based systems beat all the other models at sentiment analysis, question answering, and other tasks. The NLP community readily embraced the Transformer and never looked back.

A language model is the mathematical equivalent of the linguistic intuition that humans naturally develop as they learn a language. After seeing many, many sentences in a language, a language model learns its properties based on probability: it knows which words or word types are most likely to appear before or after which other words.

The term **sentiment** describes a speaker's attitude to a given topic. In sentiment analysis, the model learns to predict a label based on a short text. For instance, it can read a movie review and label it as positive, negative, or neutral.

As a result, modern-day NLP is not comparable to what existed thirty or even ten years ago (though, naturally, the current technologies build on decades of research).

Long gone are the days of purely hand-coded systems. Today, if you want to successfully implement an NLP system, you mainly need the right computational resources and plenty of labeled data.

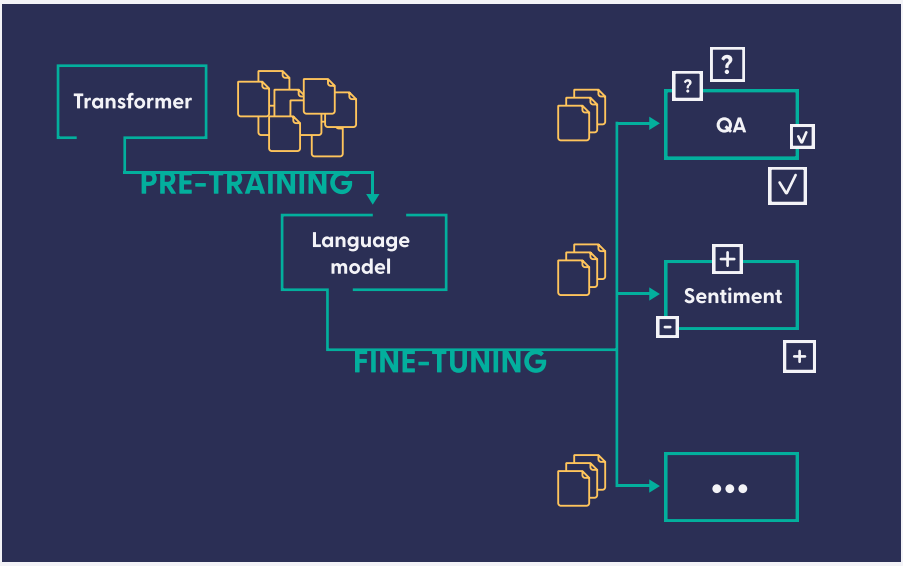
Obtaining, curating, and labeling data is often expensive and time-consuming, and not everyone has access to vast computing resources. To address these obstacles, modern NLP has two powerful tricks up its sleeve: model sharing and the pre-training/fine-tuning paradigm.

Model sharing

Training a general Transformer language model is quite an involved task: it takes several days and requires huge amounts of data and several heavy-duty processing units for parallel computing. Thankfully, once a model is trained, it can be saved for later use. The **pre-trained model** can then be shared on a central platform, where everyone can access it. Model sharing thus saves resources and democratizes access to large NLP models.

Fine-tuning

The pre-trained language model can serve as a basis for a number of "downstream tasks," like question answering. The technique used to do this is known as fine-tuning. It makes use of the **knowledge that is already stored** in the model's parameters, but then makes slight modifications to the model's setup, and trains it on new data from a specific domain. Note that, since the model has already learned a lot of a language's regularities in the pre-training phase, it can manage with much less data during fine-tuning.



The power of question answering systems

Question answering (QA) is the ability of a computer to answer questions asked in natural human language.

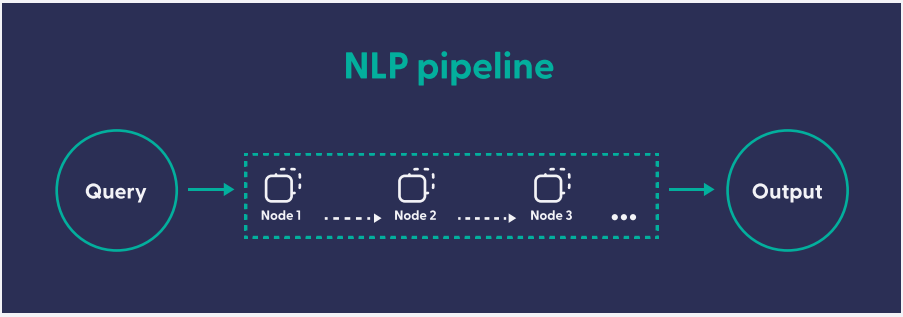
When implemented as an end-to-end system, it lets you make sense of large document repositories in a dynamic and intuitive way – all you need to do is type in a naturally worded question, such as "What was last year's most successful product?" Thus, QA can help organize knowledge bases, augment chatbots with natural-language understanding capabilities, implement a multilingual search system, and much more.

In question answering, the model receives a question and has to come up with an adequate answer. There's **extractive question answering** (the answer is extracted from a textual database) and **generative question answering** (the model generates the answer from scratch).

And it doesn't stop at text. Today's databases also store data of many other types, like images, video files, or tables. Transformer models can process all of these, and more. Ultimately, questions in natural language will likely become the main interface to any kind of data storage.

NLP pipelines

A modern NLP system is usually structured as a pipeline: a sequence of self-contained modules, each with its own task or function. This allows developers to get an overview of the system and reason about it at a high level, without needing to get bogged down in the implementation details of each module. It's also common to conceptualize such a pipeline as a "directed acyclic graph" (DAG), and the modules as nodes that are connected by (directed) edges.



A basic pipeline consists of input and output modules, which are connected by one or more consecutive nodes. Other than that, there are no limits to creativity. Nodes can be added, removed, or swapped. Developers can even design parallel branches within the pipeline, or use multiple output nodes. The next section showcases a few pipeline designs.

02 INDUSTRY APPLICATIONS FOR NLP

It is not enough to collect data in an internal database – in order to provide value, knowledge bases need to be **searchable**. As the success of the Google search engine has shown, fast and accurate extraction of information from textual databases is an almost universally sought-after skill.

Today, thanks to **model sharing**, everyone can use the same powerful language models as the likes of Google. Anybody can now apply these models' astonishing capabilities for understanding natural language to their own data. In this chapter, we will highlight a few real-world applications that incorporate state-of-the-art NLP technologies.

The power of question answering systems

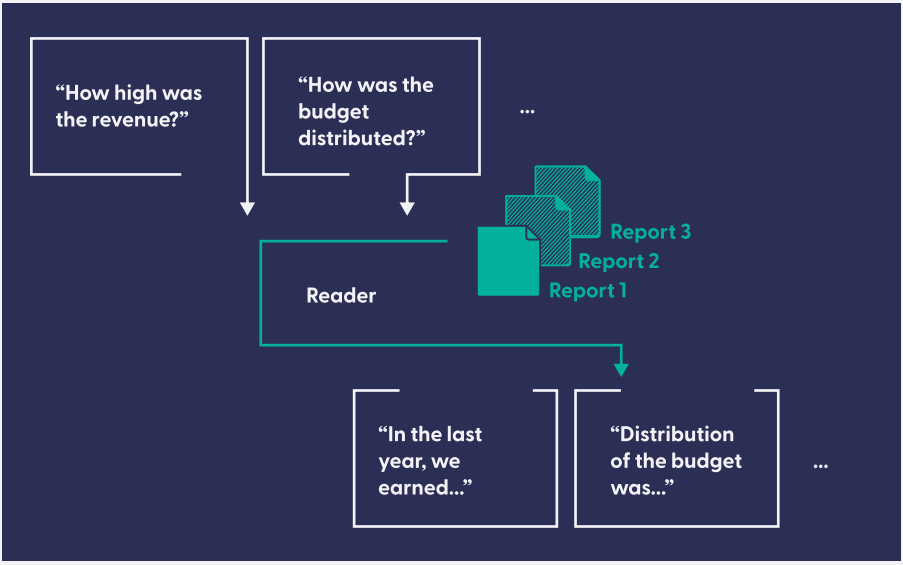
The systems described in this section all employ a technique known as semantic search. Semantic search allows users to ask questions naturally, as they would of a human. The system then provides answers on the basis of meaning rather than word matching. From a user perspective, the process of information retrieval thus becomes maximally intuitive.

In semantic search, documents or answers are selected on the basis of a congruence in meaning to the query. In contrast, a lexical search system can only find answers that use the same exact keywords as the query. To learn more about the evolution of semantic search systems, have a look at our blog post about semantic search and question answering.

Information extraction

Suppose you have a collection of annual business reports and want to know more about how the budget was distributed each year. With information extraction, that task is automated. Instead of sifting through the reports manually, you connect your database to a basic question answering pipeline that contains a "reader," which is a node with the language model.

> In a question answering pipeline, the reader node contains a Transformer language model, which accepts a question and a document. It then returns the answers extracted by the language model.



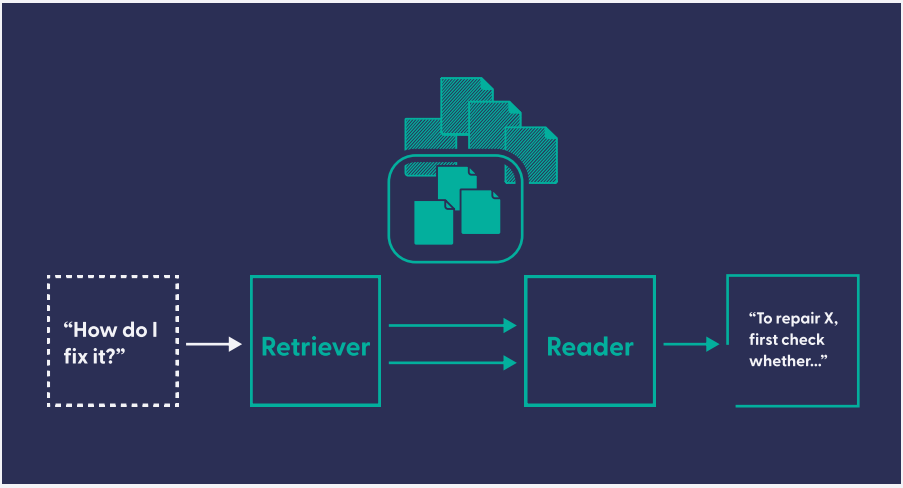
Once the pipeline is set up, you send a set of predefined queries through it and receive answers within seconds. It's a bit like creating a summary for each report, but in a more targeted way. If you want to know more, have a look at our blog post on automating information extraction.

Open-domain question answering

Most question answering systems have a document retrieval node in addition to the reader. In the previous example, the documents passed on to the reader were selected by the user.

In a retriever-reader pipeline, on the other hand, the retriever node chooses the right documents, which are then passed on to the reader's QA module. This setup is known as "open-domain question answering" (ODQA). ODQA systems are often used on top of a company-internal databases – for example, guideline manuals or internal reports.

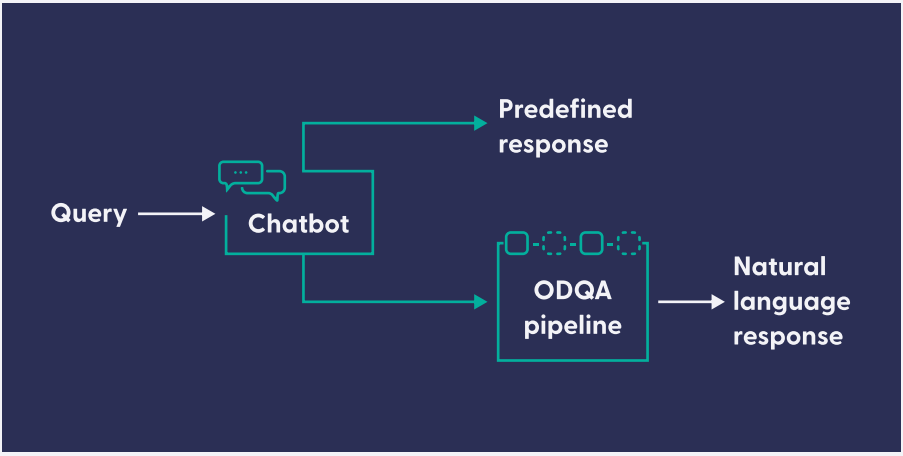
Open-domain question answering systems typically install a retriever node before the reader. The retriever can quickly match the query against the entire corpus, and return only the documents that it deems most capable of answering the question. These are then passed on to the reader.



Naturally, such a system can be open to the public, too. For an example, see our blog post on how French government agencies use question answering systems to provide citizens with better and more intuitive access to public datasets. Similarly, the technology may be used for curated search engines. As these examples show, ODQA systems make data accessible through a natural-language interface.

Chatbots

With chatbots, search can be implemented interactively. Chatbots are useful for handling customer requests at any time of the day. They save companies money and allow human operators to concentrate on the more complex cases. But when a chatbot cannot identify a user's **intent**, it sends an error message, which makes for a frustrating user experience. This can be avoided by augmenting a chatbot with a **semantic search** system that reroutes unidentifiable queries to the question answering system.



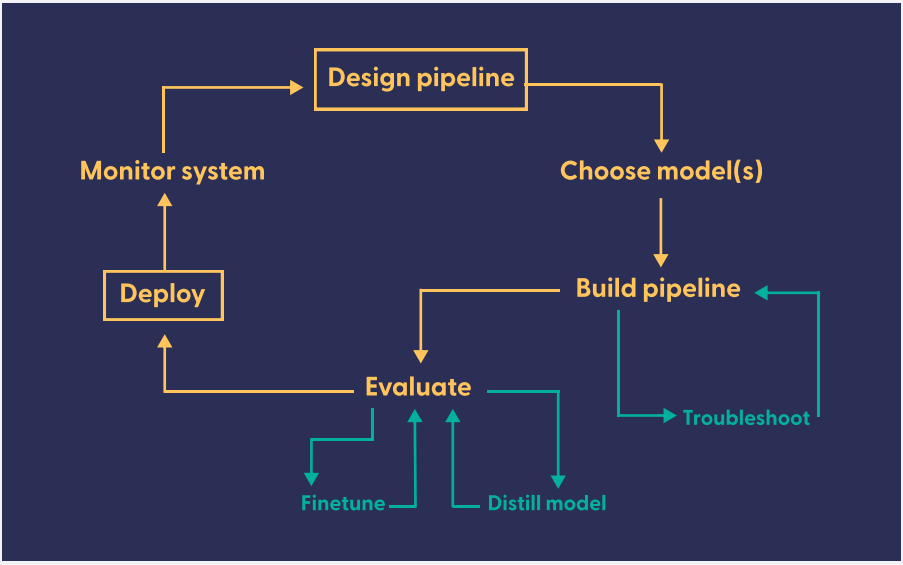
For an example on how to augment a Rasa chatbot with a question answering system, see our blog post on building smart conversational systems. Rather than sending unidentified or unanswerable requests to a human agent – who may or may not be available – this combined system can flexibly search for answers in a knowledge database, making it more capable of handling the twists and turns that the conversation might take.

Information extraction, open-domain question answering, and virtual assistance are probably the most common examples of using NLP in business applications. To get a clearer idea of what else can be achieved with NLP, a good first step is to check out available models on the Hugging Face model hub. There are models for translation, text summarization, sentiment analysis, and many, many more. As a rule of thumb, if there is a model, then you can probably build a system around it.

Hugging Face's model hub is the go-to platform for anyone looking to work with Transformers. It aggregates thousands of models that have been **pre-trained** to address a variety of use cases.

03 THE NLP IMPLEMENTATION CYCLE

Due to the modular nature of NLP systems and the massive language models involved, it is useful to break down the process of implementing any modern NLP project into a series of steps. We like to think of this process as a cycle, because it could, in theory, be improved continuously – for example, by fine-tuning a language model, annotating more data, or exploring more powerful computing resources. This is also why it's important to know when an NLP system is ready for deployment – and when it isn't. So in this chapter and the next, we look in detail at the various stages of the NLP implementation cycle.



Step 1: Design the pipeline

The first step of any modern NLP project is to design the pipeline. You've already learned about retrievers and readers in open-domain question answering systems. To a certain degree, your application will dictate what other nodes enter your pipeline. For example, a translation module is typically realized as a "wrapper" around your QA pipeline: one node to translate between the user's and the database's language, and one to translate the answers back into the user's language. For common use cases, various tools offer ready-made pipeline designs – you can find ours here. For custom applications, you can set up your own pipeline in a few simple steps.

Step 2: Choose your models

Once you have a clear idea of the nodes in your pipeline, it's time to go model-hunting. We recommend a trip to the Hugging Face model hub, where you will find **pre-trained Transformer models** of all sizes for many different use cases and languages. For a detailed guide, see our blog post on choosing the right reader model.

Alternatively, you could just go with the most popular model for your use case. For example, for an English-language ODQA prototype system you could combine the sparse BM25 retrieval method with the RoBERTa-base-SQuAD2 model. You will be able to change the models later.

Step 3: Build the pipeline

Once you have decided on your models, setting up a pipeline is straightforward and requires only a few lines of code. Connect the pipeline to your database, index your documents, and start querying.

Step 4 (optional): Troubleshoot the pipeline

If your pipeline throws any errors at this point – be it during initialization or at runtime – you'll need to troubleshoot it. Troubleshooting may involve visualizing your pipeline graph to see if all the modules are in the right places, running the modules in isolation, and checking the error messages for details on what went wrong.

Step 5: Evaluate your system

Once your system runs smoothly, it's time to **evaluate** its performance. Performance metrics compute an aggregate score of the quality of your system's predictions. They also offer a way of comparing different iterations of your NLP system. You can evaluate the entire pipeline, or look at each component individually. Parameter tweaking will help you get the most out of your system's performance. But if you want to dig deeper and really boost your models, have a look at the next two steps.

> Performance metrics tell you how well your system is doing on a given dataset. The choice of metrics should capture the end user's experience as accurately as possible.

Step 6 (optional): Fine-tune your system

This step is optional and actually consists of two steps: data **annotation** and the **fine-tuning** itself. If, during evaluation, you find that your system doesn't perform as well as expected, one option is to fine-tune one or more of your models. You may, for example, realize that your documents use very domain-specific language. If there is no specialized model for that domain on the model hub, you can annotate your own data, and use the labeled data to fine-tune the system. Once you have completed this step, you'll need to re-evaluate the system.

Step 7 (optional): Distill your model

It's not only the accuracy of the system that can be improved – sometimes, the pipeline may simply be too slow for your liking. In that case, you could distill your models. Model distillation makes use of the fact that Transformer models are typically very large, often containing millions of parameters. By **distilling** your large Transformer model, you can compress its knowledge into a smaller and faster model.
Again, once you have added your newly distilled model to the pipeline, you need to evaluate the system once more.

Step 8: Deploy your system to production

The big day has come: it's time to deploy your NLP pipeline to production. Deployment means making your model available to the end user. It is a crucial step in every software project that is meant for public consumption. Long before you actually move to production, you should be clear about how your system will be used.

Here are the most important questions that need addressing:

- **How will your system receive user requests?** If you are integrating NLP into a modern web application, usually the NLP functionality would be exposed through an API, for example, an HTTP REST API.
- **Where is your model going to run?** Will you use a public or private cloud, or host the model on-premises?
- **How many machines will you need, and what are the requirements** for those machines when it comes to CPU, RAM, disk space, and GPUs?
- **How will your system process incoming requests?** In batches or one by one?
- **What is the expected latency of your system** (the time between a user's request and the system's response)? What will the system do if the request takes longer?
- **How will the system handle a machine or multiple machines becoming unavailable** due to an outage? How will downstream systems handle the NLP system being temporarily unavailable?
- **Can your system scale** to an increase in queries by adding more machines? Do you need it to be able to auto-scale dynamically?

Once you have addressed these questions with your team, the engineers on your team can start deploying and managing your production NLP system.

Step 9: Monitor your system

Tests are a crucial component of every software project. Systems that leverage machine learning models cannot rely on traditional testing methods only. For example, you need to be aware of the issue of **model decay**. Machine learning models decay because the data they were trained on becomes outdated. That is why you should periodically benchmark your models and collect end user feedback on how well your system is doing. If you notice a dip in performance, old training data might be at fault: in machine learning projects, the quality of the data is just as important as the code itself.

Benchmarks are fixed tasks used to evaluate the performance of systems and compare different systems against each other. They help engineers understand how well their system is doing on a certain task in comparison to other solutions.

04 TIPS FOR A SUCCESSFUL NLP IMPLEMENTATION

Designing and implementing an NLP pipeline isn't an easy task. By following this guide, you're on a good path to a system that truly provides value to its users. To make sure that you really get there, here are three final tips to keep in mind throughout the entire process:

1: Think of it as an iterative process – and communicate that to your team

The iterative nature of modern NLP projects might be a new concept to some. People often expect a linear process with clear results. It is therefore important to communicate to your stakeholders that in NLP, **continuous monitoring and maintenance** is not only normal, but actually a feature that guarantees the best results.

2: Be ready to invest additional resources into improving your models

When you notice model decay or other unwanted behavior, your team will need to take a step back and invest some time and resources into re-evaluating and re-implementing the pipeline. What might seem daunting at first will pay off in the long run: a system that is tailored to your specific use case can solve your users' problems much better and will ultimately lead to higher customer satisfaction.

3: Encourage your team to experiment

Because every use case is different, it also makes sense to set aside some time in your development process to try out different pipeline designs and experiment with different models and model parameters. We've also seen teams realize that the existing metrics were not fully suitable to their application, and come up with new custom metrics. Implementing an NLP pipeline is as much a collaborative effort as it is a creative one.

05 GET STARTED

Come join Haystack's open source NLP community – either in our Slack channel, where you can talk to our developers directly and exchange ideas with other users, or on GitHub, where we host all our code.

If you have more questions on individual components of the Haystack framework, be sure to check out our comprehensive documentation.

NLP pipelines let you access the knowledge that's hidden in your textual database by querying it using natural language. But comparing different setups, evaluating the system through performance metrics and user feedback, and managing the computational resources can quickly become a demanding task that requires the time of one or more experts.

deepset Cloud handles the entire project cycle for you – learn more or have a look at the documentation.