How we built an AI-powered search engine (without being Google) - And how you can too!

Cole Thienes Follow Dec 4 · 4 min read

Coauthored by Jack Pertschuk, Check out our Github

In this article, I'll be recounting the difficulties of creating a generalizable, AI-powered search engine, and how we developed our solution, NBoost.

An exploding field

AI Information Retrieval (IR) is a booming area of research. Research in this field focuses on retrieving the most relevant search results based on the meaning of the search result, not just the keywords. Cutting-edge studies generally involve taking existing deep neural networks (such as Google's BERT), and training them to rank search results. However, problems are abundant (which I'll be talking about below). Building a robust, scalable semantic search engine is no small feat, so it's really no wonder Google makes so much money.

The hurdles

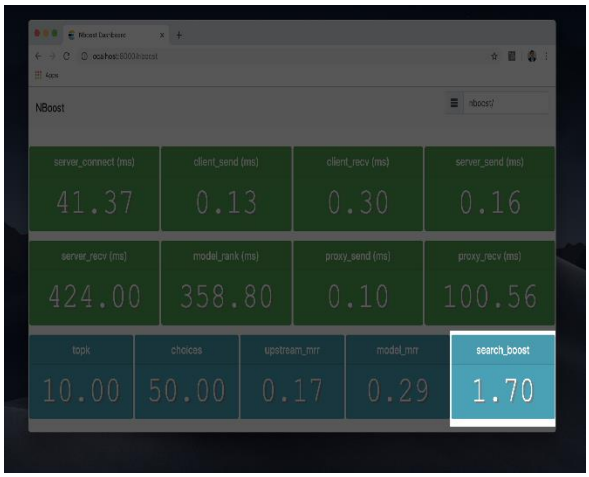
1. **It's hard to beat existing solutions.** Existing search engines such as Elasticsearch make use of text matching algorithms such as Best Match 25. These algorithms work by accounting for term frequency and other word patterns. They actually work surprisingly well. Therefore, they're hard to beat.
2. **Even if you do beat existing solutions, it's hard to generalize.** A frequently encountered problem in machine learning is training a model so much on a specific task so much that it cannot draw conclusions about a new task. This is called overfitting. Even if your model comes up with better search results for research articles than text-based search engines, that doesn't mean that it will work as well on cooking recipes.
3. **State-of-the-Art (SoTA) models are often slow and unscalable.** Even if you've got the perfect model that both beats text-matching algorithms, and works on many different domains, it may be too slow to use in production. Generally, SoTA models (such as BERT) have to be run on special hardware (a GPU) to scale to production workloads. This hardware is computationally expensive (and therefore fiscally). To build a search engine that ranks millions of documents, you can't just tell a large model to rank every search result one by one.

How we did it

As I mentioned previously, there's a massive amount of research going into studying the implications of machine learning in search engines. This means that researchers are competing to earn top spots on the Information Retrieval benchmarks such as MS MARCO. Some of these models more than double the quality of search results compared to existing search engines. We used these models, learned from them, and created our own (with top benchmarks). This is how **we beat existing solutions**.

We realized that none of this would be very useful if we couldn't scale it. That's why we built NBoost. When you deploy NBoost, you deploy a cutting edge model that sits in-between the user and the search engine, a sort of proxy. Every time the user queries the search engine, the model reranks the search results and returns the best ones to the user. We also built in support for deploying NBoost to the cloud and scaling with as many computers as needed, via the Kubernetes engine. This **combats the scalability problem**.

From the get-go, we wanted to create a platform that could be a foundation for **domain-specific** search engines. Therefore, we needed to make sure that NBoost was generalizable enough to be applied on different applications/datasets within a domain of knowledge. The NBoost default model was trained on millions of bing queries (MS MARCO). We found that our default model increased the relevancy of search results **by 80% over out-of-the-box Elasticsearch**. To test the generalizability of the model on a different corpus, we tested it on wikipedia queries (TREC CAR), a dataset that it had not seen before. It was a pleasant surprise when the frontend revealed that the default model boosted search results **by 70% on the different dataset.**



You can reproduce our results here.

You can too!

While we were building NBoost, we went out of our way to make our tools open source and easy to use. We made it available via pip, Docker, and Helm (Kubernetes). Our models are hosted on Google Buckets, and are installed automatically when you run NBoost them via `nboost --model_dir <model>`. You can find the list of available models on our benchmarks table.

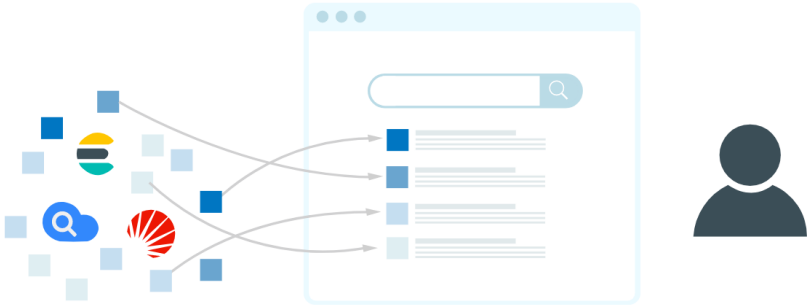
You can follow our tutorial to create your own AI search engine!

NBOOST

Highlights • Overview • Benchmarks • Install • Getting Started • Kubernetes • Documentation • Tutorials • Contributing • Release Notes • Blog

What is it

⚡**NBoost** is a scalable, search-engine-boosting platform for developing and deploying state-of-the-art models to improve the relevance of search results.



Nboost leverages finetuned models to produce domain-specific neural search engines. The platform can also improve other downstream tasks requiring ranked input, such as question answering. Contact us to request domain-specific models or leave feedback

Overview

The workflow of NBoost is relatively simple. Take the graphic above, and imagine that the server in this case is Elasticsearch.

Conventional

In a **conventional search request**, the user sends a query to *Elasticsearch* and gets back the results.

NBoost

In an **NBoost search request**, the user sends a query to the *model*. Then, the model asks for results from *Elasticsearch* and picks the best ones to return to the user.

Benchmarks

🔬 Note that we are evaluating models on differently constructed sets than they were trained on (MS Marco vs TREC-CAR), suggesting the generalizability of these models to many other real world search problems.

Fine-tuned Models	Dependency	Eval Set	Search Boost[1]	Speed
`bert-base-uncased-msmarco` (**default**)[2]	TensorFlow	bing queries	**+80%** (0.30 vs 0.17)	~300 ms/query
`bert-base-uncased-msmarco`		wiki search	**+71%** (0.29 vs 0.17)	~300 ms/query
`biobert-base-uncased-msmarco`		biomed	**+66%** (0.17 vs 0.10)	~300 ms/query
`bert-tiny-uncased` (*coming soon*)		-	-	~50ms/query

Instructions for reproducing here.

[1] MRR compared to BM25, the default for Elasticsearch. Reranking top 50.

[2] https://github.com/nyu-dl/dl4marco-bert

To use one of these fine-tuned models with nboost, run `nboost --model_dir bert-base-uncased-msmarco` for example, and it will download and cache automatically.

Using pre-trained language understanding models, you can boost search relevance metrics by nearly **2x** compared to just text search, with little to no extra configuration. While assessing performance, there is often a tradeoff between model accuracy and speed, so we benchmark both of these factors above. This leaderboard is a work in progress, and we intend on releasing more cutting edge models!

Install NBoost

There are two ways to get NBoost, either as a Docker image or as a PyPi package. **For cloud users, we highly recommend using NBoost via Docker**.

🚸 Depending on your model, you should install the respective Tensorflow or Pytorch dependencies. We package them below.

For installing NBoost, follow the table below.

Dependency	Docker	Pypi	Kubernetes
Tensorflow (*recommended*)	`koursaros/nboost:latest-tf`	`pip install nboost[tf]`	`helm install nboost/nboost --set image.tag=latest-tf`
Pytorch	`koursaros/nboost:latest-torch`	`pip install nboost[torch]`	`helm install nboost/nboost --set image.tag=latest-torch`
All	`koursaros/nboost:latest-all`	`pip install nboost[all]`	`helm install nboost/nboost --set image.tag=latest-all`
- (*for testing*)	`koursaros/nboost:latest-alpine`	`pip install nboost`	`helm install nboost/nboost --set image.tag=latest-alpine`

Any way you install it, if you end up reading the following message after `$ nboost --help` or `$ docker run koursaros/nboost --help`, then you are ready to go!

```
cole-mbp:test cole$ pip3 install nboost[tf]
Requirement already satisfied: nboost[tf] in /usr/local/lib/python3.7/site-packages (0.0.7)
```

```
ow==1.15; extra == "tf"->nboost[tf]) (0.15.6)
Requirement already satisfied: markdown>=2.6.8 in /usr/local/lib/python3.7/site-packages (from tensorboard<1.16.0,>=1.15.0->tensorflow
==1.15; extra == "tf"->nboost[tf]) (3.1.1)
Requirement already satisfied: h5py in /usr/local/lib/python3.7/site-packages (from keras-applications>=1.0.8->tensorflow==1.15; extra
 == "tf"->nboost[tf]) (2.10.0)
cole-mbp:test cole$ nboost --help
usage: nboost [-h] [--verbose] [--host HOST] [--port PORT] [--uhost UHOST]
              [--uport UPORT] [--lr LR] [--model_dir MODEL_DIR]
              [--data_dir DATA_DIR] [--max_seq_len MAX_SEQ_LEN]
              [--bufsize BUFSIZE] [--batch_size BATCH_SIZE]
              [--multiplier MULTIPLIER] [--workers WORKERS] [--codex CODEX]
              [--model MODEL]

NBoost (v0.0.7): is a scalable, search-api-boosting platform for
developing and deploying SOTA models to improve the relevance of search
results..

optional arguments:
  -h, --help            show this help message and exit
  --verbose             turn on detailed logging
  --host HOST           host of the proxy
  --port PORT           port of the proxy
  --uhost UHOST         host of the server
  --uport UPORT         port of the server
  --lr LR               learning rate of the model
  --model_dir MODEL_DIR
                        name or directory of the finetuned model
  --data_dir DATA_DIR   dir for model binary
  --max_seq_len MAX_SEQ_LEN
                        max combined token length
  --bufsize BUFSIZE     size of the http buffer
  --batch_size BATCH_SIZE
                        batch size for running through rerank model
  --multiplier MULTIPLIER
                        factor to increase results by
  --workers WORKERS     number of threads serving the proxy
  --codex CODEX         protocol class
  --model MODEL         model class
cole-mbp:test cole$ exit
exit
```

Getting Started

The Proxy

The Proxy is the core of NBoost. The proxy is essentially a wrapper to enable serving the model. It is able to understand incoming messages from specific search apis (i.e. Elasticsearch). When the proxy receives a message, it increases the amount of results the client is asking for so that the model can rerank a larger set and return the (hopefully) better results.

For instance, if a client asks for 10 results to do with the query "brown dogs" from Elasticsearch, then the proxy may increase the results request to 100 and filter down the best ten results for the client.

Setting up a Neural Proxy for Elasticsearch in 3 minutes

In this example we will set up a proxy to sit in between the client and Elasticsearch and boost the results!

Installing NBoost with tensorflow

If you want to run the example on a GPU, make sure you have Tensorflow 1.14-1.15 (with CUDA) to support the modeling functionality. However, if you want to just run it on a CPU, don't worry about it. For both cases, just run:

```
pip install nboost[tf]
```

Setting up an Elasticsearch Server

🔔 If you already have an Elasticsearch server, you can skip this step!

If you don't have Elasticsearch, not to worry! You can set up a local Elasticsearch cluster by using docker. First, get the ES image by running:

```
docker pull elasticsearch:7.4.2
```

Once you have the image, you can run an Elasticsearch server via:

```
docker run -d -p 9200:9200 -p 9300:9300 -e "discovery.type=single-node" elasticsearch:7.4.2
```

Deploying the proxy

Now we're ready to deploy our Neural Proxy! It is very simple to do this, simply run:

```
nboost --uport 9200
```

📢 The `--uhost` and `--uport` should be the same as the Elasticsearch server above! Uhost and uport are short for upstream-host and upstream-port (referring to the upstream server).

If you get this message: `Listening: <host>:<port>`, then we're good to go!

Indexing some data

NBoost has a handy indexing tool built in (`nboost-index`). For demonstration purposes, will be indexing a set of passages about traveling and hotels through NBoost. You can add the index to your Elasticsearch server by running:

`travel.csv` comes with NBoost

```
nboost-index --file travel.csv --name travel --delim ,
```

Now let's test it out! Hit the Elasticsearch with:

```
curl "http://localhost:8000/travel/_search?pretty&q=passage:vegas&size=2"
```

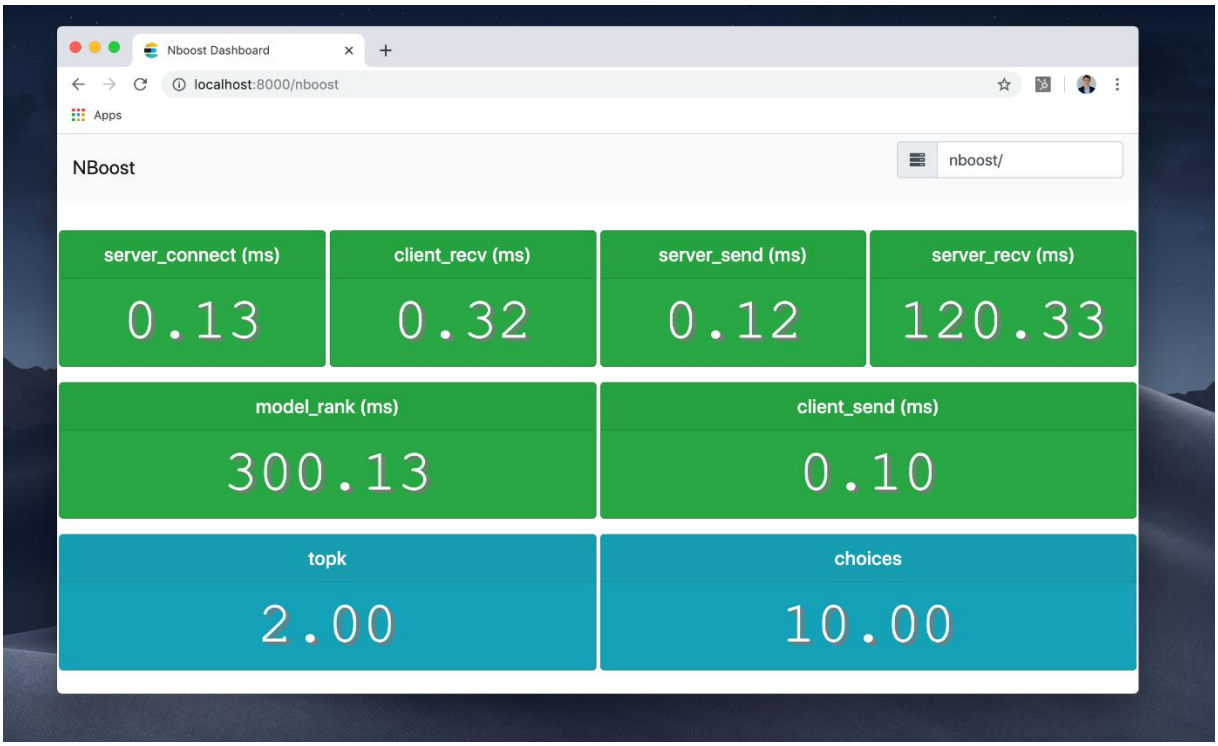
If the Elasticsearch result has the `_nboost` tag in it, congratulations it's working!

```
(nboost) cole-mbp:test cole$ nboost --uport 9200 &
[1] 84660
(nboost) cole-mbp:test cole$ I:bert-base-uncased-msmarco:[bas:dow: 36]:Using model cache from /Users/cole/Desktop/test/nboost/lib/pyth
on3.7/site-packages/nboost/.cache/bert-base-uncased-msmarco
WARNING:tensorflow:From /Users/cole/Desktop/test/nboost/lib/python3.7/site-packages/nboost/model/bert_model/__init__.py:26: The name t
f.logging.set_verbosity is deprecated. Please use tf.compat.v1.logging.set_verbosity instead.

WARNING:tensorflow:From /Users/cole/Desktop/test/nboost/lib/python3.7/site-packages/nboost/model/bert_model/__init__.py:26: The name t
f.logging.ERROR is deprecated. Please use tf.compat.v1.logging.ERROR instead.

C:BertModel:[pro:run:273]:Upstream host is 0.0.0.0:9200
I:BertModel:[ser:run: 47]:Starting 10 workers...
C:BertModel:[ser:run: 54]:Listening on 0.0.0.0:8000...
2019-12-03 23:30:01.801044: I tensorflow/core/platform/cpu_feature_guard.cc:142] Your CPU supports instructions that this TensorFlow b
inary was not compiled to use: AVX2 FMA
2019-12-03 23:30:01.812807: I tensorflow/compiler/xla/service/service.cc:168] XLA service 0x7fc10c989070 initialized for platform Host
 (this does not guarantee that XLA will be used). Devices:
2019-12-03 23:30:01.812823: I tensorflow/compiler/xla/service/service.cc:176]   StreamExecutor device (0): Host, Default Version
```

What just happened?

Let's check out the **NBoost frontend**. Go to your browser and visit localhost:8000/nboost.

If you don't have access to a browser, you can `curl http://localhost:8000/nboost/status` for the same information.



The frontend recorded everything that happened:

1. NBoost got a request for **2 search results**. *(0.32 ms)*
2. NBoost connected to the server. *(0.13 ms)*
3. NBoost sent a request for 10 search results to the server. *(0.12 ms)*
4. NBoost received **10 search results** from the server. *(120.33 ms)*
5. The model picked the best 2 search results. *(300 ms)*
6. NBoost returned the search results to the client. *(0.10 ms)*

Elastic made easy

To increase the number of parallel proxies, simply increase `--workers`. For a more robust deployment approach, you can distribute the proxy via Kubernetes (see below).

Kubernetes

Deploying NBoost via Kubernetes

We can easily deploy NBoost in a Kubernetes cluster using Helm.

Add the NBoost Helm Repo

First we need to register the repo with your Kubernetes cluster.

```
helm repo add nboost https://raw.githubusercontent.com/koursaros-ai/nboost/master/charts/
helm repo update
```

Deploy some NBoost replicas

Let's try deploying four replicas:

```
helm install --name nboost --set replicaCount=4 nboost/nboost
```

All possible `--set` (values.yaml) options are listed below:

Parameter	Description	Default
`replicaCount`	Number of replicas to deploy	`3`
`image.repository`	NBoost Image name	`koursaros/nboost`
`image.tag`	NBoost Image tag	`latest-tf`
`args.model_dir`	Name or directory of the finetuned model	`bert-base-uncased-msmarco`
`args.model`	Model Class	`BertModel`
`args.host`	Hostname of the proxy	`0.0.0.0`
`args.port`	Port for the proxy to listen on	`8000`
`args.uhost`	Hostname of the upstream search api server	`elasticsearch-master`
`args.uport`	Port of the upstream server	`9200`
`args.data_dir`	Directory to cache model binary	`nil`

Parameter	Description	Default
`args.max_seq_len`	Max combined token length	`64`
`args.bufsize`	Size of the http buffer in bytes	`2048`
`args.batch_size`	Batch size for running through rerank model	`4`
`args.multiplier`	Factor to increase results by	`5`
`args.workers`	Number of threads serving the proxy	`10`
`args.codex`	Codex Class	`ESCodex`
`service.type`	Kubernetes Service type	`LoadBalancer`
`resources`	resource needs and limits to apply to the pod	`{}`
`nodeSelector`	Node labels for pod assignment	`{}`
`affinity`	Affinity settings for pod assignment	`{}`
`tolerations`	Toleration labels for pod assignment	`[]`
`image.pullPolicy`	Image pull policy	`IfNotPresent`
`imagePullSecrets`	Docker registry secret names as an array	`[]` (does not add image pull secrets to deployed pods)
`nameOverride`	String to override Chart.name	`nil`
`fullnameOverride`	String to override Chart.fullname	`nil`
`serviceAccount.create`	Specifies whether a service account is created	`nil`
`serviceAccount.name`	The name of the service account to use. If not set and create is true, a name is generated using the fullname template	`nil`

Parameter	Description	Default
`serviceAccount.create`	Specifies whether a service account is created	`nil`
`podSecurityContext.fsGroup`	Group ID for the container	`nil`
`securityContext.runAsUser`	User ID for the container	`1001`
`ingress.enabled`	Enable ingress resource	`false`
`ingress.hostName`	Hostname to your installation	`nil`
`ingress.path`	Path within the url structure	`[]`
`ingress.tls`	enable ingress with tls	`[]`
`ingress.tls.secretName`	tls type secret to be used	`chart-example-tls`

Documentation

The official NBoost documentation is hosted on nboost.readthedocs.io. It is automatically built, updated and archived on every new release.

Contributing

Contributions are greatly appreciated! You can make corrections or updates and commit them to NBoost. Here are the steps:

1. Create a new branch, say `fix-nboost-typo-1`
2. Fix/improve the codebase
3. Commit the changes. Note the **commit message must follow the naming style**, say `Fix/model-bert: improve the readability and move sections`
4. Make a pull request. Note the **pull request must follow the naming style**. It can simply be one of your commit messages, just copy paste it, e.g. `Fix/model-bert: improve the readability and move sections`
5. Submit your pull request and wait for all checks passed (usually 10 minutes)
    - Coding style
    - Commit and PR styles check
    - All unit tests
6. Request reviews from one of the developers from our core team.
7. Merge!

More details can be found in the contributor guidelines.

Citing NBoost

If you use NBoost in an academic paper, we would love to be cited. Here are the two ways of citing NBoost:

```
\footnote{https://github.com/koursaros-ai/nboost}
```

2.

```
@misc{koursaros2019NBoost,
  title={NBoost: Neural Boosting Search Results},
  author={Thienes, Cole and Pertschuk, Jack},
  howpublished={\url{https://github.com/koursaros-ai/nboost}},
  year={2019}
}
```

License

If you have downloaded a copy of the NBoost binary or source code, please note that the NBoost binary and source code are both licensed under the Apache License, Version 2.0.