AI DEVCON 2018

Deep Learning in Personalization: Developing a retail customer use case

Mariano Phielipp
Senior Data Scientist

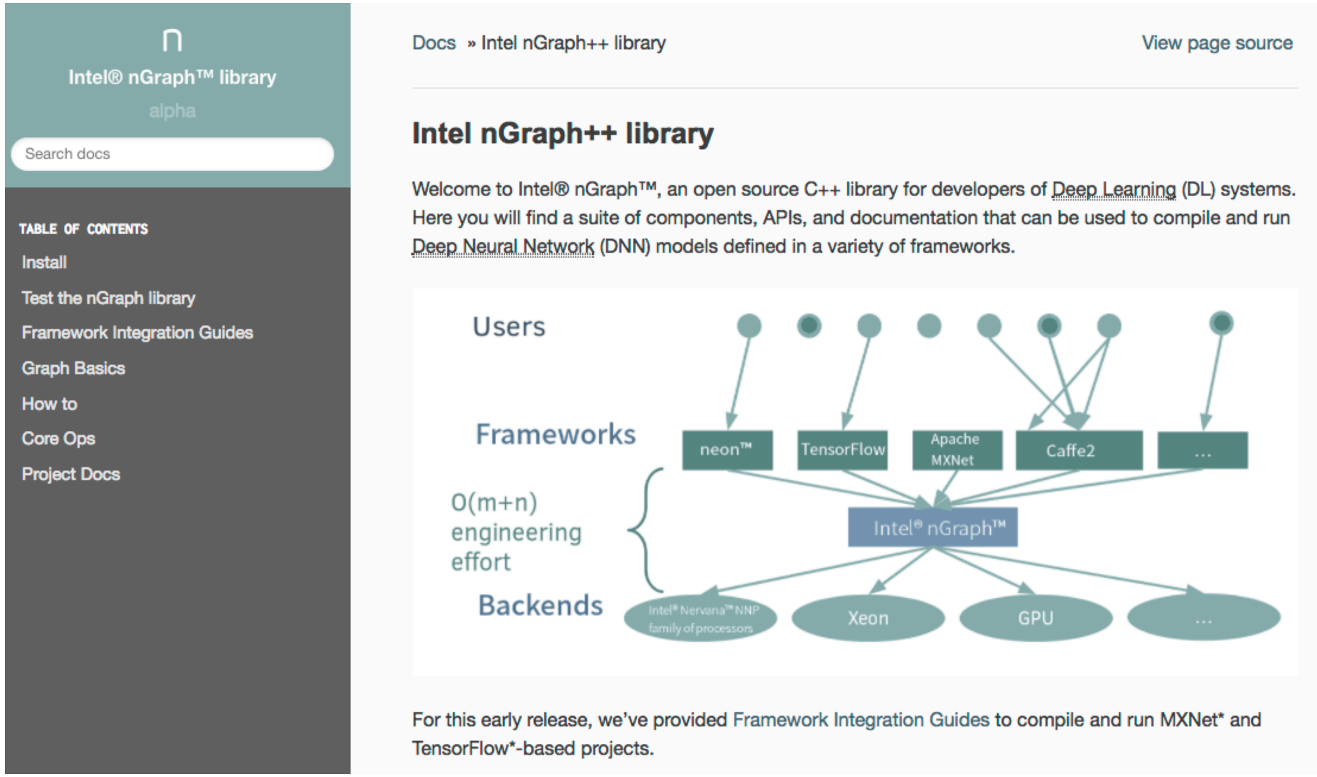
agenda

- Objective
- Recommendations Engines
- Wide and Deep Example in Intel® nGraph™
  - Use Case
  - Data
  - Model
  - Text and Image data
  - Evaluation Metric
  - Results
  - Call to Action

Objective

- This session will develop a Deep Learning Recommender System based on Wide and Deep model and Retail data.
- This exercise starts from scratch using Intel® nGraph™ . We will see a series of code snippets that describes core elements in order to build a Recommender System using Deep Learning.
- Will also present different techniques to incorporate other information when available into the model.
- Finally will show results indicating how DL techniques are performing better than traditional ML techniques in this particular use case.

INTEL ngraph


Intel® nGraph™ library

alpha

Search docs

TABLE OF CONTENTS

Docs » Intel nGraph++ library

View page source

Intel nGraph++ library

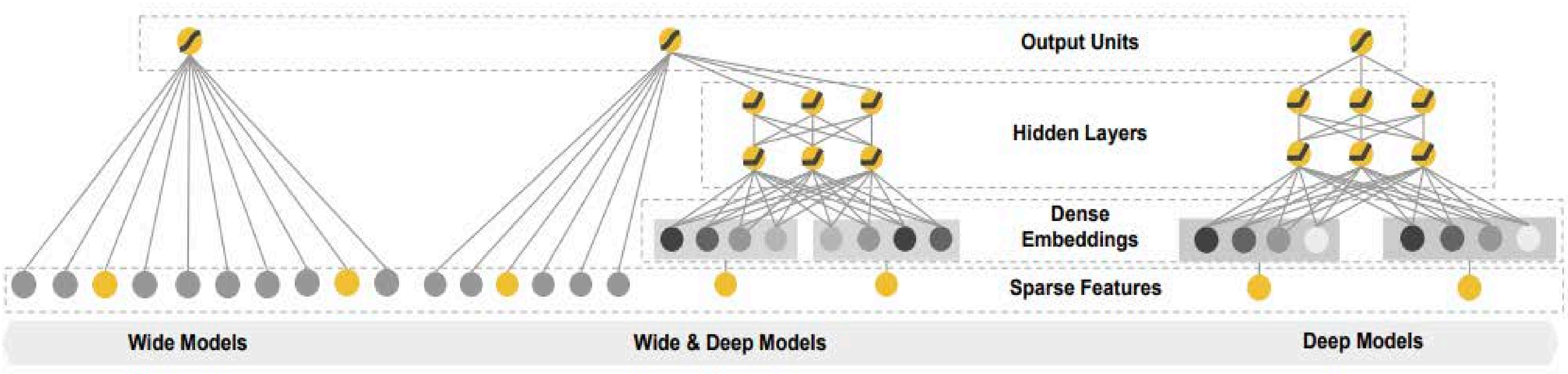
Welcome to Intel® nGraph™, an open source C++ library for developers of Deep Learning (DL) systems. Here you will find a suite of components, APIs, and documentation that can be used to compile and run Deep Neural Network (DNN) models defined in a variety of frameworks.

For this early release, we've provided Framework Integration Guides to compile and run MXNet* and TensorFlow*-based projects.

http://ngraph.nervanasys.com/docs/latest/install.html

Recommendations engines

- Classic Collaborative Filtering Approaches.
- User – Item : The similarity between users to score the items relevance.
- Item – Item : The similarity between items to score their relevance for the user.
- Non-negative factorization : A matrix factorization technique to solve the usual sparse nature of recommendations.



https://github.com/NervanaSystems/ngraph-python/tree/master/examples/wide_deep
https://www.tensorflow.org/tutorials/wide_and_deep

Use case

- Data
  - Customer Identifiers: Credit Card and Name.
  - Transaction Data : Date, Item, Amount.
  - Transaction data as an indirect signal for preference.
  - Normalization.
  - Unique mapping.

Data : pandas / SKlearn

- **#Defining the columns.**
  COLUMNS = ["Item Id", "Quantity Sold”, "Loyalty Customer Name"]

- **# Reading the data.**
  dataSales = pd.read_csv(”Sales.csv", usecols=COLUMNS, engine="python")

- **# Analyze your data and process it accordingly.**
  # Example of Filtering
  dataSales = dataSales[dataSales["Quantity Sold"] < 500.0]

- **# Normalize Columns Individually in Pandas data frame between [0,1]**

```
from sklearn import preprocessing
def normalize(df):
    min_max_scaler = preprocessing.MinMaxScaler()
    df_scaled = min_max_scaler.fit_transform(df.values)
    df_normalized = pd.DataFrame(df_scaled)
    return df_normalized
```

Data : pandas / SKlearn

- **# Unique Mapping**
- UID = "Loyalty Customer Name”
- ItemID = “Item id”

```
nCustomers = 0
customerDict = {}

nItems = 0
itemDict = {}

for index, row in dataSales.iterrows():

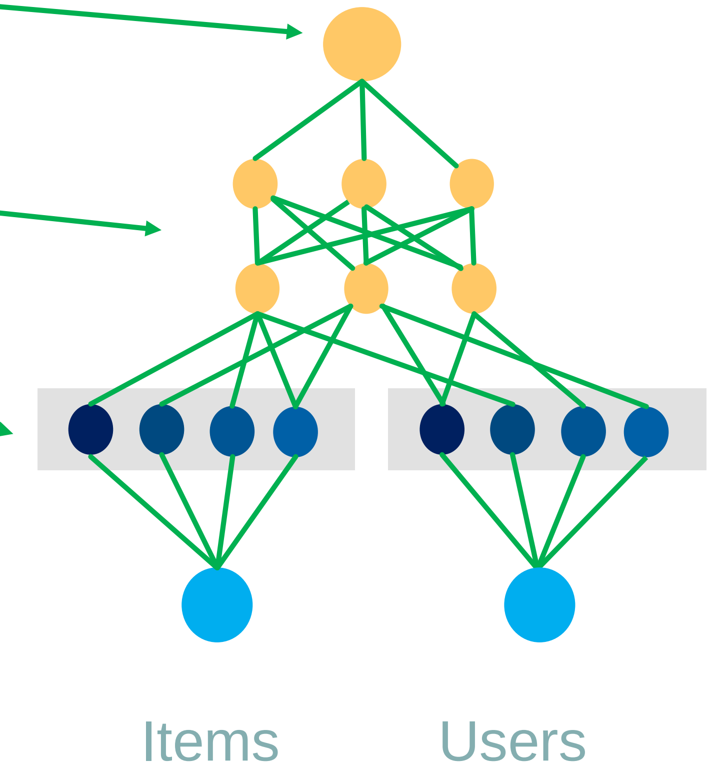
    if row[UID] not in customerDict.keys():
        customerDict[UID]=nCustomers
        nCustomers+=1
    if row[ItemID] not in itemDict.keys():
        itemDict[row[ItemID]] = nItems
        nItems +=1
```

- **dataSales[UID] = dataSales[UID].apply(lambda x: customerDict[x])**
- **dataSales[ItemID] = dataSales[ItemID].apply(lambda x: itemDict[x])**

Model - architecture

- ReLu instead of Sigmoid.
- Removed linear part.

- Deep Hidden Layers
- Embeddings



MODEL – I/o

- **# Placeholders for the embeddings and the outputs**
- **def make_placeholders(batch_size, parameters):**

```
placeholders = {}
placeholders['N'] = ng.make_axis(length=batch_size, name='N')
placeholders['Y'] = ng.placeholder(axes=[placeholders['N']], name="Y")

embeddings_placeholders = []
for lut in range(parameters[‘number_of_embeddings’]):
    embedding_placeholder = ng.placeholder(ng.make_axes([placeholders['N']]), name="EMB")
    embeddings_placeholders.append(embedding_placeholder)

placeholders['embeddings_placeholders'] = embeddings_placeholders

return placeholders
```

MODEL - Embeddings

- *# Embeddings Layers*
  # In this case we have 2 embeddings
- *# Customers and Items.*

- ```
  luts = []
  for e in range(parameters[‘number_of_embeddings’]):
      init_uniform = UniformInit(0, 1)
       lut = LookupTable(parameters['tokens_in_embeddings'][e], parameters['dimensions_embeddings'][e],
                        init_uniform, pad_idx=0, update=True)

      luts.append(lut)
  ```
  ```

MODEL – Deep model

- **# Deep Hidden Layers**
- deep_parameters = [32, 16, 16, 8]

```
layers = []

drop_out_rate = 0.1

for i in range(len(deep_parameters)):
    layers.append(Affine(nout=deep_parameters[i], weight_init=init_xavier, activation=Rectlin()))
    if drop_out_rate > 0.0:
        layers.append(Dropout(keep=drop_out_rate))

layers.append(Affine(axes=tuple(), weight_init=init_xavier))

deep_layers = Sequential(layers)
```

MODEL – building the graph and losses

- **# Building the graph**
```
inputs = make_placeholders(batch_size)

embedding_ops = []

for idx, lut in enumerate(luts):
    embedding_op = lut(inputs['embeddings_placeholders'][idx])
    embedding_ops.append(embedding_op)

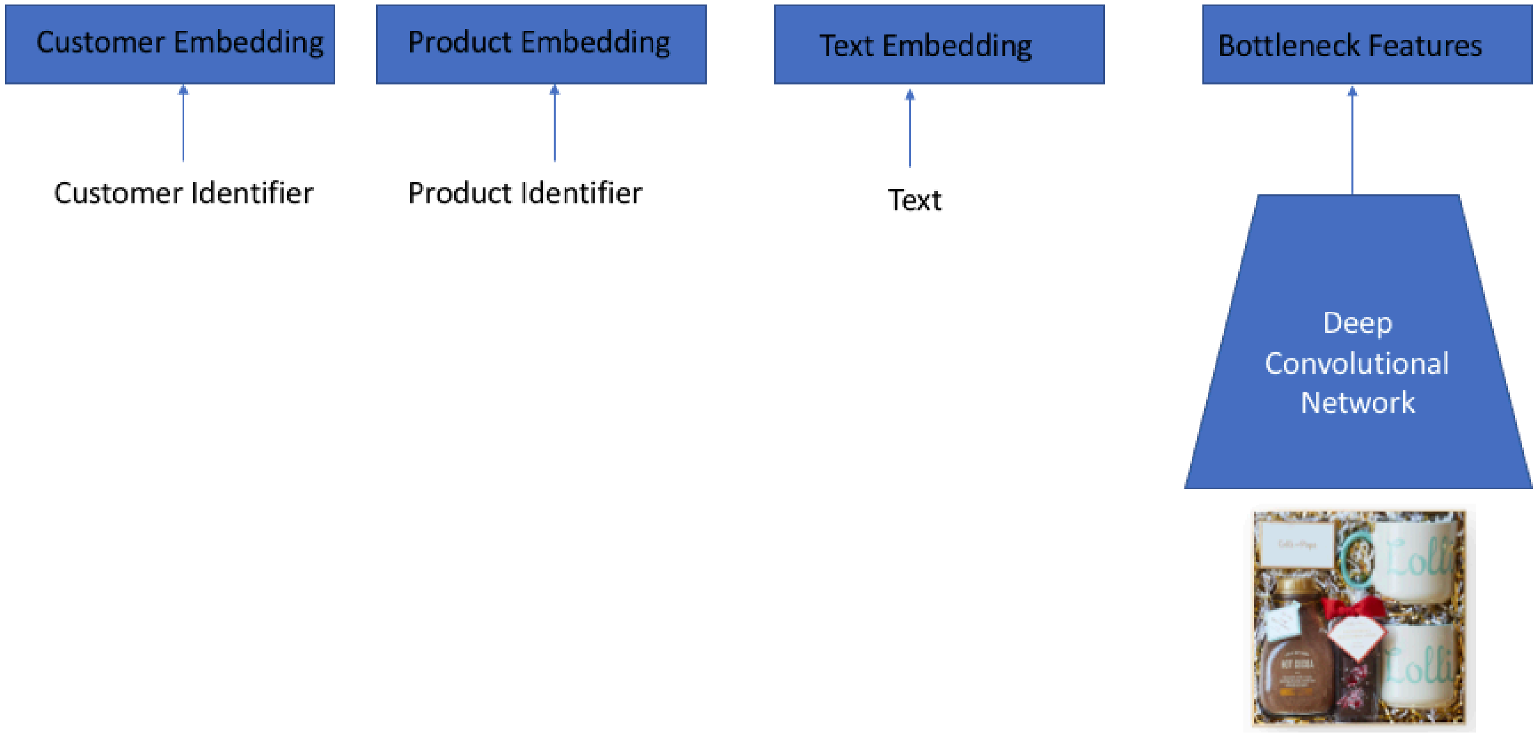
X_deep = ng.concat_along_axis(embedding_ops,ng.make_axis(name="F"))

deep = ng.maximum((deep_layers(X_deep) + ng.variable((), initial_value=0.5).named('b')),0)
```

- **# RMS**
```
loss = ng.squared_L2(deep - inputs['Y’'])
```

- **# L2 loss for regularization purposes**
```
loss = ng.sum(loss, out_axes=[])+ng.squared_L2(deep)
```

Text and Image data



Text data – i/o

- ***# Create a new column with embeddings***
  dataSales[**"Text Embedding"**] = dataSales[**"Item Id"**].apply(**lambda** x: glove.getMeanEmbedding(x))
- ***# In this example we are using a 50d glove vector embedding***
  placeholders[**'embedding_dimension'**] = ng.make_axis(length=50, name=**"F"**)
  placeholders[**'text_embedding'**] = ng.placeholder(axes=[placeholders[**'embedding_dimension'**],placeholders[**'N'**]], name=**"X_d"**)

Text data – ingestion

- # Embeddings data
- glove = pd.read_csv('glove.6B.50d.txt', sep=" ",index_col = 0, header= None, quoting=csv.QUOTE_NONE)
- # Retrieving Embedding and missing data.
- def vec(w, glove):

```
    if w in glove.index:
        return glove.loc[w].as_matrix().T
    else:
        return np.ones(50)

def getAvrGlove(shortDesc):
    ns = shortDesc.values[0].split(" ")
    a = np.empty(shape=[len(ns),50])
    i=0
    for w in ns:
        wl = w.lower()
        vecr = vec(wl)
        a[i] = vecr
        i+=1
        a=np.mean(a,axis=0)
    return a
```

image data – bottleneck features

- # Getting the bottleneck features for the entire dataset.
- base_model = InceptionV3(weights='imagenet', include_top=False, pooling = 'max')

```
bfdict = {}
if product1 in filesIds:
    filename="Images/" + filesIds[product1]
    bf = getBottleNeckFeatures( filename, base_model)
    bfdict[product1] = bf


# Save
np.save('bottleneckFeatures.npy', bfdict)
```

image data – process an image

- # Get the bottleneck features for a given image.

```
def getBottleNeckFeatures(product, image, model):
    bf = []
    if image is not None:
        img = Image.open(image)
        bf    = bfeatures(model, img, target_size)
        bf    = bf.flatten()
    return bf
```

image data – computing the features

- **# Computing the bottleneck features**
- target_size = (299, 299)
- **def** bfeatures(model, img, target_size):

```
def bfeatures(model, img, target_size):
  if img.size != target_size:
    img = img.resize(target_size)

  x = image.img_to_array(img)
  x = np.expand_dims(x, axis=0)
  x = preprocess_input(x)

  bf = model.predict(x)
  return bf
```

Evaluation Metric

The customer was interested in recommending the top 1 or 3 items of each customer.

The metric chosen were top@3 and top@1 also known as precision at k.

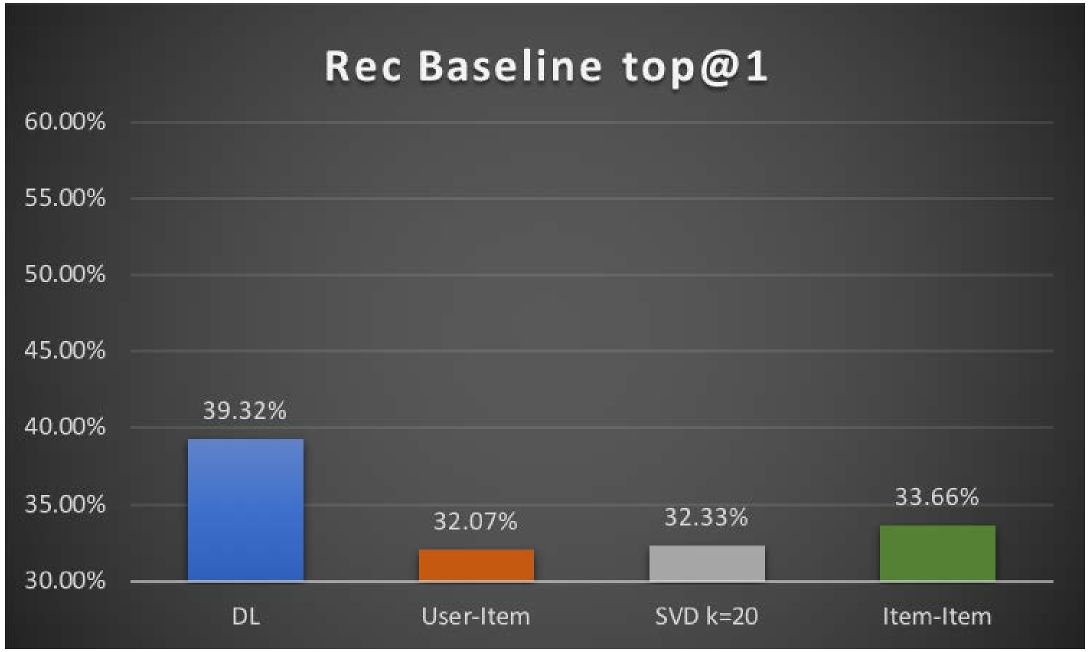
Other metrics of interest if you want to used a rank web based perspective on the results:

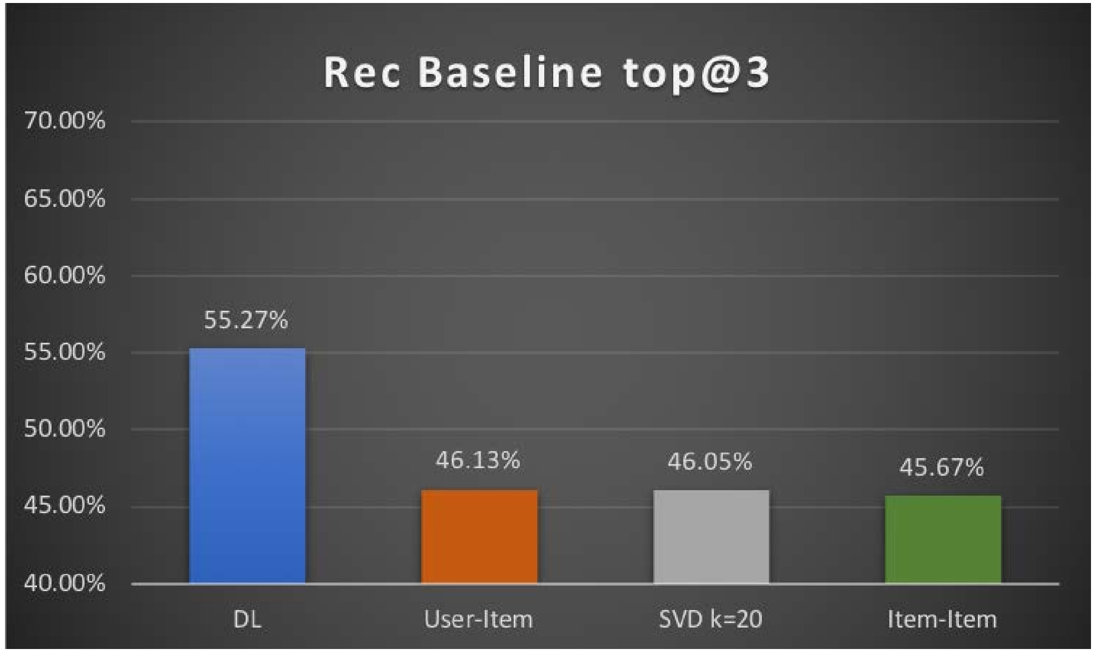
CG   : Cumulative Gain

DCG   : Discounted Cumulative Gain

NDCG : Normalized Discounted Cumulative Gain

results





Number of customers : 50K
Number of Items : 3K
Number of transactions : 250K

Call to action

- Try Deep Learning in your dataset or public available data.
- Check and Try the WD demo example in nGraph Library ® demos.
- Movielens a classical dataset to experiment.
- Netflix Prize Data another classic dataset.

https://grouplens.org/datasets/movielens/
https://www.kaggle.com/netflix-inc/netflix-prize-data

questions

mariano.j.phielipp@intel.com

Legal Notices and Disclaimers

- Intel technologies' features and benefits depend on system configuration and may require enabled hardware, software or service activation. Performance varies depending on system configuration. No computer system can be absolutely secure. Check with your system manufacturer or retailer or learn more at intel.com.
- Tests document performance of components on a particular test, in specific systems. Differences in hardware, software, or configuration will affect actual performance. Consult other sources of information to evaluate performance as you consider your purchase. For more complete information about performance and benchmark results, visit http://www.intel.com/benchmarks.
- No license (express or implied, by estoppel or otherwise) to any intellectual property rights is granted by this document.
- Intel does not control or audit third-party benchmark data or the web sites referenced in this document. You should visit the referenced web site and confirm whether referenced data are accurate.
- Intel, the Intel logo and others are trademarks of Intel Corporation or its subsidiaries in the U.S. and/or other countries.

intel® AI