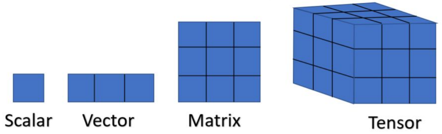
FunFact

Tensor Decomposition, Your Way

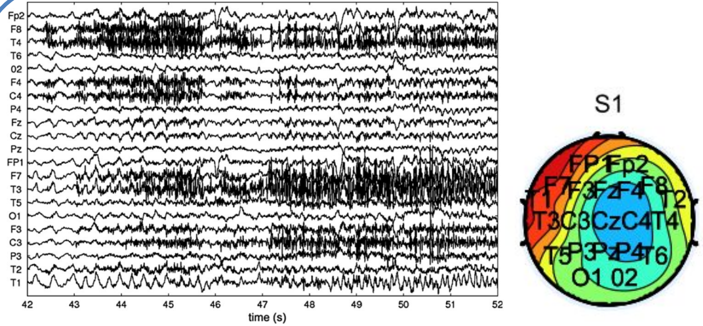
Daan Camps, Yu-Hang Tang
Lawrence Berkeley National Laboratory, NVIDIA

10th International Congress on Industrial and Applied Mathematics

Tensors decompositions have many applications

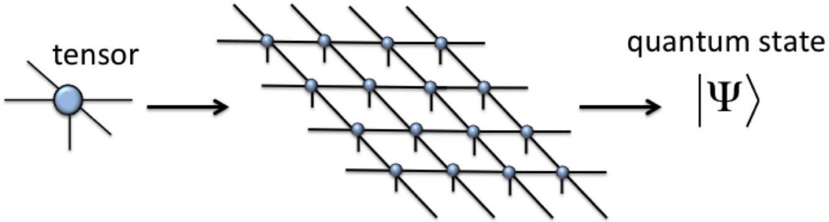


Multiway data



Unsupervised learning: Blind source separation

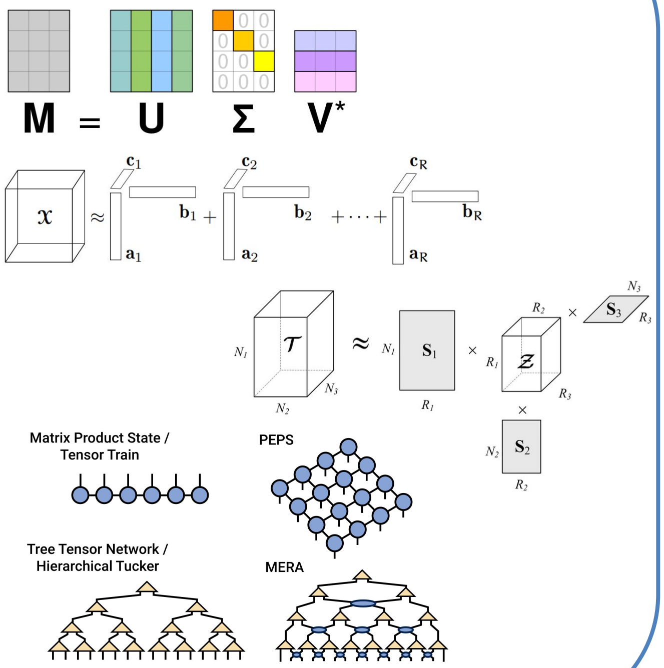
Image and video compression



Quantum physics

A zoo of decompositions and algorithms

Decompositions

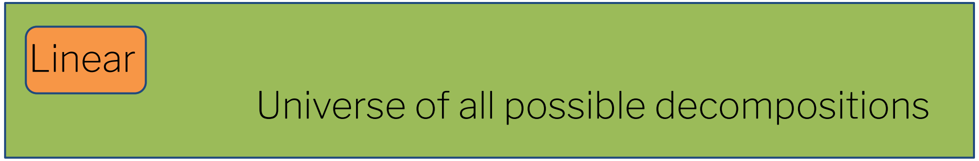


Algorithms

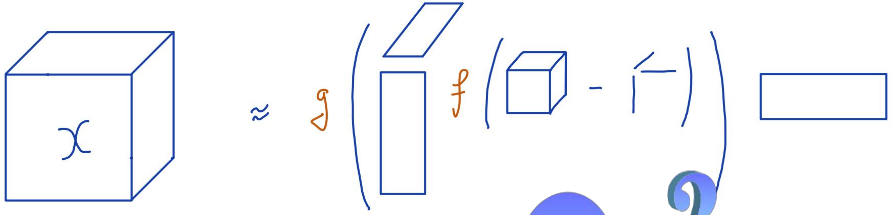
- Bidiagonalization
- Alternating Least-Squares
- CG
- ADMM
- DMRG
- Gradient based
- …

Every decomposition requires specialized algorithms

All impose linear contractions between factor tensors



FunFact: Instantaneous time-to-algorithm



Traditional workflow:

- Analyze model
- Formulate and implement algorithm
- Validate results

Process of days/weeks/months/years

Expert knowledge required

FunFact workflow:

- Write model as (nonlinear) tensor expression
- Factorize data and validate results

Process of minutes/hours

Accessible for non-experts

Behind the scenes of FunFact

Frontend: a tensor algebra language through an embedded domain specific language (eDSL) that combines NumPy API and generalized Einstein notations

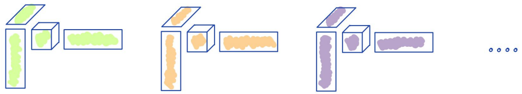
$$\boldsymbol{c}_i = \boldsymbol{a}_{ij}\boldsymbol{b}_j \qquad\qquad \boldsymbol{c}_i = \sum_j \boldsymbol{a}_{ij}\boldsymbol{b}_j$$

Backend: modern NLA libraries that support autograd on GPUs

PyTorch

NumPy

Algorithm: stochastic gradient descent with multi-replica learning



Example Workflow: Hello World!

```
!pip install funfact
import funfact as ff
```

install from PyPI and load package

```
a = ff.tensor('a', 50, 3)
b = ff.tensor('b', 3, 20)
i, j, k = ff.indices('i, j, k')
```

declare tensors and indices

```
tsrex = a[i, k] * b[k, j]
```

write tensor expression

Lazy evaluation: writing down a tensor expression does not trigger immediate evaluation. Rather, the abstract syntax tree (AST) of the calculation is saved for future use.

```
target = load_data(...)
ff.factorize(target, tsrex)
```

factorize target data tensor into tensor expression

Let's talk about grammar!

Rule

- An elementwise function evaluation of a tensor expression yields a new tensor expression.
- Binary operations between two tensor expressions yields a new tensor expression.
- Unary operations on a tensor expression yields a new tensor expression.
- An index notation is by itself a tensor expression.
- A tensor is by itself a tensor expression.
- A literal value is by itself a tensor expression.

Backus-Naur Form

```
tsrex -> f(tsrex) |
         tsrex binary_operator tsrex |
         unary_operator tsrex |
         index_notation |
         tensor |
         literal
```

Most common math routines in NumPy can be used as elementwise functions.

```
f -> abs   | exp   | log   |
     sin   | cos   | tan   |
     asin  | acos  | atan  | atan2  |
     sinh  | cosh  | tanh  |
     ...
```

Valid binary operators are multiplication, division, addition, subtraction, exponentiation, Kronecker product, and matrix multiplication.

```
binary_operator -> *  | /  | +  | -  |
                   ** | &  | @
```

A tensor expression, regardless of its complexity, can be indexed by an index set whose size is consistent with its dimensionality.

```
index_notation -> tsrex[indices]
```

Index notation and index modifiers

Rule

A valid index set consists of zero or more index variables, each of which can be optionally decorated with the ~ and * modifier.

Backus-Naur Form

```
indices -> |
           index |
           indices,  index |
           indices, ~index |
           indices, *index
```

- repeated indices in a tensor expression are normally contracted (**einsum**)
- ~ modifier indicates explicit **non-reducing/non-contracting index**
- * modifier indicates a **Kronecker index**

Example: Khatri-Rao product

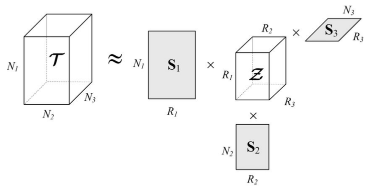
$$C = A \odot B := [a_1 \otimes b_1 \; a_2 \otimes b_2 \; \cdots \; a_n \otimes b_n],$$

```
import funfact as ff
a = ff.tensor('a', 5, 2)
b = ff.tensor('b', 3, 2)
c = ff.tensor('c', 5, 4)
i, j = ff.indices('i, j')
# (standard) Khatri-Rao product of a and b with shape 15 x 2 :
tsrex = a[[*i, ~j]] * b[i, j]
# row-wise Khatri-Rao product of a and c with shape 5 x 8 :
tsrex = a[[~i, *j]] * c[i, j]
```

Complex decompositions in a concise expression

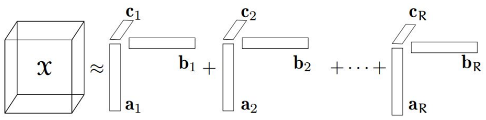
Tucker decomposition

```
tucker = Z[r1, r2, r3] * S1[r1, n1] * S2[r2, n2] * S3[r3, n3]
```



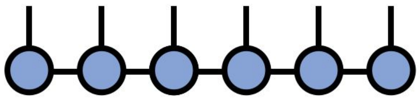
Tensor-rank decomposition

```
tensor_rank = (a[i, ~r] * b[j, r]) * c[k, r]
```



Tensor train decomposition

```
tensor_train = G1[i1, r1] * G2[i2, r1, r2] * G3[i3, r2, r3] *
               G4[i4, r3, r4] * G5[i5, r4, r5] * G6[i6, r5]
```



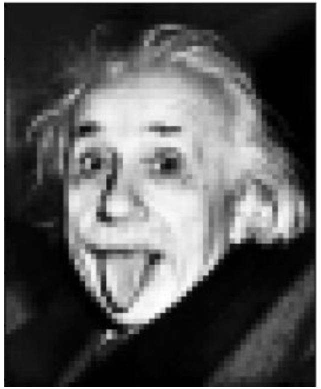
Example: Image compression through nonlinear factorization

SVD gives the best rank-r approximation

$$M = U \Sigma V^* \qquad\qquad M \approx U_r \Sigma_r V_r^*$$

```
U, S, V = np.linalg.svd(img)
```

Original



24 ranks

MSE loss: 4.31e-4

12 ranks

MSE loss: 1.95e-3

6 ranks

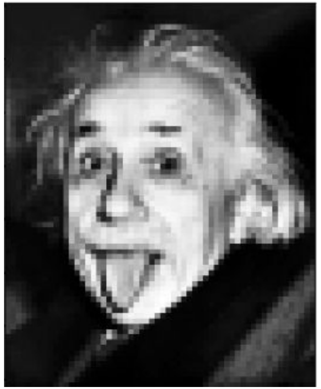
MSE loss: 4.59e-3

Example: Image compression through nonlinear factorization

FunFact finds the same solution

```
low_rank = u[i, r] * v[j, r]
```

Original



24 ranks

MSE loss: 4.31e-4

12 ranks

MSE loss: 1.95e-3

6 ranks

MSE loss: 4.59e-3

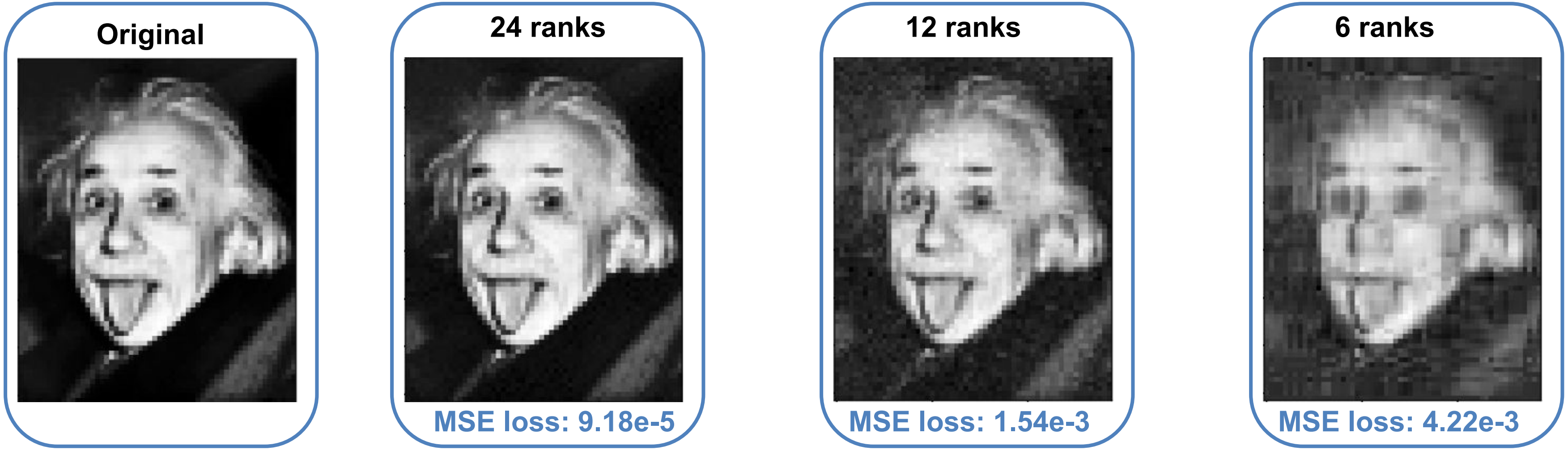
Example: Image compression through nonlinear factorization

```
rbf = ff.exp(-(u[i, ~k] - v[j, ~k])**2) * a[k] + b[[]]
```

arXiv:2106.02018

Linear combination of RBFs:

$$A_{ij} \approx \exp\left(-\left(\boldsymbol{u}_{i\tilde{k}} - \boldsymbol{v}_{j\tilde{k}}\right)^2\right)\boldsymbol{a}_k + \boldsymbol{b}$$



Nonlinear models achieve lower loss for same data complexity

SVD

24 ranks

MSE loss: 4.31e-4

12 ranks

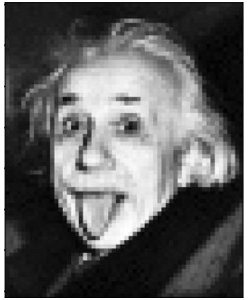
MSE loss: 1.95e-3

6 ranks

MSE loss: 4.59e-3

RBF

24 ranks



MSE loss: 9.18e-5

12 ranks

MSE loss: 1.54e-3

6 ranks

MSE loss: 4.22e-3

At least 10% reduction in MSE for same storage cost!

Conditions and Penalties

In many applications, the tensors in a tensor expression must satisfy certain condition(s):

```
from funfact.conditions import (
    UpperTriangular, Unitary, Diagonal, NonNegative
)
```

The condition is added to a tensor as a **preference:**

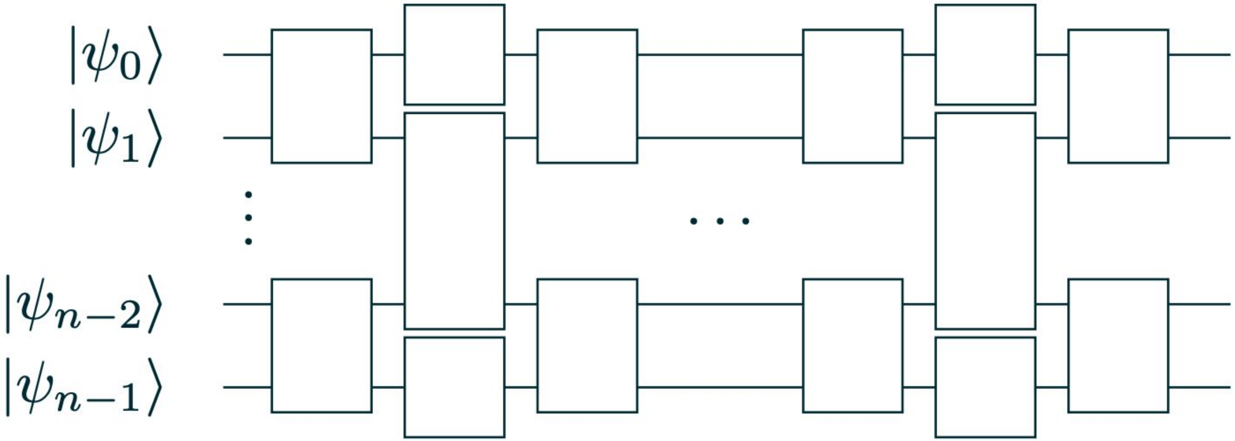
```
T = funfact.tensor(...,
        prefer=Unitary(
            weight=1.0,
            elementwise='mse', #'l1'
            reduction='mean'   #'sum'
        )
)
```

And included in the optimization as a **penalty:**

```
ff.factorize(target, tsrex, penalty_weight=1.0)
```

Example: Quantum circuit compilation as a tensor decomposition

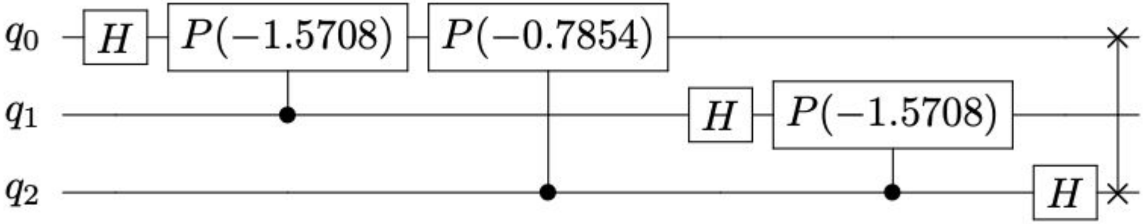
- Quantum circuit synthesis or compilation is the task of finding a quantum gate representation for a given unitary operator
- This problem can be formulated as a tensor decomposition problem



Quantum Circuit Synthesis of Fourier Transform

Quantum Fourier Transform DOI: 10.1002/nla.2331

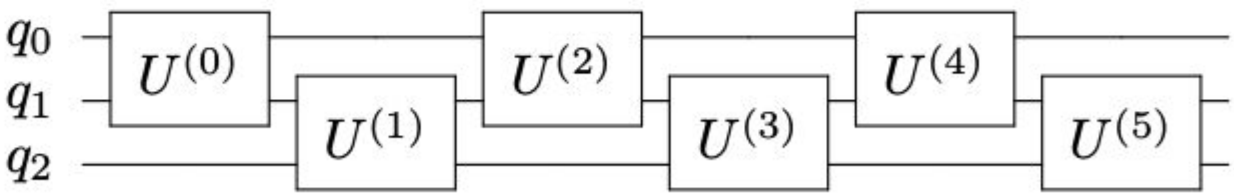
- O((log N)^2) circuit is known



- Might not correspond to hardware qubit topology

Nearest-Neighbor Connectivity

- The simplest topology is nearest-neighbor connectivity



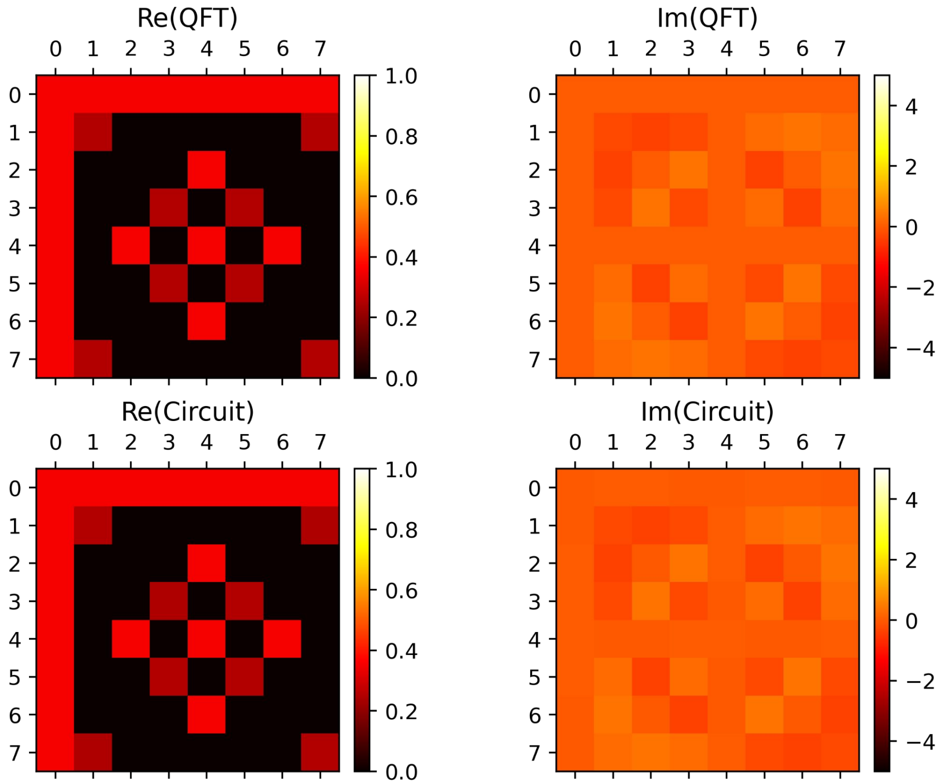

```
def two_qubit_gate(i: int, n: int):
    G = ff.tensor(4, 4, prefer=cond.Unitary)
    return ff.eye(2**i) & G & ff.eye(2**(n-i-2))
```

```
circuit3 = two_qubit_gate(1, 3) @ \
           two_qubit_gate(0, 3) @ \
           two_qubit_gate(1, 3) @ \
           two_qubit_gate(0, 3) @ \
           two_qubit_gate(1, 3) @ \
           two_qubit_gate(0, 3)
```

Optimizing the circuit as a tensor expression

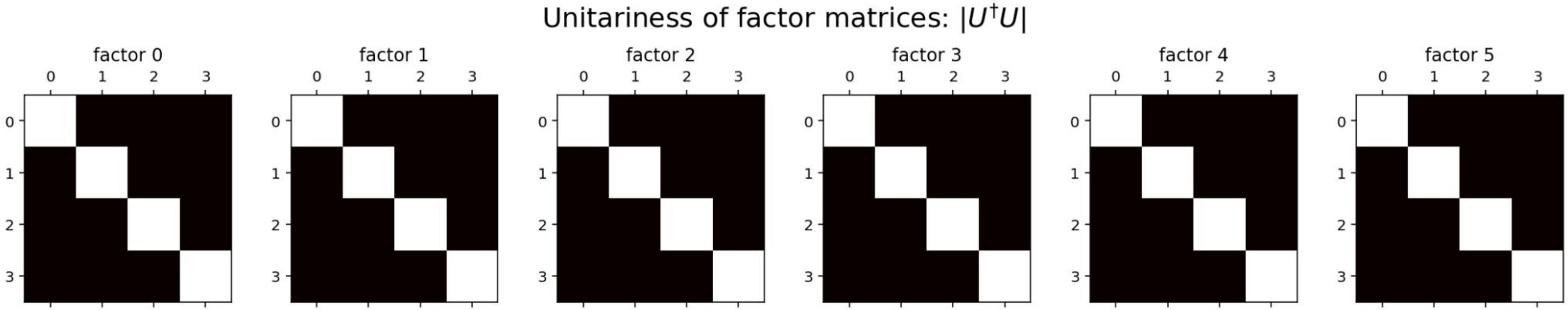
```
circuit_fac = ff.factorize(
    circuit3, QFT_matrix3,
    max_steps=1000,
    tol=1e-3,
    lr=7e-2,
    vec_size=32,
    loss=MSE_loss,
    dtype=ab.complex64,
    checkpoint_freq=40,
    penalty_weight=2.0
)
```

```
28%|███        | 280/1000 [00:02<00:07, 97.39it/s]
```



Optimizing the circuit as a tensor expression

```
loss: 0.009713371542746886
penalty: 8.032669575186446e-05
```



Conclusion

- FunFact is a rich and flexible language for (non-)linear tensor algebra expressions
- FunFact can solve the inverse problem thanks to modern NLA backends such as JAX and PyTorch
- Dramatically reduced time-to-algorithm for new tensor factorization models

Released V1.0 under BSD license

Find out more at:

- **funfact.readthedocs.io**
- **github.com/yhtang/FunFact/**
- **pypi.org/project/funfact/**

Funding acknowledgment:
LDRD No. DE-AC02-05CH11231

slides available at: https://tinyurl.com/funfact-iciam