# FunFact

## Build Your Own Tensor Decomposition Model in a Breeze

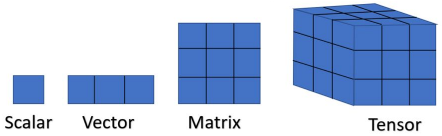
Camera space


Daan Camps
Scalable Solvers Group
Applied Mathematics and Computational Research Division

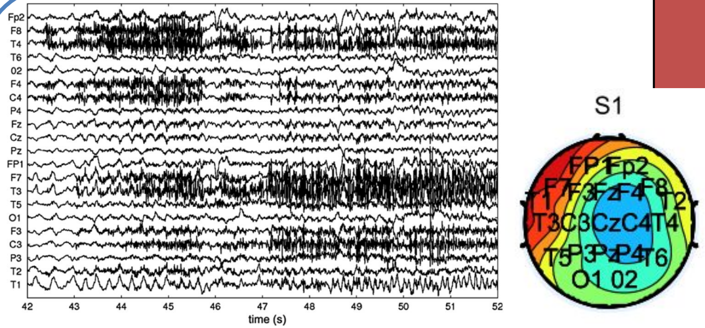

2022 CS Postdoc Symposium Presentation

# Tensors decompositions have many applications

Camera space

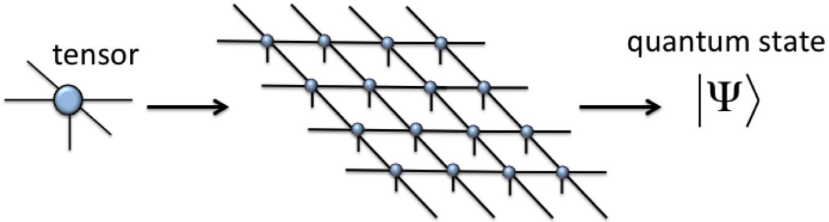



Multiway data




Unsupervised learning: Blind source separation

Image and video compression

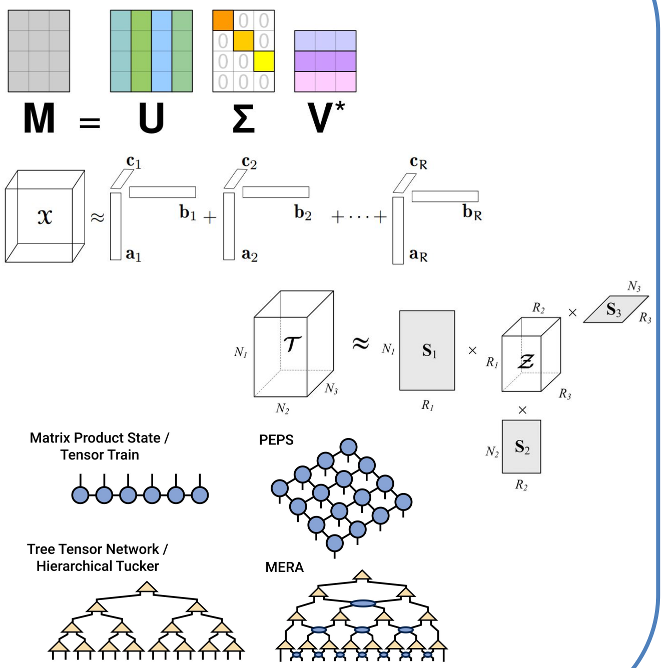



Quantum physics

# A zoo of decompositions and algorithms

Camera space

## Decompositions

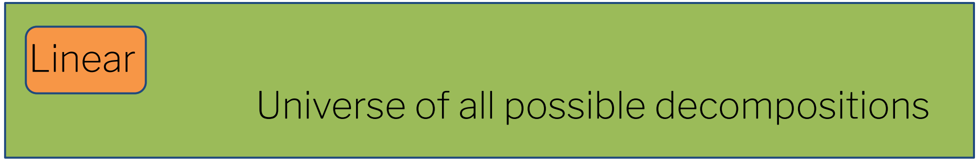



## Algorithms

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

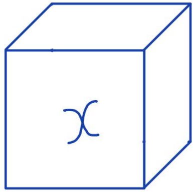
**Every decomposition requires specialized algorithms**

**All impose linear contractions between factor tensors**



# Instantaneous time-to-algorithm

Camera space



**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

Camera space

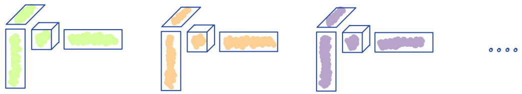
**Frontend:** a tensor algebra language through an embedded domain specific language (eDSL) that combines NumPy API and generalized Einstein notations

$$c_i = a_{ij} b_j \qquad\qquad c_i = \sum_j a_{ij} b_j$$

**Backend:** modern NLA libraries that support autograd

JAX PyTorch NumPy

**Algorithm:** stochastic gradient descent with multi-replica learning



# Hello World

Camera space

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

install from PyPI and load

```
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 AST of the calculation is saved for future use.

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

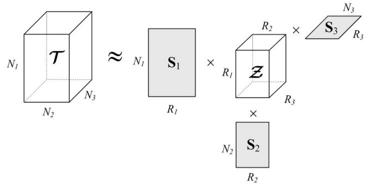
factorize target tensor

# Complex decompositions in a concise expression

Camera space

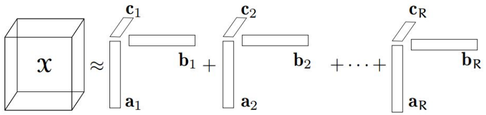
## Tucker decomposition

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



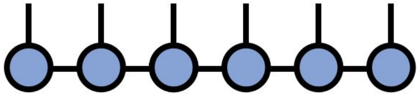

## Tensor-rank decomposition

```
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]
```



# Image compression through nonlinear factorization

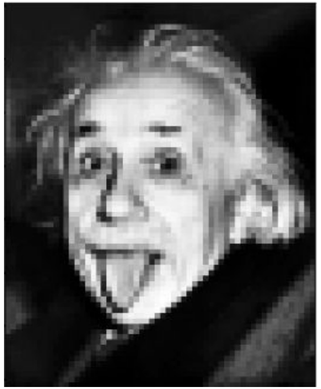
Camera space

## 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**

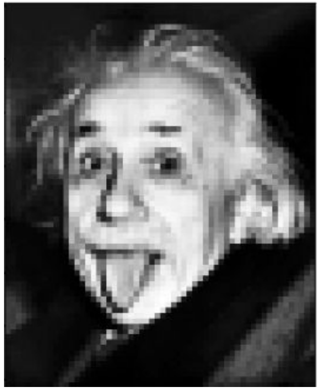
# Image compression through nonlinear factorization

Camera space

## 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**

# Image compression through nonlinear factorization

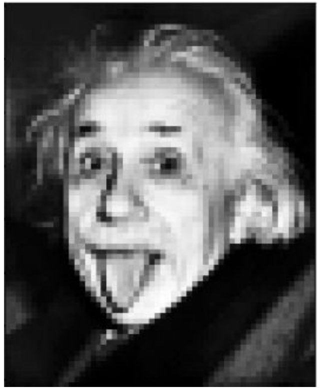
Camera space

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

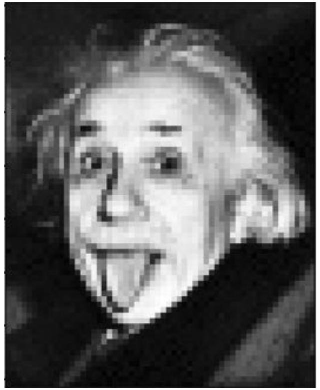
$$A_{ij} \approx \exp\left(-\left(\boldsymbol{u}_{i\tilde{k}} - \boldsymbol{v}_{j\tilde{k}}\right)^2\right)\boldsymbol{a}_k + \boldsymbol{b}$$

arXiv:2106.02018

**Original**

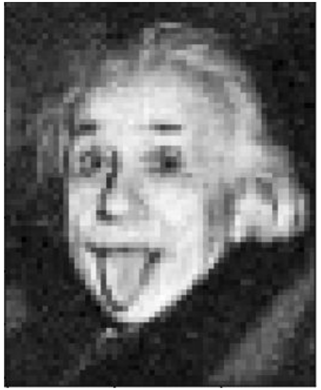


**24 ranks**



MSE loss: 9.18e-5

**12 ranks**



MSE loss: 1.54e-3

**6 ranks**

MSE loss: 4.22e-3

# Nonlinear models achieve lower loss for same data complexity

Camera space

**SVD**

24 ranks

MSE loss: 4.31e-4

12 ranks

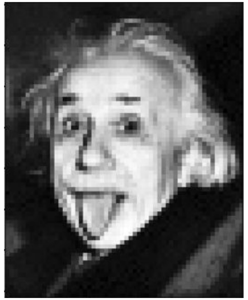
MSE loss: 1.95e-3

6 ranks

MSE loss: 4.59e-3

**RBF**

24 ranks



MSE loss: 9.18e-5

12 ranks

MSE loss: 1.54e-3

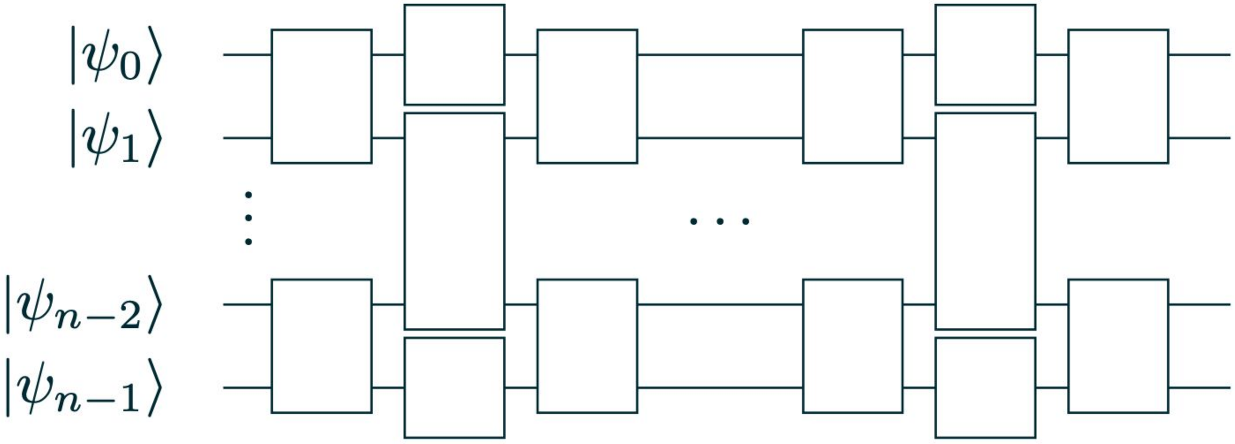
6 ranks

MSE loss: 4.22e-3

# Quantum circuit compilation as a tensor decomposition

Camera space

- 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

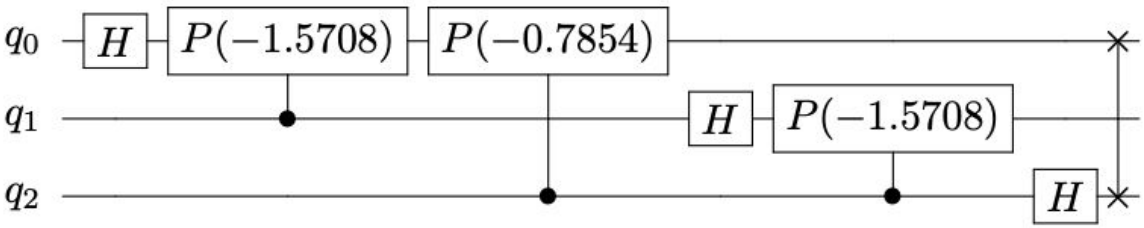


# Quantum Circuit Synthesis of Fourier Transform

Camera space

**Quantum Fourier Transform** DOI: 10.1002/nla.2331

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

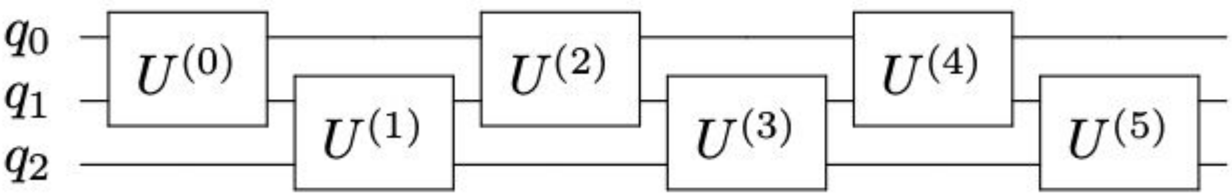



- Might not correspond to hardware qubit topology

# Nearest-Neighbor Connectivity

Camera space

- The simplest topology is nearest-neighbor connectivity




```python
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))
```

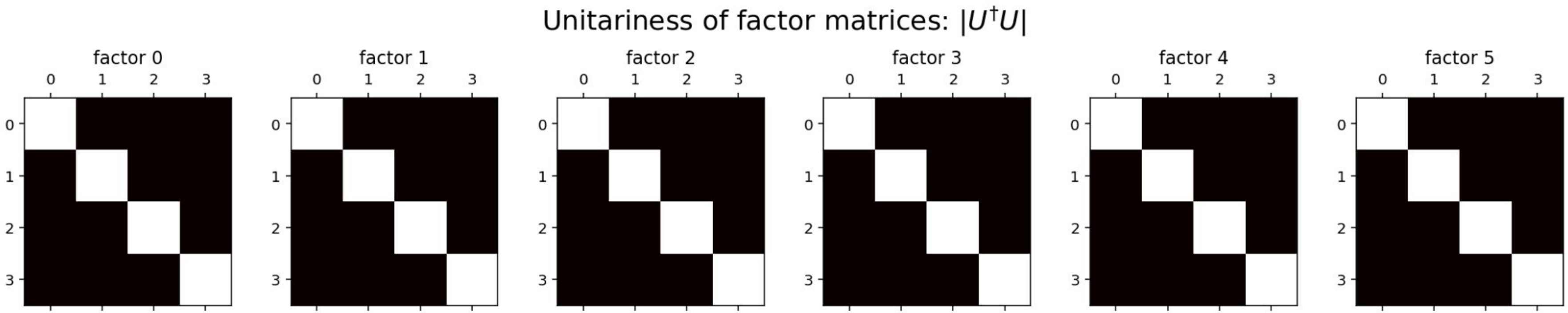
```python
circuit3 = two_qubit_gate(1, 3) @ \
           two_qubit_gate(0, 3) @ \
           two_qubit_gate(1, 3) @ \
           two_qubit_gate(0, 3) @ \
           two_qubit_gate(1, 3) @ \
           two_qubit_gate(0, 3)
```

# Optimizing the circuit as a tensor expression

Camera space

```
circuit3_fac = ff.factorize(QFT3, circuit3, ...)
```

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

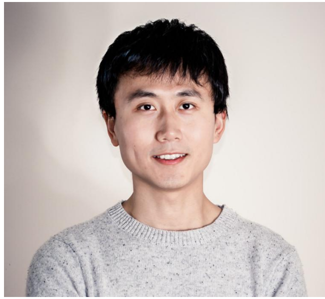


# The Team

Camera space

FunFact is developed in collaboration with:

- Yu-Hang Tang (SSG)



- Liza Rebrova (previous contributor)

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

# Conclusion

Camera space

- 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 V1.0RC under BSD license**

**Find out more at:**

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

**We're looking for users and applications!**

**Don't hesitate to reach out at dcamps@lbl.gov**