Thank you for the introduction!

The Simulation of Quantum-Many body systems is a very challenging problem. A general many-body wavefunction that describes a system with N subsystems can be written as follows. We sum over basis states, which are made up of the local basis states of the subsystems, for example subsystem 1 can be in d\_1 states, and so this sum goes up to d\_1. The amplitude of each basis vector is stored in this complex number Psi. We can now count how many parameters we need to describe this state and it turns out to be d^N, which is of course exponential in system size. Because of this exponential scaling it is very hard to do anything with the wave function, and thus we need approximations. 1min

This brings me to the contents of my talk. I will start by introducing Matrix Product States (MPS), which have great success in tackling this problem in one-dimensional systems.

Next, I will talk about isometric Tensor Product States (isoTPS), which generalize the canonical form of MPS to two and higher dimensions. These isoTPS were introduced in 2019.

In the third and biggest section of my talk I will discuss the work of my thesis, which was implementing an alternative canonical form for isoTPS, which we call YB-isoTPS for reasons I will talk about later. I will discuss how this canonical form is defined and how certain algorithms can be implemented. I will then show benchmarks of this method on the Transverse Field Ising Model. 2min

Before I start with MPS, I want to give a quick general introduction to tensor networks. We define a tensor of rank n to be an n-dimensional array of complex numbers T. It is convenient to introduce a diagrammatic notation. In this notation, we draw tensors as shapes, and indices as lines emerging from these shapes. A few simple examples: A vector has only one index, so we can draw it as just a circle with a singular index sticking out. A matrix has two indices which we call I and j, and last we draw a general tensor of rank n.

I want to also introduce the linear operation called index contraction. An index contraction of two tensors is performed by multiplying the two tensors, summing over certain connected indices. A simple example would be the scalar product of two vectors. Both vectors have a single index alpha, over which we sum here. The result is a scalar with zero indices. In the diagrammatic notation, index contractions are denoted by connecting the bonds that are contracted. Vectors have one index, so we can just connect the two legs of A and B to form the scalar product. 3 min

Similarly, the matrix product of two matrices is an index contraction, and can also be drawn as a tensor diagram.

Ok, with this we can then define tensor networks as a set of tensors that is contracted in a specified order. For example, here we show a simple tensor network of three tensors that are contracted in a triangle shape.

Matrix product states are a powerful Ansatz for one-dimensional quantum states, for example spins on a chain. I write here again the many-body wavefunction. Note that we can interpret the amplitude Psi as a tensor of rank N. The idea of MPS is now to write this large tensor as a contraction of many smaller tensors T of rank 3. If we additionally restrict the bond dimension between these tensors to a maximum value of chi, we have then reduced the number of parameters from d^N to N chi^2 d, which is linear in system size. Of course this is an approximation. However, as it turns out, ground states of local, gapped Hamiltonians have an entanglement structure that follows an area law, which means that when cutting the chain into two parts, the entanglement between the left and right part is smaller than a constant. Remarkably, these states are exactly those that can be represented efficiently by the MPS ansatz. To go from the large tensor Psi to the MPS, one can for example use a series of Singular Value decompositions. 4:40

A powerful feature of MPS is the existence of a canonical form which greatly speeds up many algorithms. To discuss this canonical form, we must first introduce isometries. An isometry is a m x n matrix W with n >= m, which satisfies the so called isometry constraint: The adjoint of W multiplied with W must be equal to the identity, while multiplying the other way gives a projector P. In tensor diagrams we draw this by decorating the legs of the tensor with arrows. As a convention we draw the larger dimension, in this case n, with incoming arrows, and the smaller dimension, in this case m, with outgoing arrows. The isometry condition can then be visualized as follows: Contracting W with its complex conjugate along the larger dimension gives identity, which we visualize with just a line, and contracting the other way gives a projector. An isometric tensor is then a tensor that can be transformed into an isometry by grouping all the legs with incoming and outgoing arrows respectively, as shown here. 6:00

We can now bring an MPS into the canonical form as follows: Starting from a normal MPS, we perform QR-decompositions starting from the left. A QR-decomposition decomposes a tensor into an isometry Q and an upper triangular matrix R. We do the same starting from the right, until we meet at the center. We then call all tensors with arrows pointing to the right A and all tensors with arrows pointing to the left B. The single tensor with only incoming arrows is called the orthogonality center Lambda. Now, the A tensors fulfil a left-isometry condition: They turn into identity when contracting with the complex conjugate from the left. The B tensors similarly fulfil a right-isometry condition. Note that no information was last, we were able to exactly transform the MPS into canonical form. 7 min

This allows us to simplify certain algorithms. As an example i want to show the computation of a local expectation value. Let us assume we want to compute the expectation value of an operator that acts on a site in the center of the chain. We can interpret the operator as a tensor of rank two, and computing the expectation value can then be drawn as a tensor network as follows: We contract the MPS with its complex conjugate, sandwiching the operator in between. Without the canonical form, computing this expectation value actually scales linearly with the system size, because we need to contract the full network. However, since were in the canonical form, we can successively use the isometry condition to turn the left and right boundaries into identities, reducing the expectation value to a contraction of only 3 tensors! 8 min

I want to introduce one more algorithm for MPS, which can be used to perform time evolution. To evolve the state by a time Delta t, we need to apply the time evolution operator U to the state, which is given by the exponential of -i\*\Delta t\*the Hamiltonian of the model we want to look at, where we set hbar to 1. Let us assume that the Hamiltonian can be split into terms acting only on neighboring sites j and j+1. We can then split the Hamiltonian into two terms, one acting only on even and one acting only on odd bonds. We can then approximate this exponential by a so-called Suzuki-Trotter decomposition with an error of Delta t^2. We have now split the exponentials of even and odd terms. Because all operators acting on even bonds commute with each other, this exponential factorizes into these bond operators acting only on neighboring bonds, and the same holds true for the odd terms. 9 min

If we now want to evolve the MPS by one time step, we need to apply these operators to the MPS. We can draw this as a tensor network as follows: Here we have the MPS, and we act first with the bond operators on all even bonds and then on all odd bonds. Now, we need to approximate this network with an updated MPS. This can be done by absorbing the bond operators into the MPS one after the other. A single bond operator can be absorbed by contracting it with the two MPS tensors into a tensor theta, and then splitting this tensor again with a truncated SVD. 9:45

Next, let me introduce isometric Tensor Product States (isoTPS), which generalize the canonical form of MPS to two and higher dimensions. First of all, the generalization of MPS to higher dimensions is known as Projected entangled pair states (PEPS), where we now write the wavefunction of a system on a 2D square lattice as the contraction C of tensors as shown in this tensor network. We call the bond dimension of the legs drawn in black D. The problem with this approach is that it is not possible to transform the network into a canonical form without error because of the presence of closed loops. Because of this, algorithms defined for PEPS are often very expensive and also instable. For example, a full TEBD update scales with the bond dimension D to the power of 10. 10:40

Isometric Tensor Product States are then defined as follows. The tensors are isometrized in a way that all arrows point towards a special row and column which is called the orthogonality hypersurface and is here drawn in red. Along the orthogonality hypersurface, all arrows point to a singular orthogonality center, which we draw in orange. Now, similar to MPS, we call columns left of the orthogonality hypersurface A and columns right of the orthogonality hypersurface B, while we call the orthogonality hypersurface itself Lambda. In contrast to MPS, a PEPS can not be transformed into an isoTPS without error. In fact, the expressional power of isoTPS is lower than PEPS, meaning that not all states that can be represented with PEPS can also be represented with isoTPS. 11:30

Most algorithms defined on isoTPS need to be able to move around the orthogonality center. Moving the orthogonality center along the orthogonality hypersurface is easy, and can be done via simple QR-decompositions.

Moving the orthogonality hypersurface to the left or to the right is a harder problem. Lets say the orthogonality column is at position n and we want to move it to position n+1. This problem can be solved by first splitting the column lambda^n into two columns A^n and Lambda, and then absorbing the column Lambda to the right. As a result we again obtain two columns, but now all arrows point to the right column. This last step of multiplying Lambda with the column B is equivalent to multiplying an MPS with a Matrix Product Operator, and this problem has existing efficient algorithms. For solving the first problem we want to minimize the distance Lambda^n – A^n contracted with Lambda. One idea is to variationally minimize this by sweeping over the two columns and optimizing one tensor at a time. However it is found that the so-called Moses-Move is more efficient while producing a solution that is almost as good as the variational one. 13:00

The Moses-Move can be performed as follows: We start at the bottom of the orthogonality column, and split the orthogonality center into three tensors using a tripartite decomposition. We then absorb the orange tensor into the next upper tensor. We now repeat this procedure, again splitting into three tensors and so on, until the top of the column is reached. The question is now how this tripartite decomposition can be performed. 13:30

A tripartite decomposition can be performed as follows: First, the tensor W is split into two tensors Q and theta via a singular value decomposition, where the bonds are also truncated to D^2. Next it is crucial to note that there is a gauge degree of freedom on the connecting bonds, we can insert a unitary and its adjoint. After absorbing U into theta and Udagger into Q, we then split theta via another SVD into two tensors B and C, finalizing the tripartite decomposition. In the third step, the unitary U can be chosen in such a way that it minimizes the truncation error of the final split in the last step. A good choice is a unitary that reduces the entanglement between the subsystems represented by the upper and lower indices of theta. This unitary is therefore called the disentangling unitary. We will discuss how to find a good disentangling unitary later. 15:00

In a similar way to MPS, one can perform a TEBD update step on isoTPS by first updating the tensors along all columns and then along all rows. Along the orthogonality hypersurface, the TEBD is very similar to MPS. The algorithm is much faster than for PEPS, reducing the computational complexity from D^10 to D^7! 15:20

In my thesis, I implemented an alternative canonical form, which was already thought of when the original isoTPS was developed but was never implemented. We call an isoTPS in this alternative canonical form Yang-Baxter isoTPS, for reasons that will become clear later. The canonical form is defined as follows: First, a PEPS is rotated by 45 degrees as shown here. We then introduce so-called auxiliary tensors, which we draw in red, and which make up the orthogonality hypersurface. The orthogonality hypersurface is placed in between two rows of PEPS tensors. Again, we isometrize such that all arrows point towards the orthogonality hypersurface, and towards a single tensor which is called the orthogonality center. Here, I also draw a unit cell, which now includes two sites. 16:20

In analogy to MPS and isoTPS, one can easily evaluate local expectation values. Lets say we want to compute the expectation value of an operator acting only on the two sites i and j. The two tensors representing sites I and j and the three neighboring auxiliary tensors form a sub-network of only incoming arrows. To compute the expectation value, we contract the state with its complex conjugate, sandwiching the operator in between. Now we can use the isometry conditions starting from the boundary, reducing contractions to identity, until we end up with a contraction of only the five marked tensors, there complex conjugates, and the bond operator. 17:00

As in the original isoTPS, most algorithms need to be able to move around the orthogonality center. Again, moving the orthogonality center along the orthogonality hypersurface is easy and just requires QR-decompositions. The problem arises again when we want to move the hypersurface to the left or to the right. 17:20

This can be done by starting with the orthogonality center at the bottom of the orthogonality hypersurface. Now, two auxiliary tensors are pulled through the physical tensor to the other side. Then we move up the orthogonality center and again pull two tensors through to the other side. The final tensor can then be easily moved by contracting and using a SVD. The main difficulty is the process of pulling the auxiliary tensors through the physical tensor, which we draw again here. We call this a Yang-Baxter move, because of the visual similarity to the Yang-Baxter equation. We call the tensors before this move T, W\_1 and W\_2, and denote the tensors after the move by T^prime, W\_1^prime and W\_2^prime. The error of the YB-move can be written as the norm of Psi – Psi^prime. We can expand this as the square root of the norm squared of psi + the norm squared of psi^prime – 2 times the real part of the overlap. We impose that both psi and psi^prime are normalized to one, and so this simply becomes the square root of 2 – the real part of the overlap. 18:30

The problem of finding a good YB move can therefore be written as a constrained optimization problem: The best tensors T^prime, W\_1^prime and W\_2^prime are the ones that maximise the real part of the overlap, under the constraint that T^prime and W\_1^prime are isometries and W\_2^prime is normalized to one, since it will be the new orthogonality center. In my thesis, I looked at two algorithms: A variational optimization with local updates and a tripartite decomposition similar to the one used in the MM. 19:10

Let’s start with the first of the two algorithms, the variational optimization. The real part of the overlap can be written as the following tensor network, where we have again used the isometry condition to turn a contraction of the full networks to a contraction of only a few tensors. Around the orthogonality center. One can now maximize this overlap by sweeping over the three lower tensors and optimizing them iteratively. For example, lets start by fixing W\_1^\prime and W\_2^\prime and only vary T\_prime. In this case the constrained optimization problem is known as the orthogonal Procrustes problem and permits a closed form solution. We update T\_prime with this closed form solution and then continue optimizing W\_1 and W\_2, which is in total one iteration of the algorithm. This is repeated until convergence. 20:00

Lets now qualitatively discuss how the algorithm performs. I plot here the approximation error against the number of iterations. For the tensors W\_1, W\_2 and T I choose an environment that was encountered during ground state search on the TFI model. As you can see here, the algorithm converges only very slowly, It is not converged even after 10’000 iterations. 20:30

The second algorithm for the YB move is a tripartite decomposition similar to the MM. We start by contracting the three tensors into a big tensor Psi. This tensor is then split horizontally via a truncated SVD, and we truncate the purple leg to D^2. In the next step, we split the connecting bonds into two bonds of bond dimension D. As in the MM, we can now insert a disentangling unitary U and its adjoint. This can be used to minimize the error of the next step, where we split theta into W\_1^\prime and W\_2^\prime. The disentangling is crucial for a good tripartite decomposition, so lets talk about it in more detail. 21:30

First a few definitions: The unitary U is contracted with theta, forming a tensor theta tilde. We group the upper and lower legs into this purple leg with bond dimension chi\*D. The tensor is then split along the vertical direction with an SVD into tensors X, S, and Y.

So how do we choose a good disentangling unitary? The idea is to choose U such that it minimizes a given cost function. We looked at two cost functions. The first one is simply the truncation error of truncating the last SVD to a bond dimension of chi. The second cost function is the Renyi-entropy. Its given by 1/(1- a parameter alpha) times the logarithm of the trace of the density matrix to the power of alpha. The density matrix is obtained by tracing out one side of the theta tilde tensor. Therefore, we can write this Trace of rho to the power of alpha as a sum over the singular values to the power of two alpha. 22:30

If we now set alpha = 2, the renyi entropy takes on a particularly simple form. Its then just given as – the trace of rho^2. The trace of rho^2 can actually be easily written as this tensor network. And what we can now do, we can again use the Evenbly-Vidal algorithm to maximize this trace, which is equivalent to minimizing the renyi-entropy. So we fix all tensors except for one U, and then the problem permits a closed form solution. We then update U and repeat this procedure. Contracting this network which must be done once per iteration, scales as D^9. 23:20

This seems quite expensive, but as it turns out, this algorithm converges really fast in practice. Here we try it out on the same environment of tensors that we used for the iterative YB-move, and as you can see here this converges after roughly 20 iterations, where the other algorithm wasn’t converged even at 10’000 iterations. 23:40

Ok, so this was for the renyi-entropy with alpha = 2, but what if we want to use a different alpha or use the truncation error directly as cost function? In that case, the cost function cannot easily be written as a tensor network and we cannot use the same optimization method. However, there exists a really powerful framework called Riemannian optimization of matrix manifolds.

We want to optimize over the set of all unitary matrices of some dimensions. More generally, the set of all complex n times m isometries is called the Stiefel manifold, or St(n, m) for short. It is a submanifold of the Euclidean vector space of complex n times m matrices. The idea of Riemannian optimization is to optimize a cost function while staying on this manifold. Lets say our current best guess for the unitary is given by some point on the manifold, lets call it U\_k. One can now compute a update direction xi\_k which has to be projected to the tangent space of U\_k which we draw here in green. This update vector can for example simply be the negative gradient of the cost function. We want to now move in the update direction, but if we do that, we would leave the manifold. To solve this problem one introduces a so-called retraction R, which we draw here in gray, which can be thought of as moving in the direction xi\_k while staying on the manifold. We can then move along this path to obtain a new iterate U\_{k+1} 25:20

For the cost functions we discussed the gradient can be computed analytically, here is for example the gradient of the truncation error cost function. Computing the gradient also costs D^9. I implemented two Riemannian optimization algorithms, namely Conjugate Gradients and the Trust-Region method. Conjugate gradients uses the search directions of previous iterates to try to obain a new conjugate search direction, which performs a lot better than simple gradient descent. The trust-region method is a robust quasi-Newton method. For the trust region method we also need to compute the hessian vector product for second order information about the cost function, but for the cost functions discussed this is also possible to compute analytically. 26:10

Both of these methods are computationally very expensive, and the problem lies in the SVD we have to do, and in contractions involving X, S, and Y because of the large bond dimension. What we found is that if we instead only compute an approximate SVD we can achieve similar results at much lower costs.

We can perform this approximate SVD as follows: First, we split theta tilde into Q and R via an approximate QR-decomposition. This approximate QR-decomposition can be performed by again using an Evenbly-Vidal style optimization algorithm, computing this overlap and alternatingly optimizing Q and R. When this is converged, we then split the tensor R via a normal SVD. We can then absorb this tensor A into Q, which finalizes the decomposition. This whole procedure has a cost of D^7 times the number of iterations. This iteration here converges very quickly in practice. Especially because for example from one conjugate gradients to the next, the iterate is not expected to change drastically. Therefore, a good initialization for Q and R is just the result from the previous iterate, and in practice, this then takes less than 5 iterations to converge. 27:30

Okay, so with this approximate procedure we can bring down the cost of computing the gradient from D^9 to D^8 + N\_svd D^7. Here in this plot, I show the truncation error cost function plotted against the time. As you can see, the approximate CG doesn’t quite reach as low of an error as the exact version but is a lot faster. For the trust-region method, also the approximate version is much faster, for both methods it is roughly an order of magnitude. 28:10

I talked a lot about these two algorithms, let us now actually compare them. Here I plot the YB-error against the time. The dashed line is just a simple SVD with a smart initial guess for the disentangling unitary that can be computed by isometrizing the theta tensor. In blue we show the iterative algorithm with the slow convergence. The orange line is the Renyi-2 disentangler, that converges very quickly but doesn’t reach a very small error. In green we show the Riemannian truncation error disentangler, using the Trust Region method with approximate gradients. This performs the best out of the methods tested. 28:50

We have now discussed a lot how the orthogonality surface can be moved throughout the lattice, but I want to introduce one more algorithm, which is the generalization of TEBD. First of all, lets look at how a local bond operator can be applied to a YB-isoTPS. We want to apply the bond operator to the tensors T\_i and T\_j, giving an updated state psi^prime. In this diagram I have marked an environment of five tensors around T\_i and T\_j with only incoming arrows. We can now again write this as an optimization problem, optimizing this overlap. Same as we did before, this overlap again reduces to a contraction only over this environment, with now the bond operator sandwiched in between. To solve this, we again use the Evenbly-Vidal style algorithm, iteratively optimizing the bottom five tensors. Because the time step Delta t is small, this operator here is close to identity. Therefore, a good initialization for the updated tensors are just the original tensors, and the algorithm converges very quickly after only a few iterations. 30:00

To globally apply now a TEBD update of first order, we start in the left-most column and apply all bond operators that act on this column. We then move two columns to the right and again apply all bond operators. We repeat this until the right-most column is reached. On the way back, we then apply the bond operators on all odd columns, concluding the first order update. We can very easily improve this update to second order by symmetrizing the first-order Trotter decomposition and applying the operators along a chain. This then looks like this: We again start at the left-most column, but now apply the operators in a chain. We move one column to the right and again apply along a chain. We repeat this until we hit the right edge. On the way back, we again apply operators on each column, but in the reverse order. Arriving back at the left edge, we have then applied a second order TEBD step with the error scaling as the third power of the time step. 31:00

In order to benchmark this method we looked at the Transverse Field Ising model. The model is a spin-1/2 lattice model. The Hamiltonian is made up of two terms, one interaction term and a transverse field in z-directoin. At zero temperature, the model exhibits a quantum phase transition, which happens at a critical field of roughly 3.04 on the 2D square lattice.

We start by using imaginary time evolution to find the ground state on the models. The error of TEBD is made up of three parts: The error from applying the local bond operators, the trotterization error, and the error of performing the YB move. The first two errors get smaller when choosing a smaller time step, but the YB move error is not directly affected by the time step. However, for smaller time steps we need to compute more TEBD steps for reaching the same time, and this also means more YB errors. Therefore it is clear that there exists some sweet spot, where the two parts of the errors are roughly the same, and this is the regime where the overall error will be minimized. 32:00

This simulation was performed on a 4x4 diagonal square lattice with 32 spins. We look at 4 different bond dimensions, 2, 4, and 6. First, lets compare the different methods for the YB move. In the left-most figure, a simple SVD was used. We can already see, that there exists a minimum error for some optimal time step Delta t. In the second figure we use the iterative Evenbly-Vidal style algorithm, which performs about the same. The reason for this is its slow convergence. Minimizing the Renyi-2 entropy works really well compared to the other algorithms, and on top of that is very fast. For this reason, we use this disentangling as an initialization for the following methods.

Interestingly, choosing the truncation error as a cost function performs a bit worse than the Renyi-2 entropy. The reason for this is that while the error of a single YB move is smaller, the error of moving a complete column is actually larger when directly disentangling the truncation error. I think this is because the entropy is a more physical cost function, which is able to represent the full state better. Notably the approximate TRM perform almost as good as the exact TRM while being much faster. 33:15

In our testing, the best method was found to be the Riemannian optimization of the Renyi-entropy with alpha = ½, which reaches relative energy errors of well below 10^{-5}. In the following, we will therefore use the approximate TRM, optimizing the Renyi entropy with alpha = 0.5. Again we see that the approximate version of the algorithm basically performs as good as the exact version.

Here I plot the time spent doing YB-moves during one TEBD iteration. What I want you to take away here is the comparison of the exact and approximate Riemannian optimization algorithms. I plot here the approximate versions in light blue, directly next to the respective exact versions. As you can see, they generally are half an order of magnitude up to an order of magnitude faster.

34:10

Its also interesting to compare TEBD of first order to TEBD of second order. As you can see, TEBD2 can obtain an error that is about an order of magnitude smaller. Note the x-axis: we can go to much larger time steps when using TEBD2 because of the smaller trotter error. This way, we perform less YB moves per unit time, and this leads to a lower error.

Let us now go to larger lattices. When increasing the system size from a 4x4 lattice to a 7x7 lattice, the error becomes larger, but seems to converge to a constant relative error when going to larger and larger systems. 34:50

To investigate this further, we performed a ground state search up to a system size of 20x20. First, lets look at the DMRG reference simulation. We plot the energy density against the linear system size. We would expect this energy density to approach a constant for infinite system size, since the finite size effects become less and less important. However, as you can see, the reference simulation diverges at some point. This happens earlier for smaller maximum bond dimensions chi. Here the black line is obtained by extrapolating the bond dimension to infinity. In contrast, the isoTPS simulation doesn’t show this divergence and really seems to approach a constant energy density. This is numerical evidence that the ansatz is able to correctly represent the area law entanglement structure of the TFI model ground state! In the zoomed in plot we can see that the highest bond dimension DMRG simulation in blue was still able to achive better ground state energy even on a 20x20 lattice, but this is expected to change when going to even larger systems or to more complicated models. 36:00

Finally, we also performed real-time evolution after a global quench. For this, we initialized the system with all spins pointing up on an 8x8 square lattice, and then performed a time evolution. Here we plot the sigma\_z expectation value of a spin in the center of the lattice against time. This simulation was performed at the critical field, and we therefore expect the entanglement to grow very quickly with time, making this a hard problem. As you can see here, the method still has problems with real-time evolution, diverging really quickly. The error of the YB move is the main problem. As a potential solution, one could variationally optimize the tensors after shifting a column completely, this was also done on the original isoTPS after the MM and found to be especially important when doing real-time evolution. Alternatively, one could try to find a better procedure for performing the YB move. 37:00

This brings me to a short summary of my talk. We have introduced a new canonical form for isoTPS. The YB move is a bit more expensive than the MM, scaling as D^8 instead of D^7. Nevertheless, it is interesting to discuss different canonical forms. One advantage of the YB-isoTPS is that it is very easily generalizable to other lattices. For example, I also generalized it to the honeycomb lattice as shown here, and with this we were also able to find ground states of the TFI model. Also, more involve algorithms like DMRG anyways scale with high powers like D^12, making it less important if moving the ortho surface scales with D^9 or D^8.

As an outlook: It would be interesting to implement a DMRG like energy minimization algorithm and compare it to the original isoTPS. Also it would be nice to generalize this method to further lattices, for example the Kagome lattice. Finally, the algorithm would greatly benefit from porting it to Graphics Processing Units (GPUs), as they can significantly speed up tensor contractions and decompositions, while also increasing the power efficiency.

This concludes my talk, id be happy to answer some questions! 38:15