The Illustrated AlphaFold

Elana Simon, Jake Silberg

Background and Existing Work

Protein Folding Challenge

- Proteins are linear chains of amino acids
- Amino acids interact with each other and naturally folds into lowest-energy 3D structure
- This structure determines protein's function
- Applications: drug design, disease research, protein engineering

Existing Work - AlphaFold 1 (2018):

- DeepMind's first attempt at the protein folding problem
- Built on previous work using evolutionary data to predict protein structure
- Used deep learning to predict distances between amino acid and construct protein structures.
- Significantly outperformed other methods at the time, but still not at experimental accuracy levels

Existing Work - AlphaFold 2 (2020):

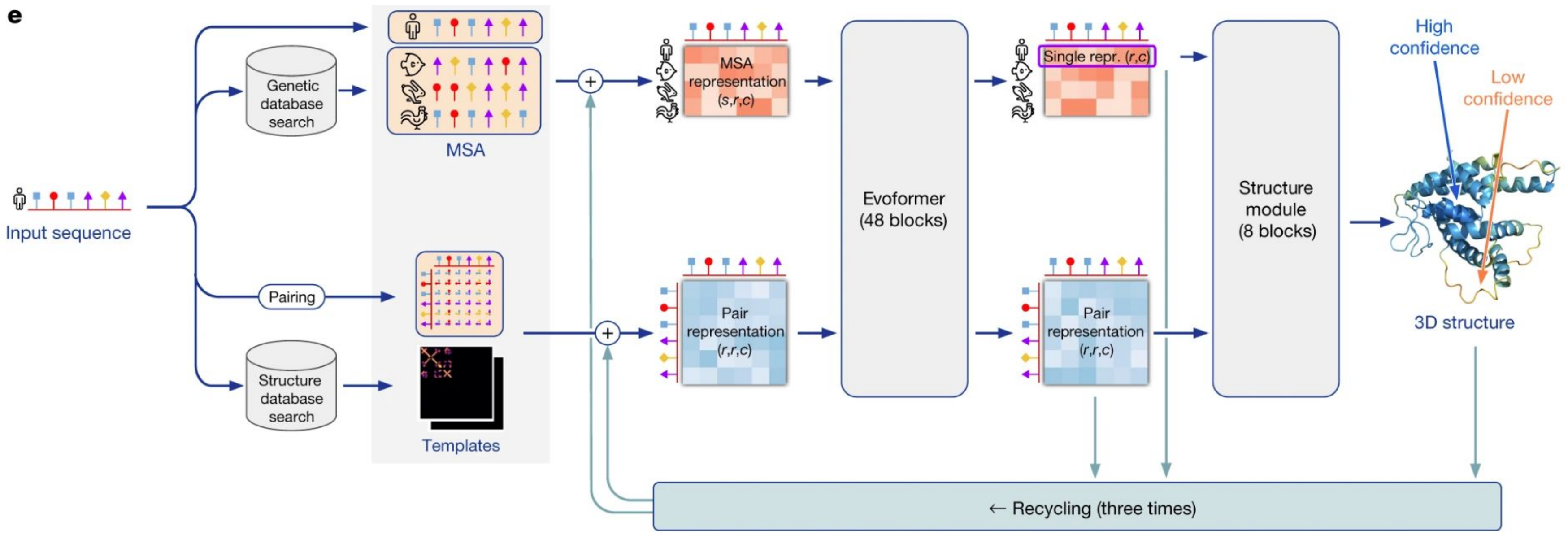
Machine Learning Framework:

- Utilized an attention-based neural network architecture called Evoformer
- Incorporated a multiple sequence alignment (MSA) and pair representation

Performance:

- Used pattern recognition for structure prediction, with minimal physics-based refinement
- Achieved breakthrough accuracy in protein structure prediction
- Predicted structures for nearly all cataloged proteins known to science

Existing Work - AlphaFold 2 (2020):


e
Input sequence
Genetic database search
MSA
Pairing
Structure database search
Templates
MSA representation (s,r,c)
Pair representation (r,r,c)
Evoformer (48 blocks)
Single repr. (r,c)
Pair representation (r,r,c)
Structure module (8 blocks)
High confidence
Low confidence
3D structure
← Recycling (three times)

Molecular Structure Prediction

Limitation of AlphaFold 2:

- Focused on single protein structures
- Input: Amino acid sequences only
- Limited to standard amino acids

Real Biology is Complex:

- Proteins rarely work alone
- Drug design needs protein-ligand interactions
- DNA/RNA interactions crucial for gene regulation

AlphaFold 3 (2024):

Function:

- Expanded beyond proteins to predict structures and interactions of various biomolecules
- Can model proteins, DNA, RNA, ligands, and their interactions
- Predicts chemical modifications that control cell functioning

Machine Learning Framework:

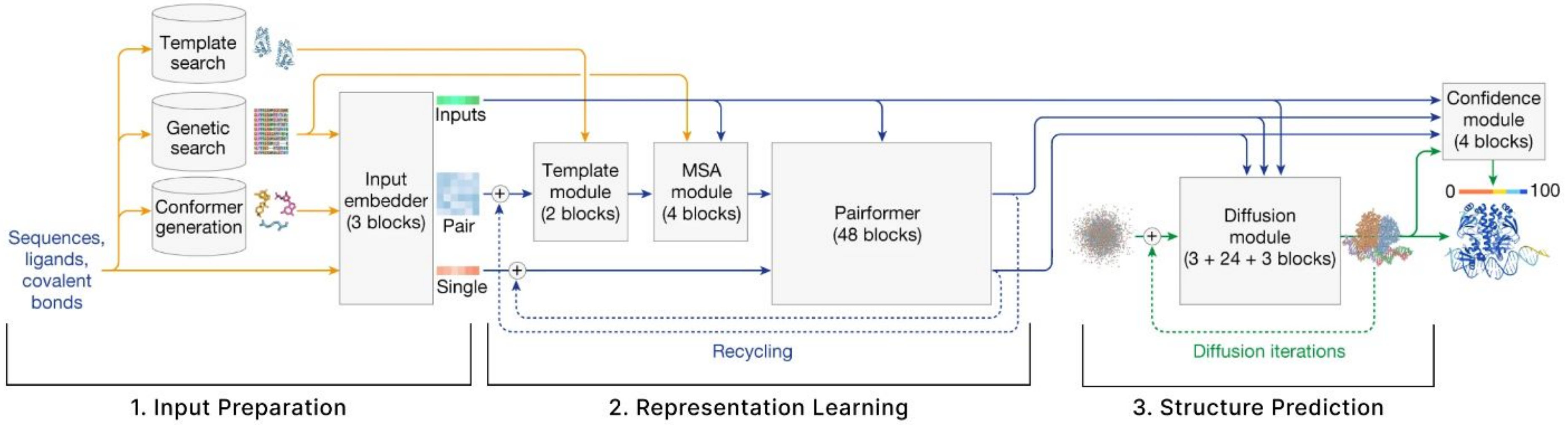
- Enhanced version of the Evoformer module
- Introduced a diffusion network for structure assembly

Performance:

- Improved accuracy in predicting molecular interactions by at least 50% compared to existing methods

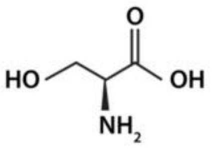
Key Idea & Designs

Architecture Overview



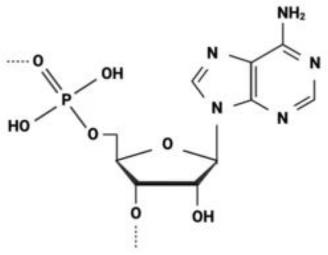
Input Preparation - Two key representations

Standard Amino Acid
(serine)



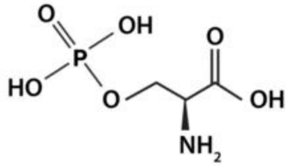
7 atoms*
1 tokens

Standard Nucleotide
(adenosine)



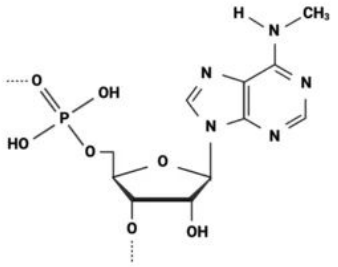
23 atoms*
1 token

Modified Amino Acid
(phosphoserine)



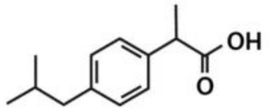
11 atoms*
11 tokens

Modified Nucleotide
(methyladenosine)



24 atoms*
24 tokens

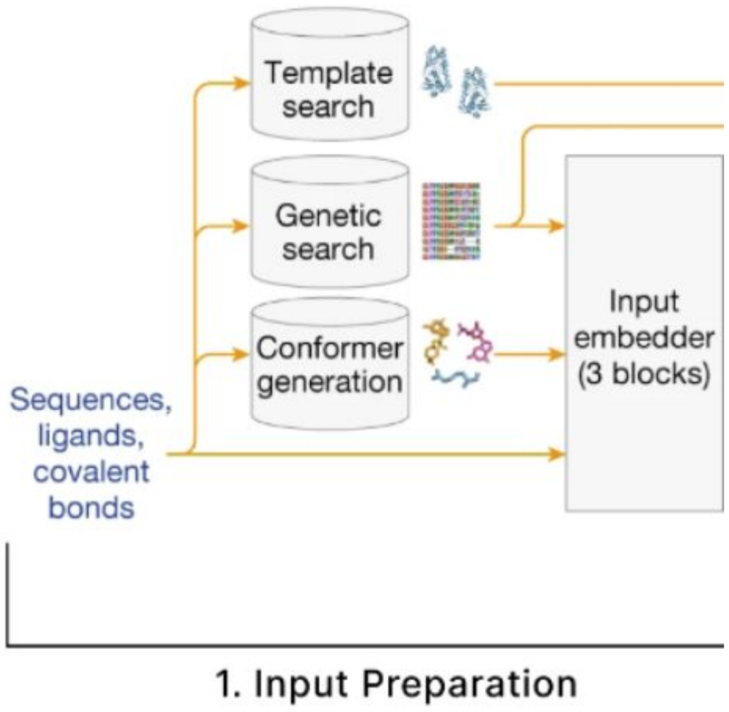
Ligand
(ibuprofen)



15 atoms*
15 tokens

*atoms=heavy atoms

Input Preparation



Genetic Search: finds similar molecular sequence across different species, get Multiple Sequence Alignment (**MSA**) (N_MSA*N_token*C_m)

Template Search: finds similar known structures, get template matrix **t** (N_token*N_token*C_t*N_templates)

Conformers : local chemical arrangements (initial atom positions, chemical properties), get original atom-level representation matrix **c** (C_atom*N_atom)

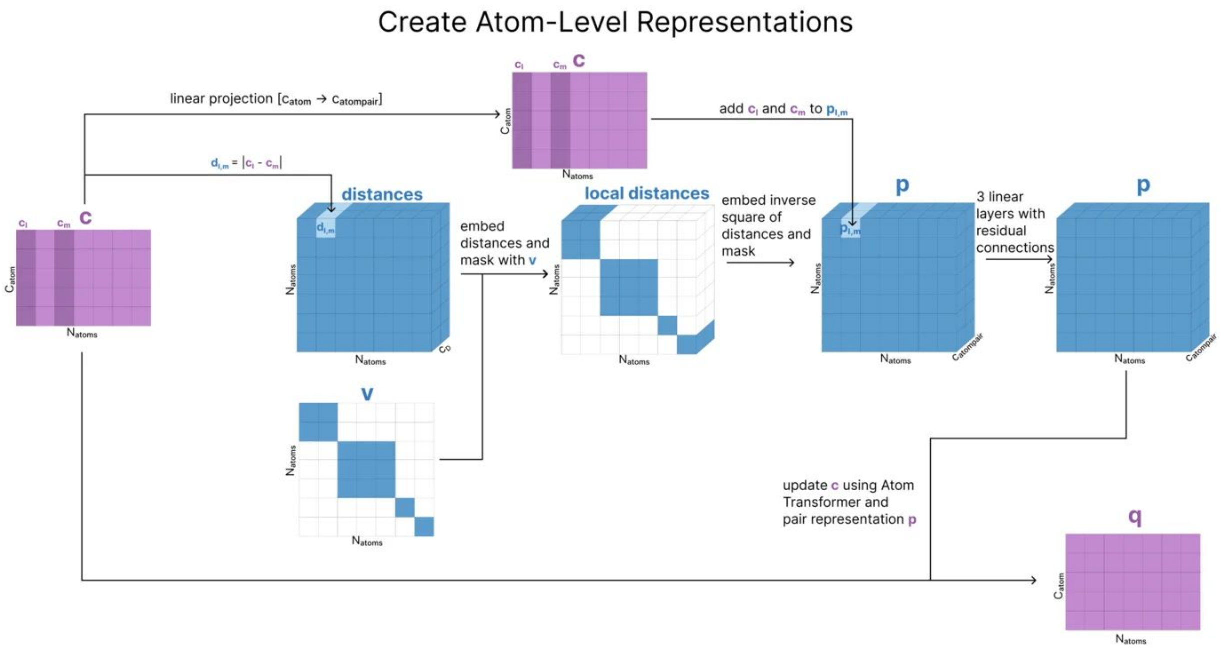
Create Atom-Level Representations

Single Representation (c → q):

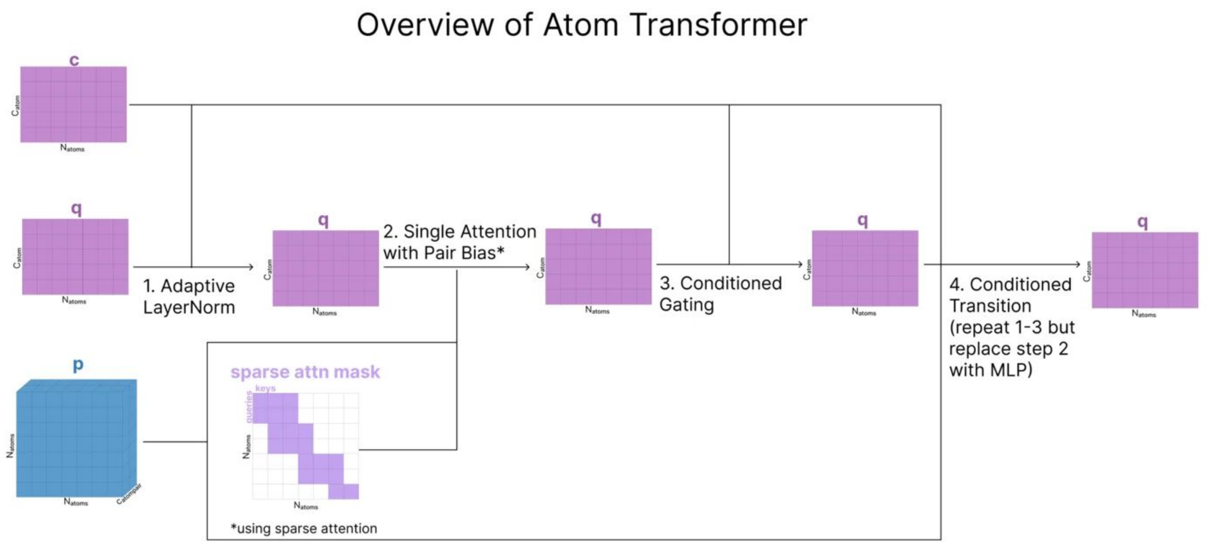
Matrix c: stores atom properties

Matrix q: copy of c that will be updated during processing

Pair Representation (p): Stores distances between atoms within each token


Create Atom-Level Representations
linear projection [$c_{atom} \rightarrow c_{atompair}$]
$d_{l,m} = |c_l - c_m|$
c_l c_m c
C_{atom}
N_{atoms}
distances
$d_{l,m}$
c_D
embed distances and mask with v
v
local distances
embed inverse square of distances and mask
add c_l and c_m to $p_{l,m}$
p
$p_{l,m}$
$c_{atompair}$
3 linear layers with residual connections
update c using Atom Transformer and pair representation p
q

Update Atom-Level Representations (Atom Transformer)



Adaptive LayerNorm: uses input c to dynamically generate normalization parameters for q

Attention with Pair Bias: enhances standard self-attention by using pair representation as bias

Conditioned Gating: Controls information flow using gates generated from original atom representation c

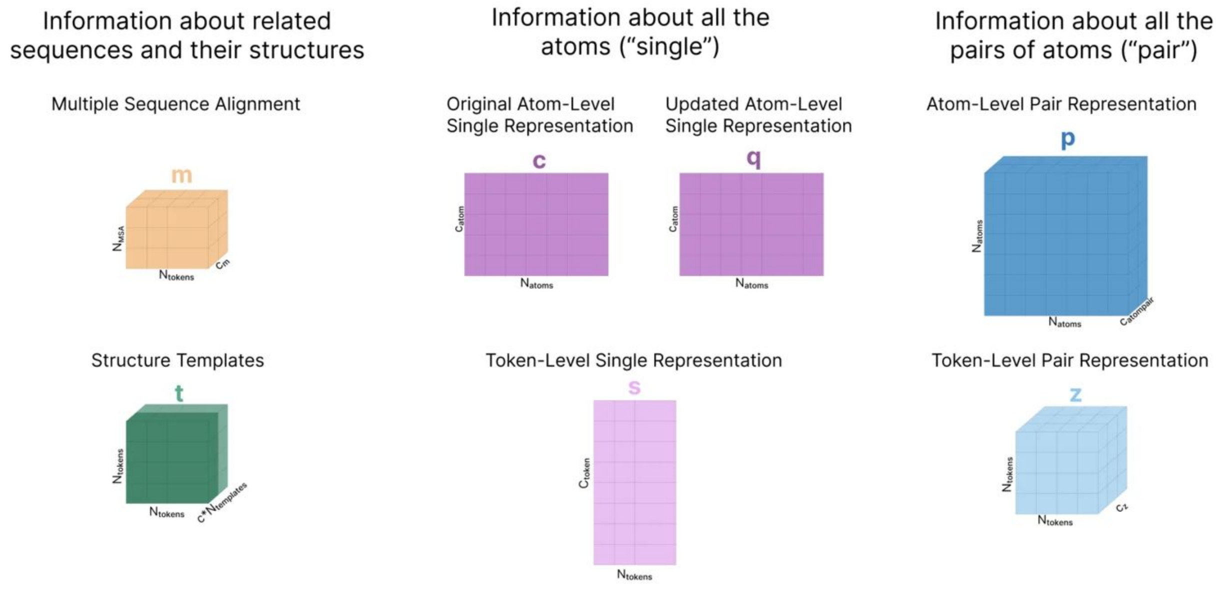
Conditioned Transition: Modified MLP layer using SwiGLU activation, sandwiched between AdaNorm and gating, both conditioned on c

Aggregate Atom-Level → Token-Level

Token-Level Single Representation (s): atoms (q) → projection → averaging → feature addition → projection → s_init (s)

Token-Level Pair Representation (z): s_init (s) → projection (c_token → c_z) → z_ij = si, sj→ add r.p.e → add bonds → z_init

Set aside the **atom-level** representations (**c, q, p**) and focus on updating our token-level representations **s** and **z** in the next section (with the help of m and t).

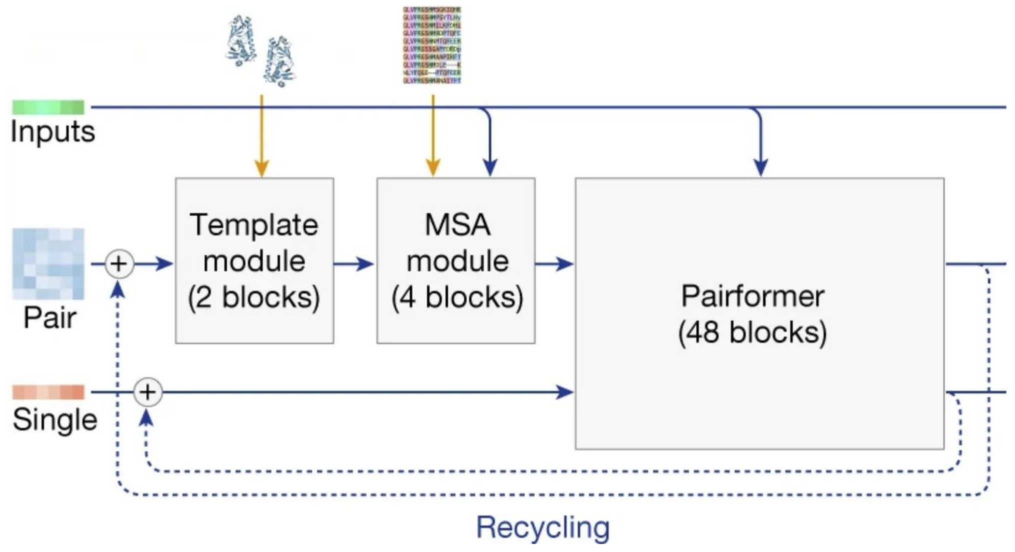

Information about related sequences and their structures
Multiple Sequence Alignment
m
N_{MSA}
N_{tokens}
c_m
Structure Templates
t
N_{tokens}
N_{tokens}
$c{*}N_{templates}$
Information about all the atoms ("single")
Original Atom-Level Single Representation
c
c_{atom}
N_{atoms}
Updated Atom-Level Single Representation
q
c_{atom}
N_{atoms}
Token-Level Single Representation
s
c_{token}
N_{tokens}
Information about all the pairs of atoms ("pair")
Atom-Level Pair Representation
p
N_{atoms}
N_{atoms}
$c_{atompair}$
Token-Level Pair Representation
z
N_{tokens}
N_{tokens}
c_z

Representation Learning

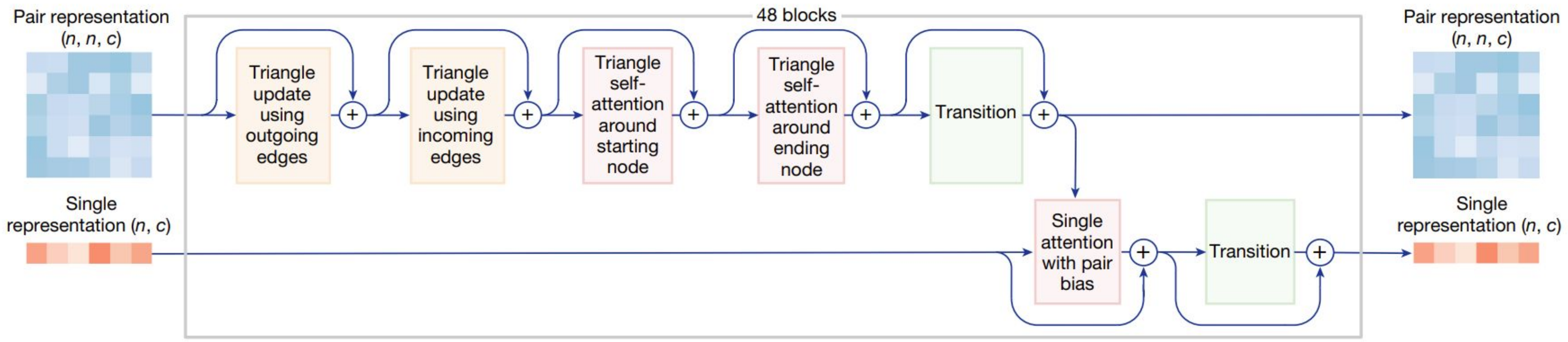
Template module: updates **z** using the structure templates **t**

MSA module: first updates the MSA using input token level single representation (**q**)

- adds **MSA** to update token-level pair representation (**z**) using Outer Product Mean
- updates the **MSA** based on **z** with a simplified version of self attention with pair bias



Pairformer Module


Pair representation (n, n, c)
48 blocks
Triangle update using outgoing edges
Triangle update using incoming edges
Triangle self-attention around starting node
Triangle self-attention around ending node
Transition
Pair representation (n, n, c)
Single representation (n, c)
Single attention with pair bias
Transition
Single representation (n, c)

Updates s and z with geometry-inspired (triangle) attention

Why Look at Triangles

Geometric Principle

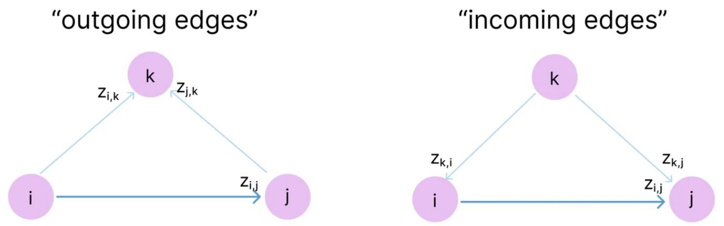
- Based on triangle inequality: sum of any two sides > third side
- Helps constrain predictions using geometric relationships

Implementation in AF3

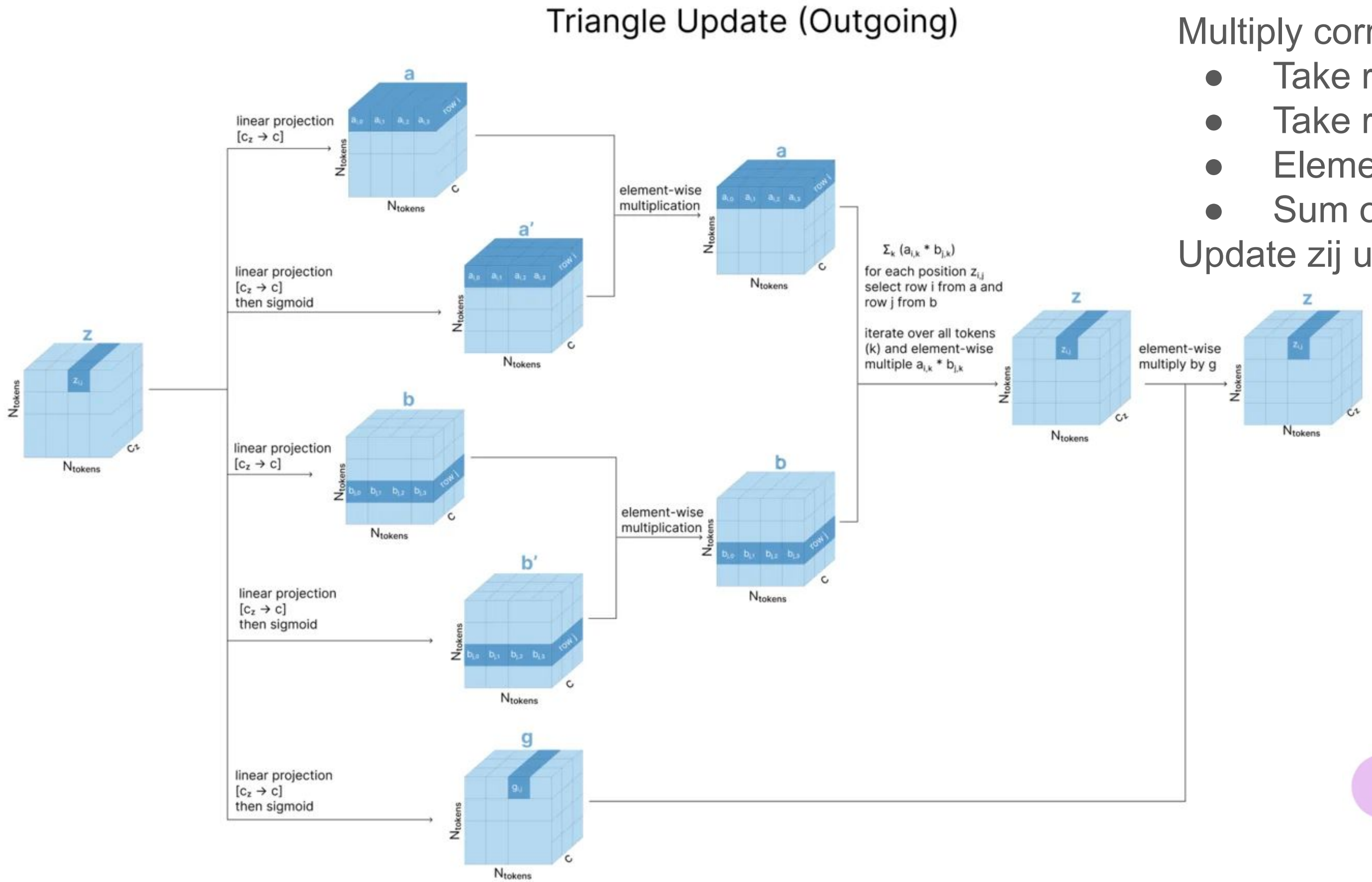
- Each pair relationship (z_ij) is updated using all possible third points (k)

Directional Relationships

- Two types of paths considered: outgoing edges and incoming edges



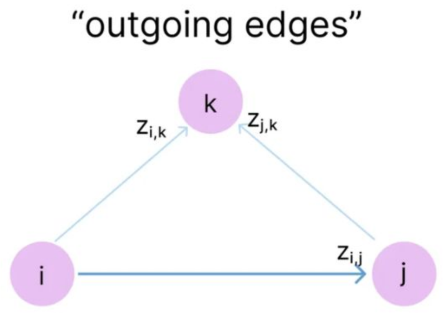
Triangle Updates - Outgoing


Triangle Update (Outgoing)
z
linear projection [c_z → c]
a
linear projection [c_z → c] then sigmoid
a′
linear projection [c_z → c]
b
linear projection [c_z → c] then sigmoid
b′
linear projection [c_z → c] then sigmoid
g
element-wise multiplication
element-wise multiplication
Σ_k (a_i,k * b_j,k)
for each position z_i,j select row i from a and row j from b
iterate over all tokens (k) and element-wise multiple a_i,k * b_j,k
element-wise multiply by g
N_tokens
c
row i
row j

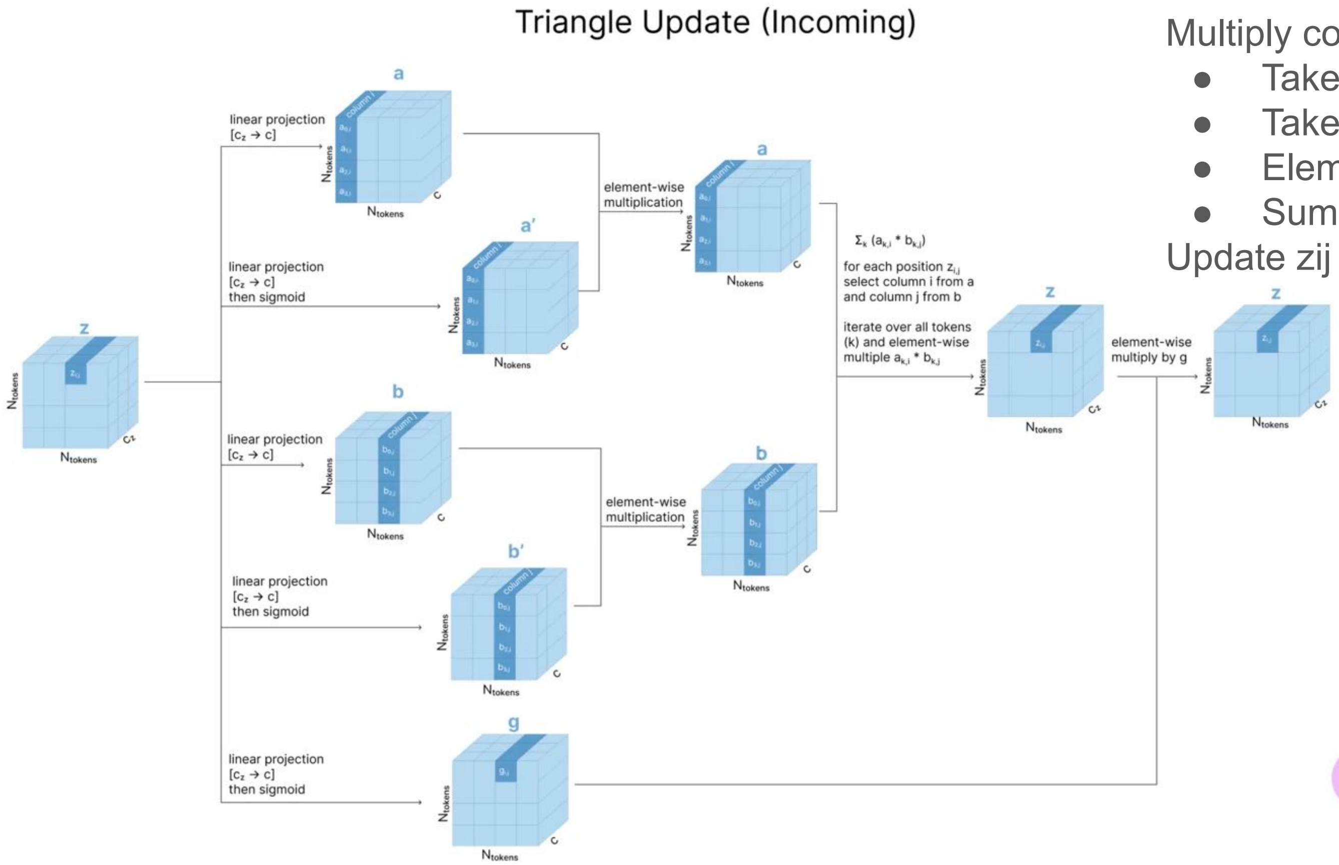
Multiply corresponding elements:

- Take row i from a
- Take row j from b
- Element-wise multiplication
- Sum over k dimension

Update zij using zik and zjk



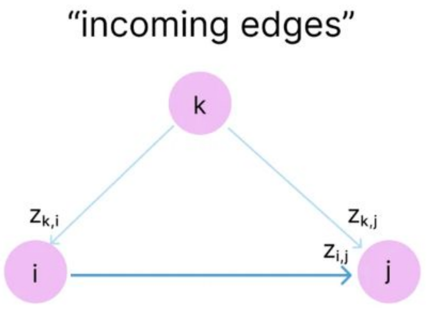
Triangle Updates - Incoming


Triangle Update (Incoming)
z
a
linear projection [$c_z \to c$]
a′
linear projection [$c_z \to c$] then sigmoid
b
linear projection [$c_z \to c$]
b′
linear projection [$c_z \to c$] then sigmoid
g
linear projection [$c_z \to c$] then sigmoid
element-wise multiplication
element-wise multiplication
$\Sigma_k (a_{k,i} * b_{k,j})$
for each position $z_{i,j}$ select column i from a and column j from b
iterate over all tokens (k) and element-wise multiple $a_{k,i} * b_{k,j}$
element-wise multiply by g
N_{tokens}
column i
column j

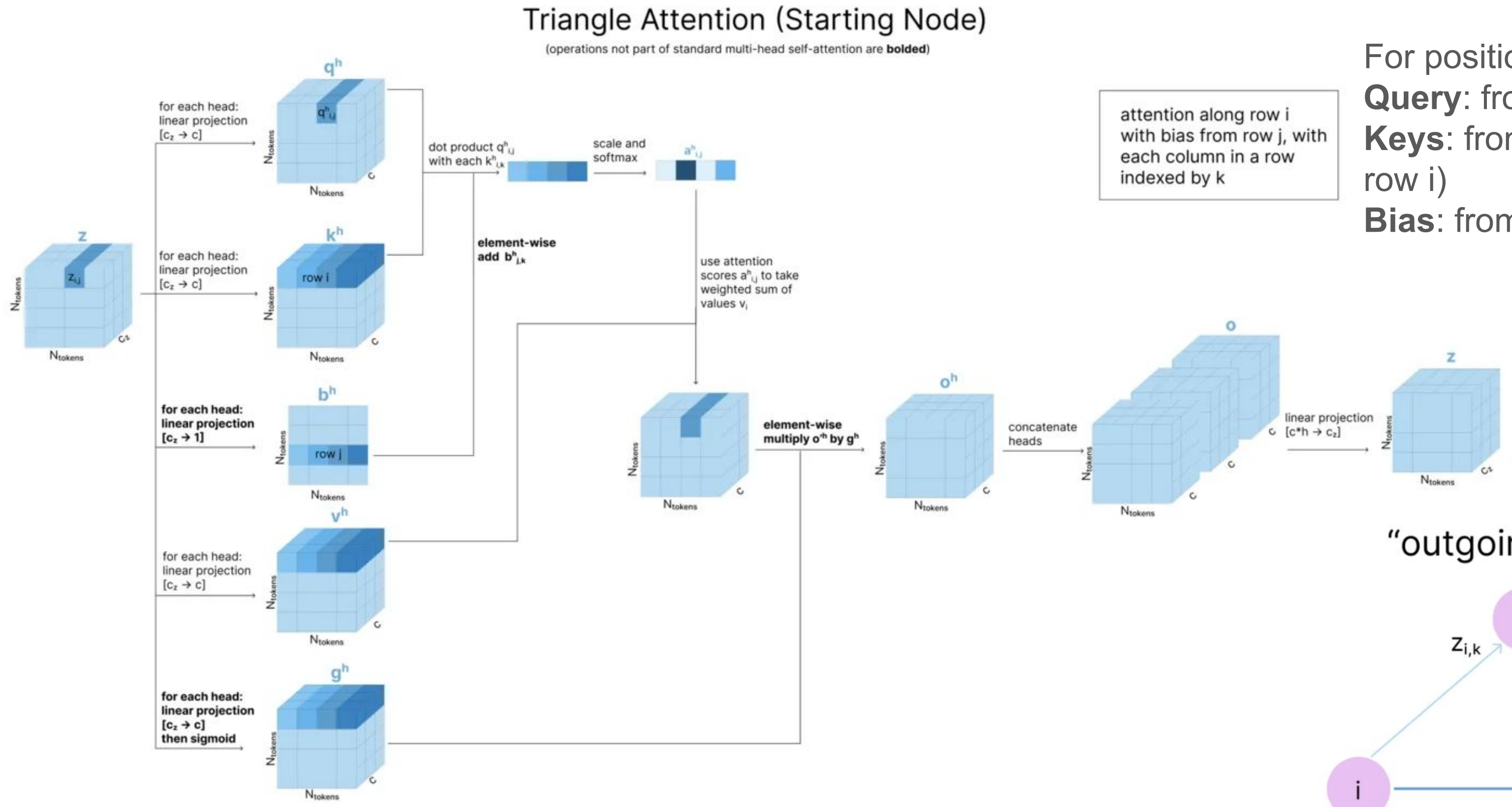
Multiply corresponding elements:

- Take column i from a
- Take column j from b
- Element-wise multiplication
- Sum over k dimension

Update zij using zki and zkj



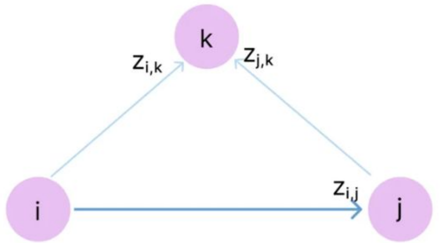
Triangle Attention (Starting Node)


Triangle Attention (Starting Node)
(operations not part of standard multi-head self-attention are **bolded**)
z
$z_{i,j}$
N_{tokens}
c_z
for each head: linear projection [$c_z \to c$]
q^h
$q^h_{i,j}$
k^h
row i
for each head: linear projection [$c_z \to c$]
for each head: linear projection [$c_z \to 1$]
b^h
row j
for each head: linear projection [$c_z \to c$]
v^h
for each head: linear projection [$c_z \to c$] then sigmoid
g^h
c
dot product $q^h_{i,j}$ with each $k^h_{i,k}$
element-wise add $b^h_{j,k}$
scale and softmax
$a^h_{i,j}$
use attention scores $a^h_{i,j}$ to take weighted sum of values v_i
element-wise multiply o^h by g^h
o^h
concatenate heads
o
linear projection [$c*h \to c_z$]
z

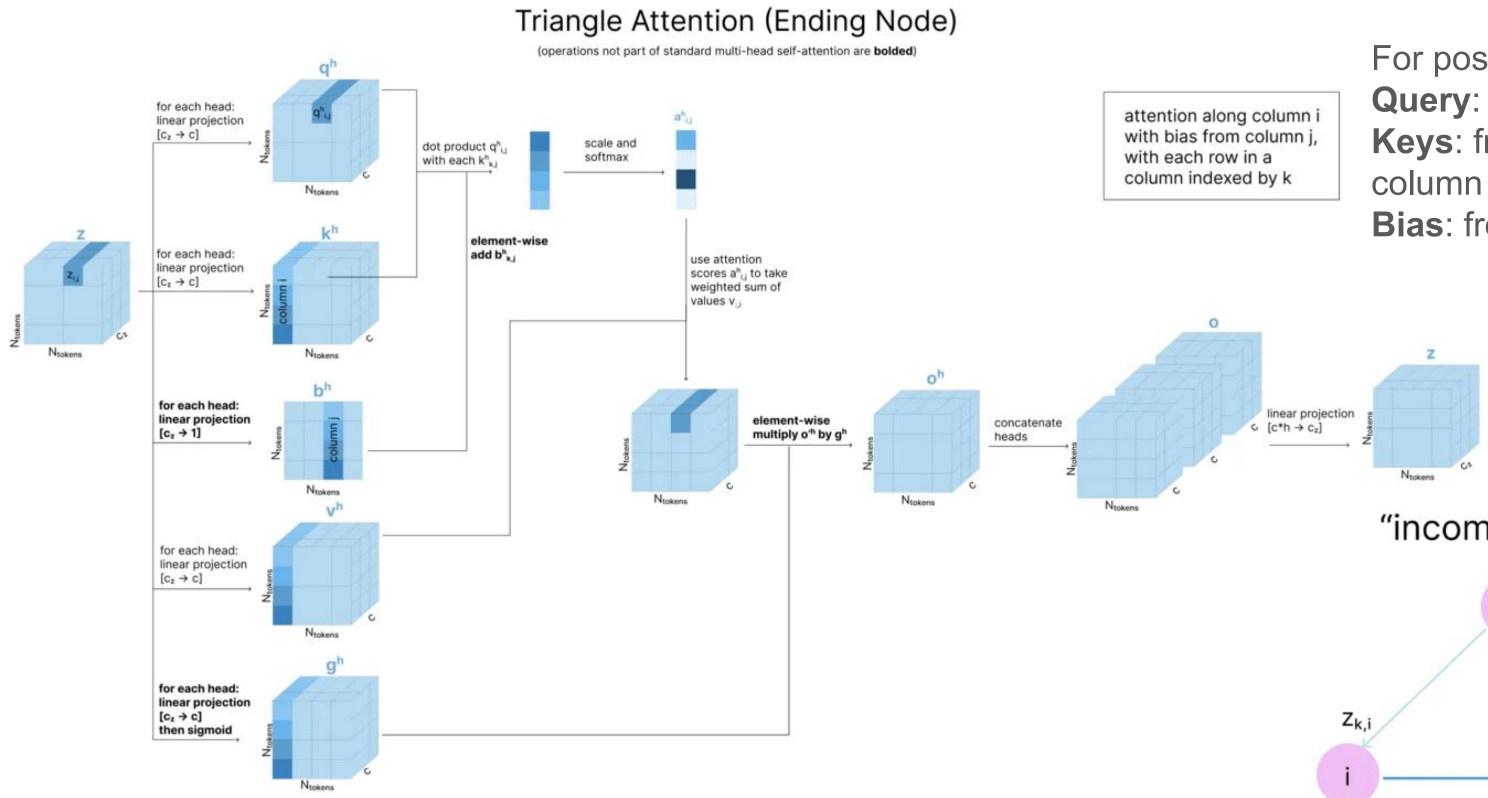
attention along row i with bias from row j, with each column in a row indexed by k

For position zij:
Query: from zij
Keys: from zik (all k in row i)
Bias: from zjk

"outgoing edges"



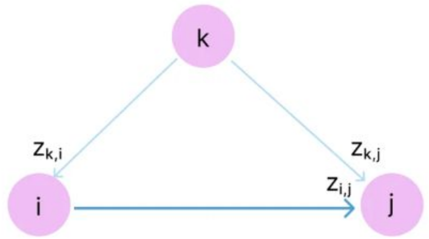
Triangle Attention (Ending Node)


Triangle Attention (Ending Node)
(operations not part of standard multi-head self-attention are **bolded**)
z
$z_{i,j}$
N_{tokens}
c_z
for each head: linear projection [$c_z \to c$]
q^h
$q^h_{i,j}$
k^h
column i
for each head: linear projection [$c_z \to 1$]
b^h
column j
v^h
for each head: linear projection [$c_z \to c$] then sigmoid
g^h
dot product $q^h_{i,j}$ with each $k^h_{k,i}$
element-wise add $b^h_{k,j}$
scale and softmax
$a^h_{i,j}$
use attention scores $a^h_{i,j}$ to take weighted sum of values $v_{:,i}$
element-wise multiply $o^{\prime h}$ by g^h
o^h
concatenate heads
o
linear projection [c*$h \to c_z$]
z
c

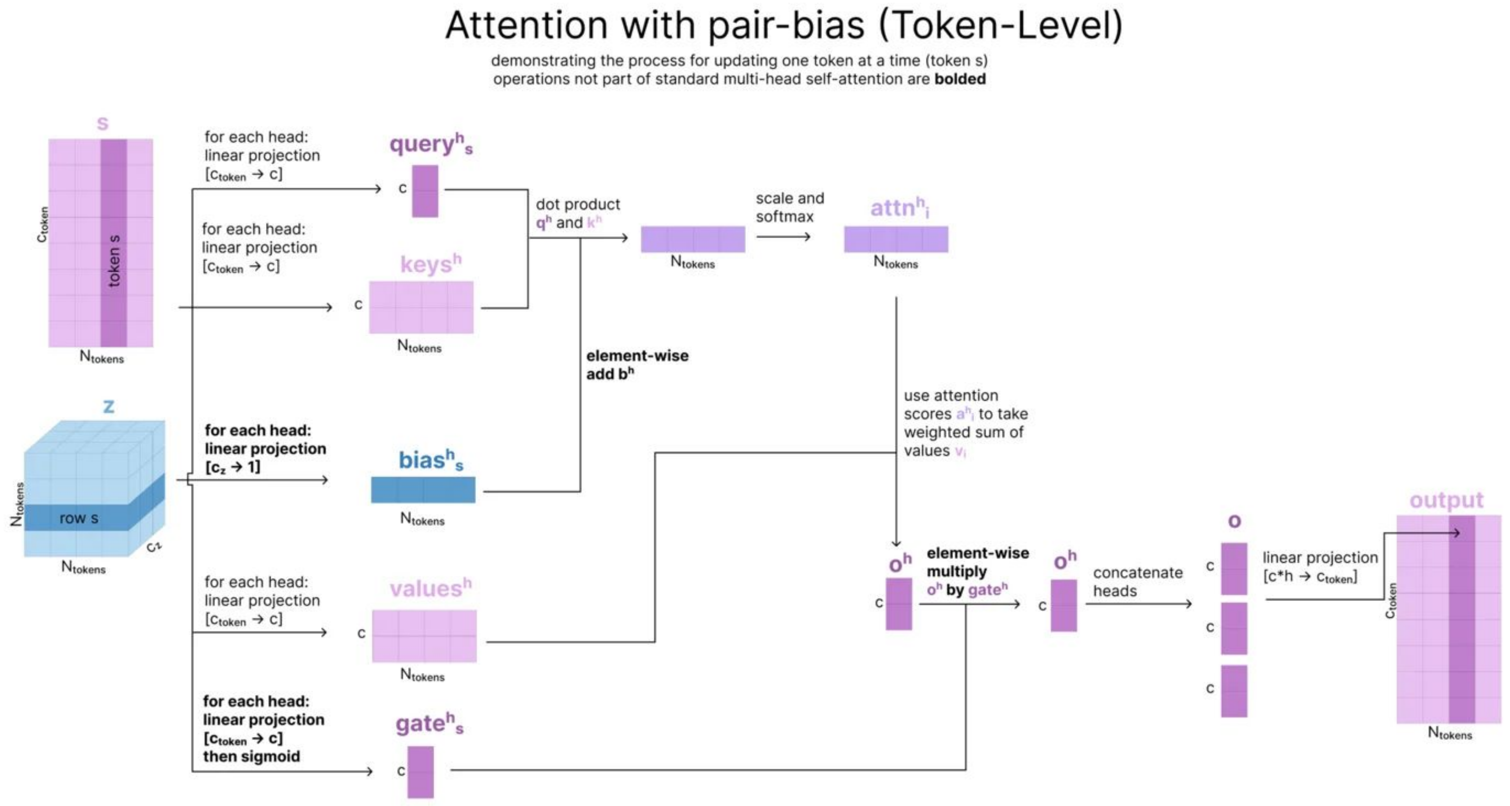
attention along column i with bias from column j, with each row in a column indexed by k

For position zij:
Query: from zij
Keys: from zki (all k in column i)
Bias: from zkj

“incoming edges”



Single Attention with Pair Bias


Attention with pair-bias (Token-Level)
demonstrating the process for updating one token at a time (token s)
operations not part of standard multi-head self-attention are **bolded**
s
c_{token}
token s
N_{tokens}
z
N_{tokens}
row s
c_z
N_{tokens}
for each head: linear projection [$c_{token} \to c$]
query^h_s
c
for each head: linear projection [$c_{token} \to c$]
keys^h
c
N_{tokens}
for each head: linear projection [$c_z \to 1$]
bias^h_s
N_{tokens}
for each head: linear projection [$c_{token} \to c$]
values^h
c
N_{tokens}
for each head: linear projection [$c_{token} \to c$] then sigmoid
gate^h_s
c
dot product q^h and k^h
element-wise add b^h
N_{tokens}
scale and softmax
attn^h_i
N_{tokens}
use attention scores a^h_i to take weighted sum of values v_i
o^h
c
element-wise multiply o^h by gate^h
o^h
c
concatenate heads
o
c
c
c
linear projection [$c*h \to c_{token}$]
output
c_{token}
N_{tokens}

Structure Prediction via Diffusion

Training Phase:

- Start: Real atomic coordinates (xt=0)
- Process: Add noise gradually → xt=T
- Learn: Predict noise added at each step
- Loss: Compare predicted vs actual noise

Inference Phase:

- Start: Random coordinates (xt=T)
- Process: Iteratively remove predicted noise
- End: Final denoised structure (x0)

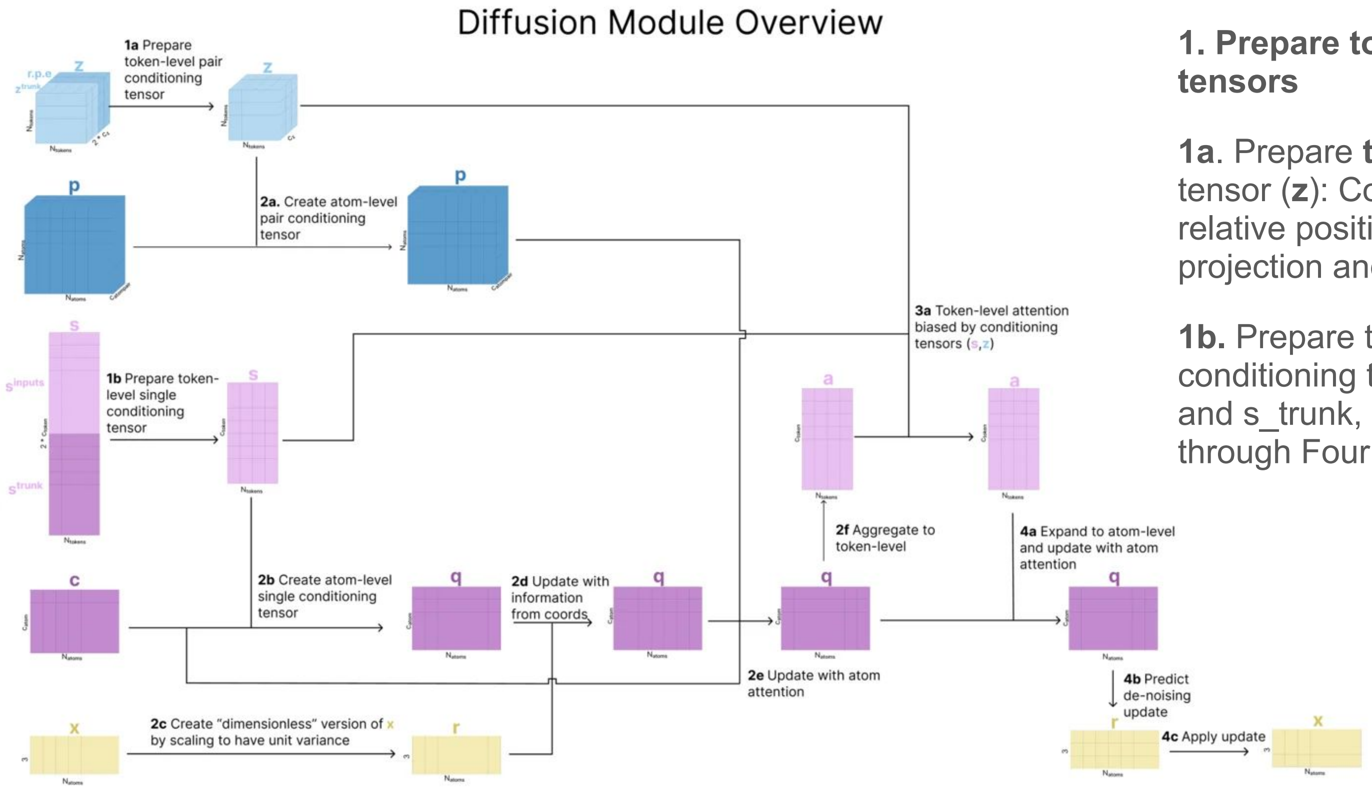
Conditional Diffusion

Conditional Diffusion: final generation matches the information represented by dataset and conditioning input.

AF3 Implementation:

- **data**: a matrix x with the x,y,z coordinates of all the atoms.
- **Training phase**: add noise to matrix x and predict the noises while having input condition
- **Inference phase**:
    - starting with random coordinates
    - first randomly rotate and translate our entire predicted complex.
    - then add a small amount of noise to the coordinates to encourage more heterogeneous generations.
    - Finally, we predict a de-noising step using the Diffusion Module.

Diffusion Module

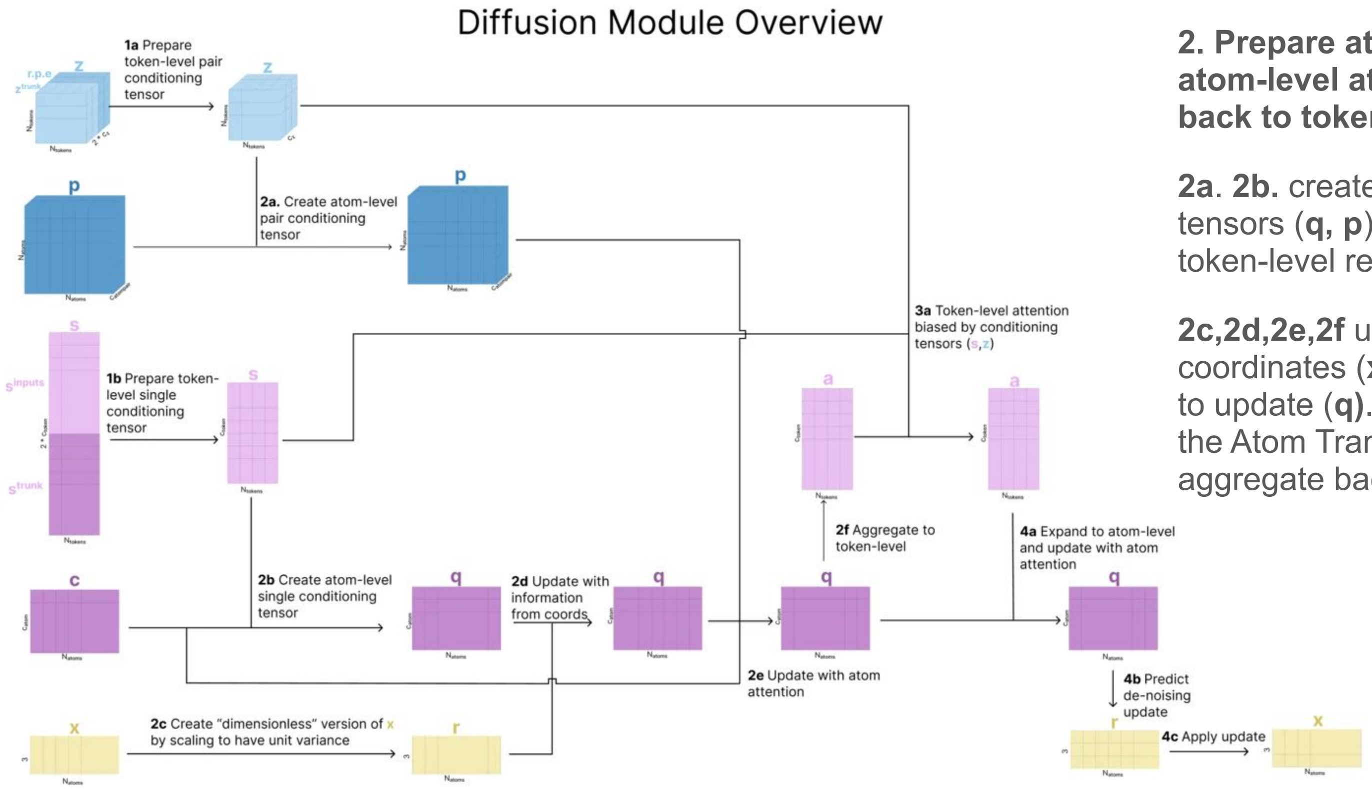

Diffusion Module Overview
r.p.e
z
z trunk
N tokens
2 * C z
1a Prepare token-level pair conditioning tensor
z
C z
p
N atoms
C atompair
2a. Create atom-level pair conditioning tensor
p
s
s inputs
2 * C token
s trunk
1b Prepare token-level single conditioning tensor
s
C token
c
C atom
2b Create atom-level single conditioning tensor
q
2d Update with information from coords
q
2e Update with atom attention
q
2f Aggregate to token-level
a
3a Token-level attention biased by conditioning tensors (s, z)
a
4a Expand to atom-level and update with atom attention
q
4b Predict de-noising update
r
4c Apply update
x
x
3
2c Create "dimensionless" version of x by scaling to have unit variance
r

1. Prepare token-level conditioning tensors

1a. Prepare **token-level** pair conditioning tensor (**z**): Combine z_trunk with the relative positional encodings through projection and transitions

1b. Prepare token-level single conditioning tensor (**s**): Merge s_inputs and s_trunk, add timestep information through Fourier embedding

Diffusion Module



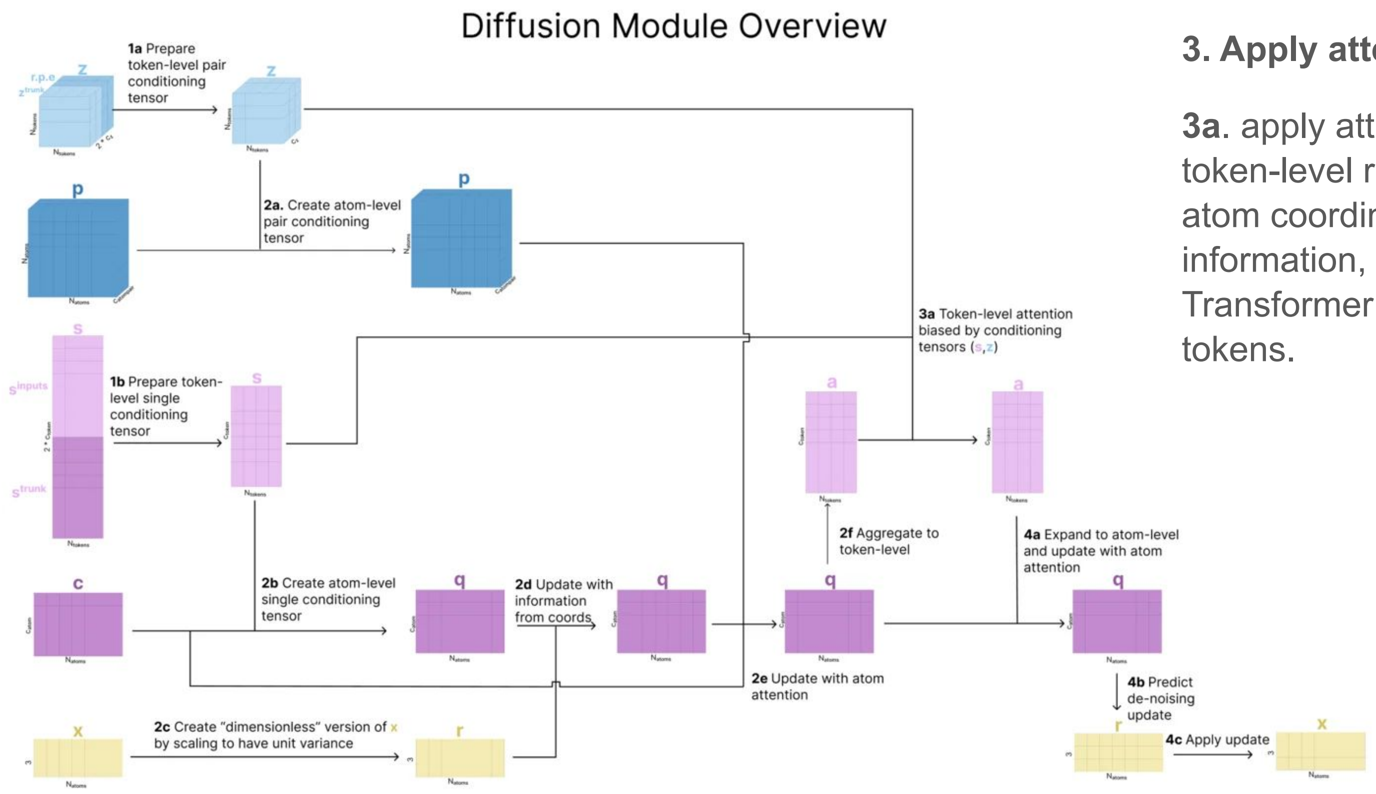
Diffusion Module Overview
r.p.e
z
z^trunk
1a Prepare token-level pair conditioning tensor
z
p
2a. Create atom-level pair conditioning tensor
p
s
s^inputs
s^trunk
1b Prepare token-level single conditioning tensor
s
3a Token-level attention biased by conditioning tensors (s,z)
a
a
2f Aggregate to token-level
4a Expand to atom-level and update with atom attention
c
2b Create atom-level single conditioning tensor
q
2d Update with information from coords
q
q
q
2e Update with atom attention
4b Predict de-noising update
x
2c Create "dimensionless" version of x by scaling to have unit variance
r
r
4c Apply update
x
N_tokens
N_atoms

2. Prepare atom-level tensors, apply atom-level attention, and aggregate back to token-level

2a. **2b.** create atom-level conditioning tensors (**q, p**), based on the current token-level representations (**s, z**)

2c,2d,2e,2f use the atom's current coordinates (**x**) by the variance of the data to update (**q**). Finally, we update (**q**) with the Atom Transformer using (**p**), and aggregate back to tokens level (**a**)

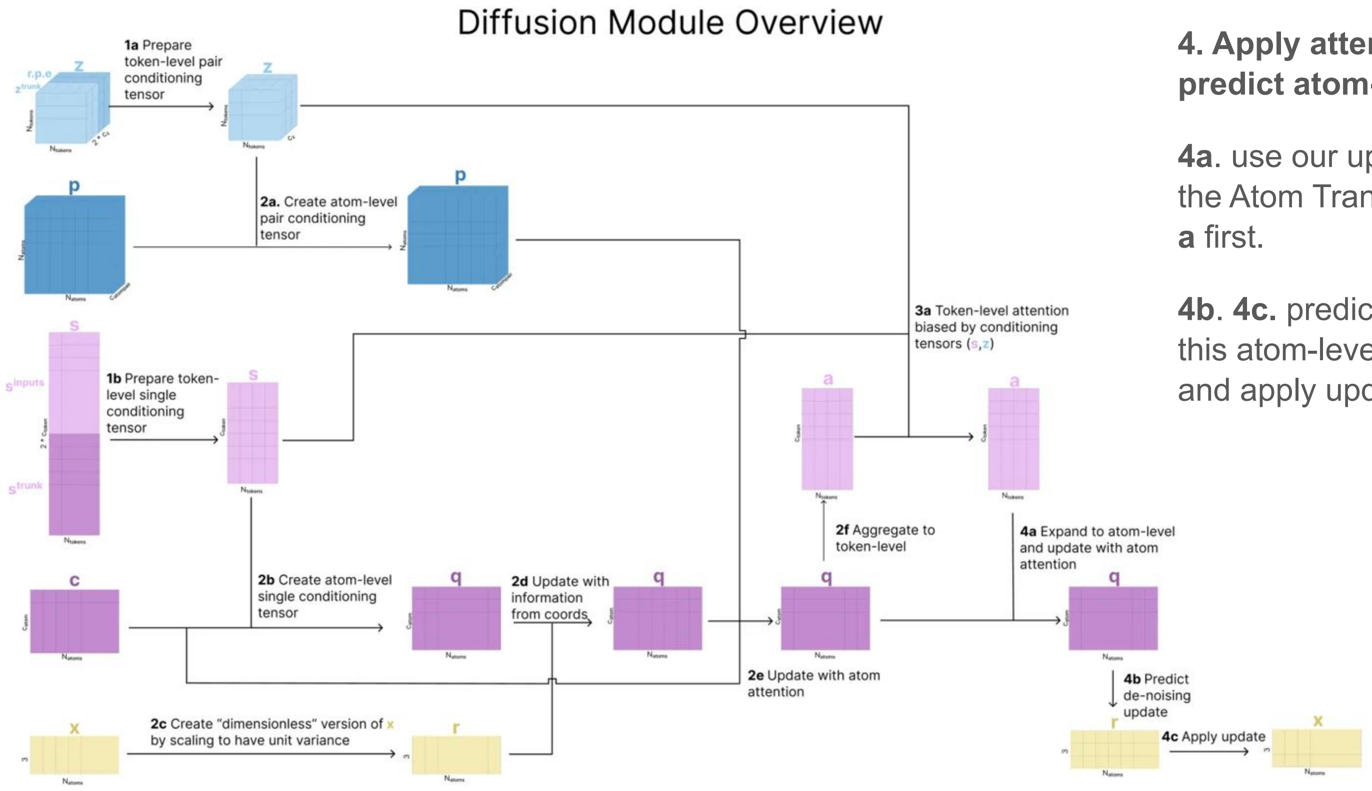
Diffusion Module


Diffusion Module Overview
r.p.e
z
z trunk
1a Prepare token-level pair conditioning tensor
z
p
2a. Create atom-level pair conditioning tensor
p
s
s inputs
s trunk
1b Prepare token-level single conditioning tensor
s
3a Token-level attention biased by conditioning tensors (s,z)
a
a
2f Aggregate to token-level
4a Expand to atom-level and update with atom attention
c
2b Create atom-level single conditioning tensor
q
2d Update with information from coords
q
q
q
2e Update with atom attention
4b Predict de-noising update
x
2c Create "dimensionless" version of x by scaling to have unit variance
r
r
4c Apply update
x

3. Apply attention at the token-level

3a. apply attention to update token-level representation (**a**) of the atom coordinates and sequence information, which mirrors the Atom Transformer at input preparation but for tokens.

Diffusion Module


Diffusion Module Overview
1a Prepare token-level pair conditioning tensor
1b Prepare token-level single conditioning tensor
2a. Create atom-level pair conditioning tensor
2b Create atom-level single conditioning tensor
2c Create "dimensionless" version of x by scaling to have unit variance
2d Update with information from coords
2e Update with atom attention
2f Aggregate to token-level
3a Token-level attention biased by conditioning tensors (s,z)
4a Expand to atom-level and update with atom attention
4b Predict de-noising update
4c Apply update
z
r.p.e
z trunk
p
s
s inputs
s trunk
c
q
a
x
r
N tokens
N atoms

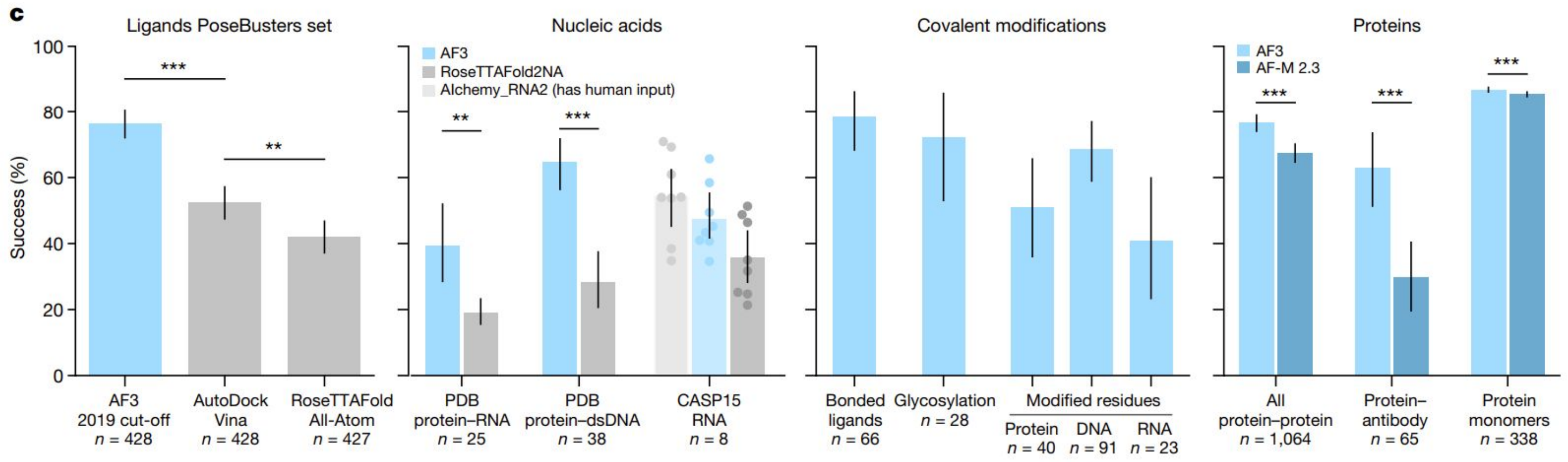
4. Apply attention at the atom-level to predict atom-level noise updates

4a. use our updated **a** to update **q** using the Atom Transformer, by broadcasting our **a** first.

4b. **4c.** predict de-noising update by maps this atom-level representation q back to R3 and apply update to x.

Evaluation

Evaluation



c
Ligands PoseBusters set
Nucleic acids
Covalent modifications
Proteins
Success (%)
0
20
40
60
80
100
***
**
AF3
RoseTTAFold2NA
Alchemy_RNA2 (has human input)
AF-M 2.3
AF3 2019 cut-off n = 428
AutoDock Vina n = 428
RoseTTAFold All-Atom n = 427
PDB protein–RNA n = 25
PDB protein–dsDNA n = 38
CASP15 RNA n = 8
Bonded ligands n = 66
Glycosylation n = 28
Modified residues
Protein n = 40
DNA n = 91
RNA n = 23
All protein–protein n = 1,064
Protein–antibody n = 65
Protein monomers n = 338

Thoughts

ML Musings

AlphaFold as Retrieval-Augmented Generation:

Include retrievals from the training set at inference time, by utilizing an MSA and template search

Large Language Models routinely use Retrieval Augmented Generation systems such as a traditional web search at inference time to orient the model toward relevant information

Pair-Bias Attention

Attention where the queries, keys, and values all originate from the same source (like in self-attention), but there is a bias term added to the attention map from another source

This particular type of cross-biasing is not seen to be used in other fields