Accurate structure prediction of biomolecular interactions with AlphaFold 3 (Part 1)

Abramson, J., Adler, J., Dunger, J. et al.

https://www.nature.com/articles/s41586-024-07487-w

Burak Suyunu

LifeLU Reading Group

15.05.2025

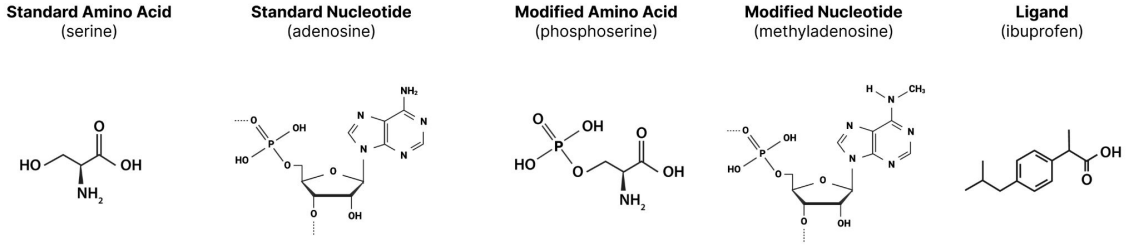
From Sequences to Multi-molecular Structures

Traditional approaches like X-ray crystallography, Cryo-EM, NMR are **time-consuming, expensive, and sparse**.

AlphaFold 2 revolutionized monomer structure prediction — but biology is more than monomers.

Need for a **unified model** that handles:

- **Multimers**
- **Ligands** (small molecules, ions)
- **RNA and DNA**

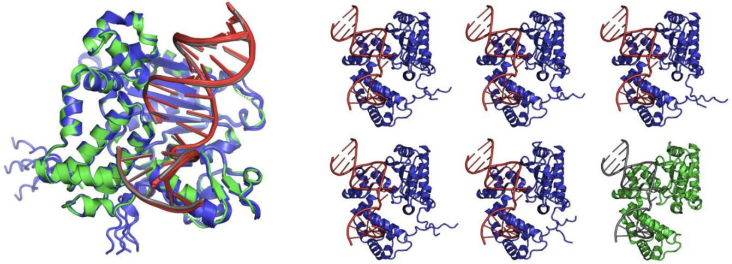


AlphaFold 3 is not just an extension; it's a **new paradigm**.

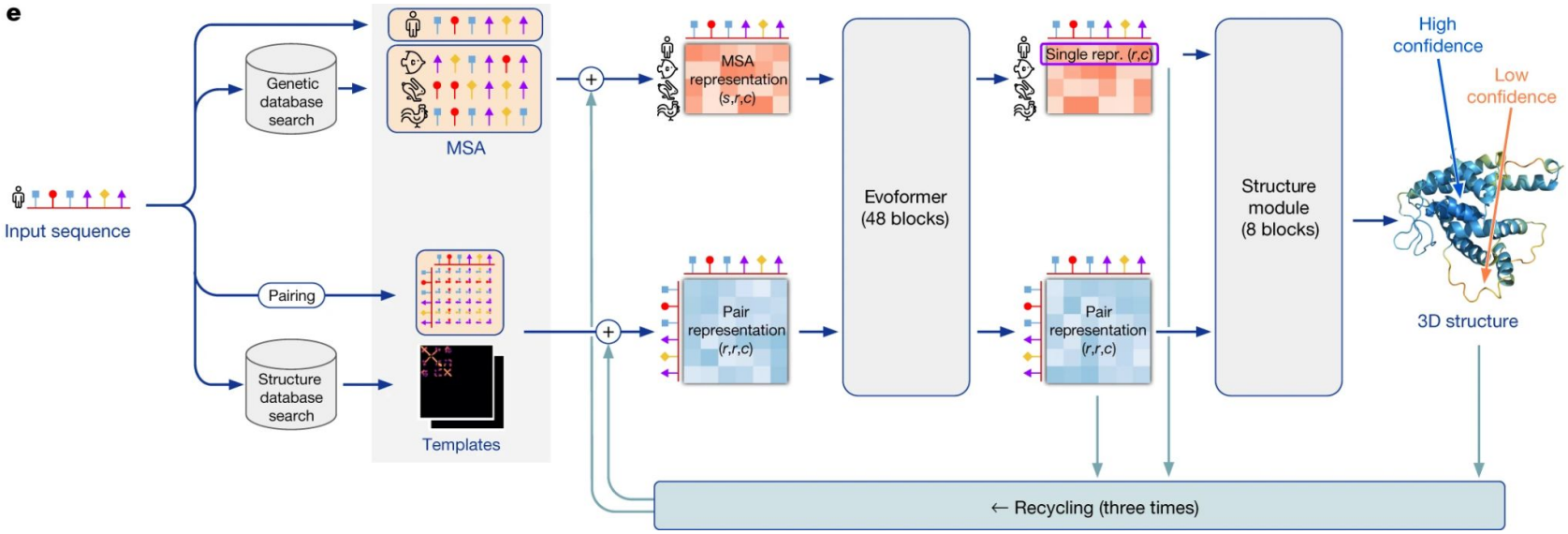
Why Predict Interactions?

- Many cellular processes involve **complex interactions**:
  - Protein–protein (multimers)
  - Protein–ligand (e.g., drug binding)
  - Protein–nucleic acid (e.g., CRISPR-Cas9)
- Structural biology must **generalize across molecules**.
- Predicting *joint structures* requires **flexible representations**.
- Classic models are modular/limited. Need a **single, generalizable model**.

• DNA in complex with globular proteins (5TBA)



From AlphaFold2 to AlphaFold3 – A Paradigm Shift



From AlphaFold2 to AlphaFold3 – A Paradigm Shift

Feature	AlphaFold2 (2021)	AlphaFold-Multimer (2021)	AlphaFold3 (2024)
Input types	Protein sequences	Protein sequences	Proteins, RNA, DNA, ligands
Complex prediction	No	Yes (proteins only)	Yes (all molecule types)
Ligand/RNA/DNA modeling	❌	❌	✅
Underlying architecture	Evoformer + structure module	Extended Evoformer	**Pairformer + Diffusion**
Training task	Supervised (PDB structures)	Supervised (PDB structures)	**Conditional diffusion**

AlphaFold 3 vs AlphaFold 2 and AlphaFold-Multimer

1. General Modeling Capabilities

- **AF2 / AF-Multimer:** Focused on protein monomers (AF2) and protein-protein complexes (AF-Multimer).
- **AF3:** Models a broader range of biomolecular complexes, including **ligands, nucleic acids, metals, and modified residues**.

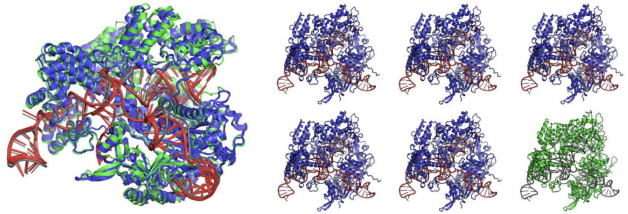
2. Structural Representation

- **AF2:** Uses rigid body frames for amino acids; side chains are parameterized with χ-angles.
- **AF3:** Represents molecules at the **atomic level**, with each atom having **independent global coordinates**—removes rigid constraints and enables modeling of arbitrary molecules.

3. Tokenization

- **AF2:** Residue-centric representations.
- **AF3:** Uses **tokens** for efficient computation:
    - Whole residues/nucleotides for standard biomolecules.
    - Individual heavy atoms for others.

• Conserved RNA-DNA-protein complexes (4OO8)



AlphaFold 3 vs AlphaFold 2 and AlphaFold-Multimer

4. Structure Prediction Approach

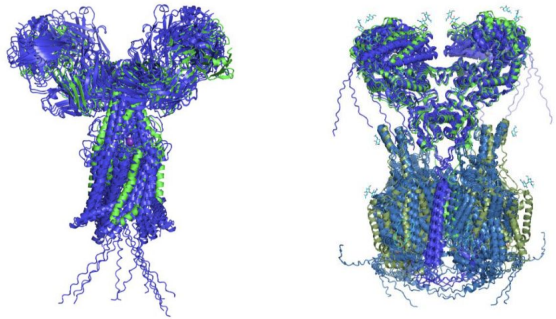
- **AF2:** Uses equivariant spatial transformations and post-relaxation for clash resolution.
- **AF3:** Employs a **generative diffusion model** (adds/removes Gaussian noise) with **minimal spatial inductive bias**. Post-processing/relaxation is rarely needed.

5. Architecture

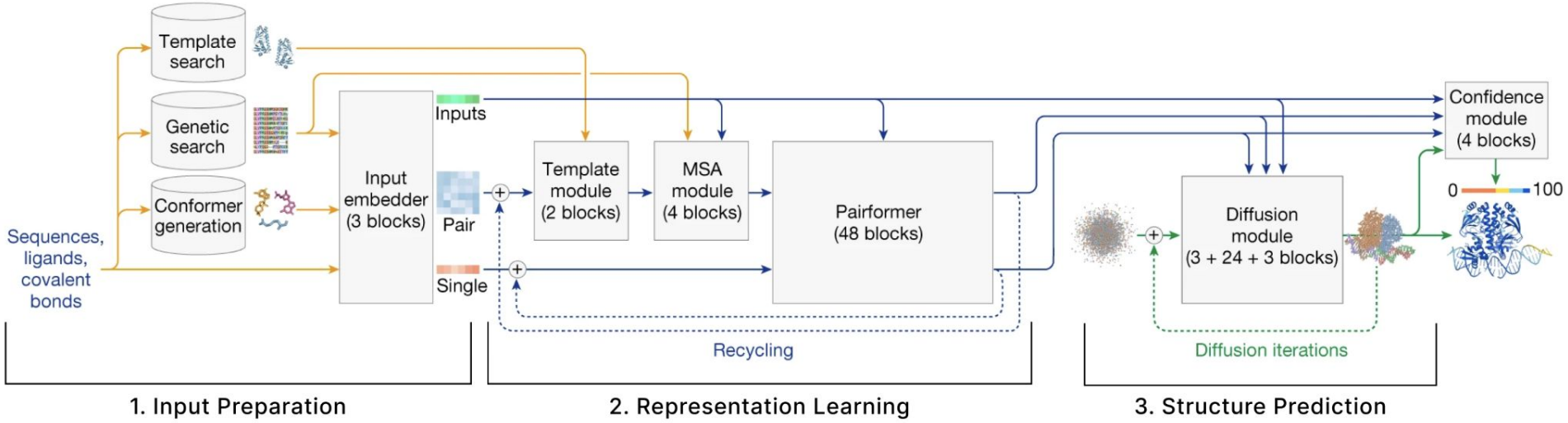
- **AF2:** Uses the **Evoformer** trunk, based on MSA and residue pair representations.
- **AF3:** Replaces Evoformer with **Pairformer**:
  - **No MSA stack** in the core, operates **only on token pairs**.
  - No outer product mean (no single-to-pair flow).

6. Activation and Data

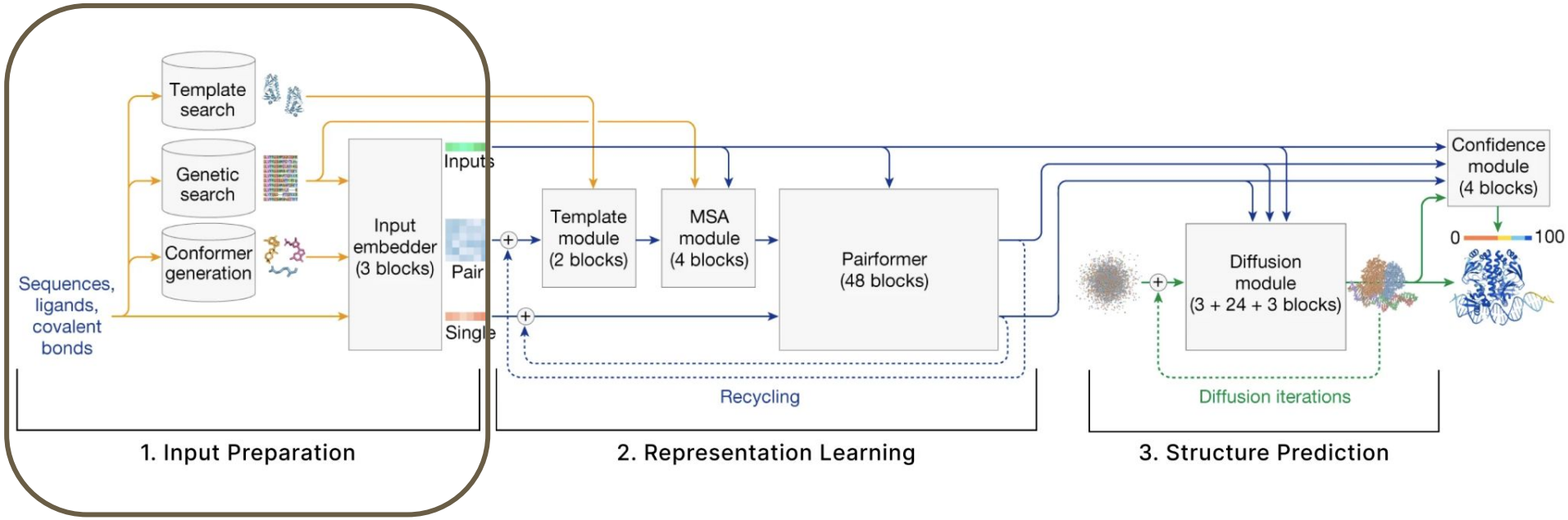
- **AF2:** Uses **ReLU** activations.
- **AF3:** Switches to **SwiGLU** in transition blocks (ReLU still used in atom attention).



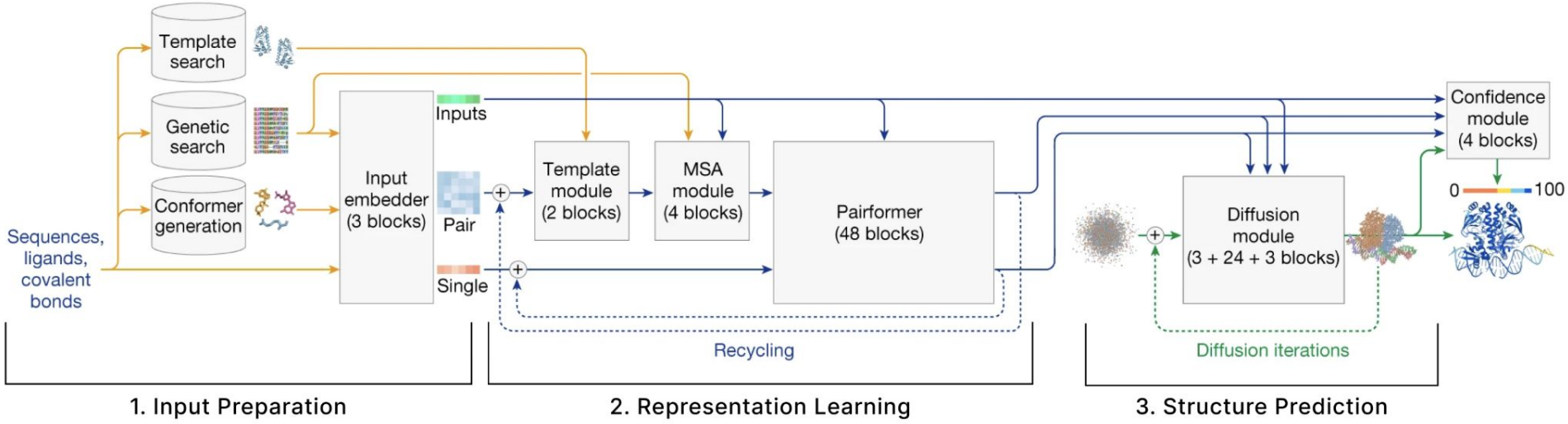
What We'll Cover in Part 1



What We'll Cover in Part 1



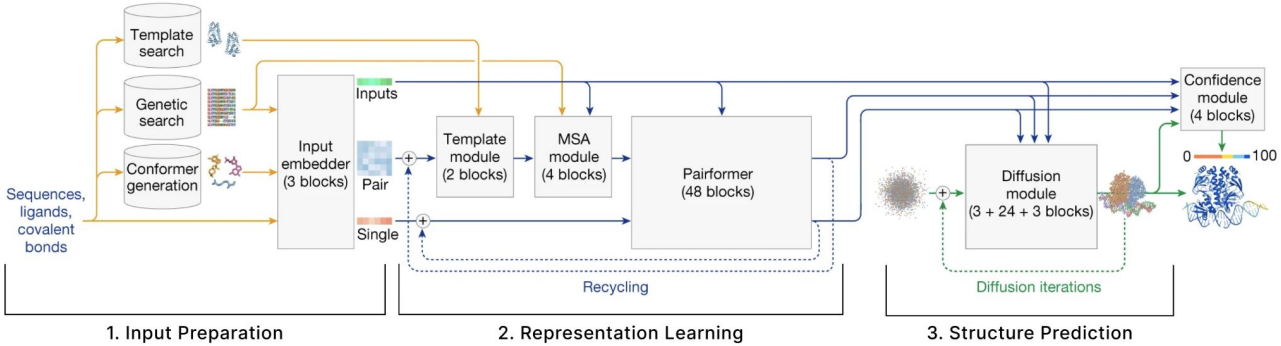
Architecture Overview



Unlike earlier AlphaFold models that only handled standard amino acid sequences, AlphaFold 3 must represent more complex inputs, requiring more advanced tokenization. A "token" can be an amino acid, nucleotide, or atom outside standard residues.

The model has three main parts:

1. **Input Preparation**: Converts user-provided sequences and similar molecules into tensors.
2. **Representation Learning**: Refines the Single and Pair tensors created in section 1 using various attention mechanisms.
3. **Structure Prediction**: Uses refined features and inputs to predict structures via conditional diffusion.



Algorithm 1 Main Inference Loop

def MainInferenceLoop($\{\mathbf{f}^*\}, N_{\text{cycle}} = 4, c_s = 384, c_z = 128$) :

$\{\mathbf{s}_i^{\text{inputs}}\} = \text{InputFeatureEmbedder}(\{\mathbf{f}^*\})$

$\mathbf{s}_i^{\text{init}} = \text{LinearNoBias}(\mathbf{s}_i^{\text{inputs}})$ $\quad \mathbf{s}_i^{\text{init}} \in \mathbb{R}^{c_s}$

$\mathbf{z}_{ij}^{\text{init}} = \text{LinearNoBias}(\mathbf{s}_i^{\text{inputs}}) + \text{LinearNoBias}(\mathbf{s}_j^{\text{inputs}})$ $\quad \mathbf{z}_{ij}^{\text{init}} \in \mathbb{R}^{c_z}$

$\mathbf{z}_{ij}^{\text{init}} \mathrel{+}= \text{RelativePositionEncoding}(\{\mathbf{f}^*\})$

$\mathbf{z}_{ij}^{\text{init}} \mathrel{+}= \text{LinearNoBias}(\mathbf{f}_{ij}^{\text{token\_bonds}})$

$\{\hat{\mathbf{z}}_{ij}\}, \{\hat{\mathbf{s}}_i\} = \mathbf{0}, \mathbf{0}$

for all $c \in [1, \ldots, N_{\text{cycle}}]$ **do**

  $\mathbf{z}_{ij} = \mathbf{z}_{ij}^{\text{init}} + \text{LinearNoBias}(\text{LayerNorm}(\hat{\mathbf{z}}_{ij}))$ $\quad \mathbf{z}_{ij} \in \mathbb{R}^{c_z}$

  $\{\mathbf{z}_{ij}\} \mathrel{+}= \text{TemplateEmbedder}(\{\mathbf{f}^*\}, \{\mathbf{z}_{ij}\})$

  $\{\mathbf{z}_{ij}\} \mathrel{+}= \text{MsaModule}(\{\mathbf{f}_{Si}^{\text{msa}}\}, \{\mathbf{z}_{ij}\}, \{\mathbf{s}_i^{\text{inputs}}\})$

  $\mathbf{s}_i = \mathbf{s}_i^{\text{init}} + \text{LinearNoBias}(\text{LayerNorm}(\hat{\mathbf{s}}_i))$ $\quad \mathbf{s}_i \in \mathbb{R}^{c_s}$

  $\{\mathbf{s}_i\}, \{\mathbf{z}_{ij}\} = \text{PairformerStack}(\{\mathbf{s}_i\}, \{\mathbf{z}_{ij}\})$

  $\{\hat{\mathbf{s}}_i\}, \{\hat{\mathbf{z}}_{ij}\} \leftarrow \{\mathbf{s}_i\}, \{\mathbf{z}_{ij}\}$

end for

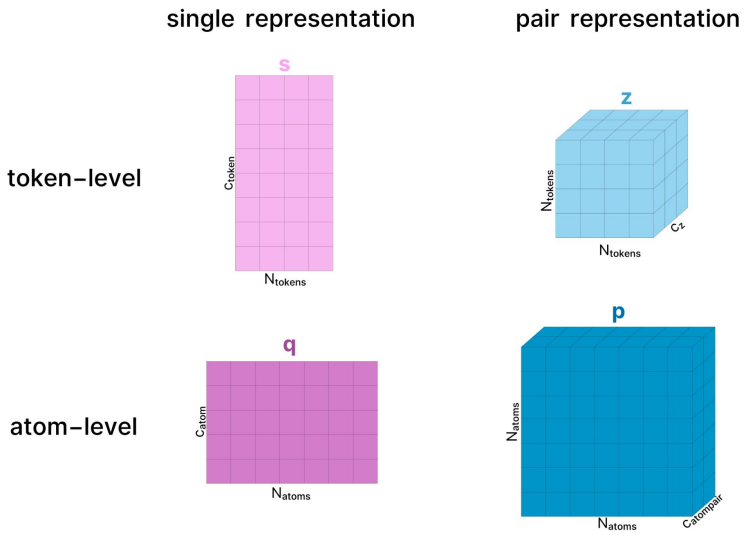
$\{\vec{\mathbf{x}}_l^{\text{pred}}\} = \text{SampleDiffusion}(\{\mathbf{f}^*\}, \{\mathbf{s}_i^{\text{inputs}}\}, \{\mathbf{s}_i\}, \{\mathbf{z}_{ij}\})$

$\{\mathbf{p}_l^{\text{plddt}}\}, \{\mathbf{p}_{ij}^{\text{pae}}\}, \{\mathbf{p}_{ij}^{\text{pde}}\}, \{\mathbf{p}_l^{\text{resolved}}\} = \text{ConfidenceHead}(\{\mathbf{s}_i^{\text{inputs}}\}, \{\mathbf{s}_i\}, \{\mathbf{z}_{ij}\}, \{\vec{\mathbf{x}}_l^{\text{pred}}\})$

$\mathbf{p}_{ij}^{\text{distogram}} = \text{DistogramHead}(\mathbf{z}_{ij})$

return $\{\vec{\mathbf{x}}_l^{\text{pred}}\}, \{\mathbf{p}_l^{\text{plddt}}\}, \{\mathbf{p}_{ij}^{\text{pae}}\}, \{\mathbf{p}_{ij}^{\text{pde}}\}, \{\mathbf{p}_l^{\text{resolved}}\}, \{\mathbf{p}_{ij}^{\text{distogram}}\}$

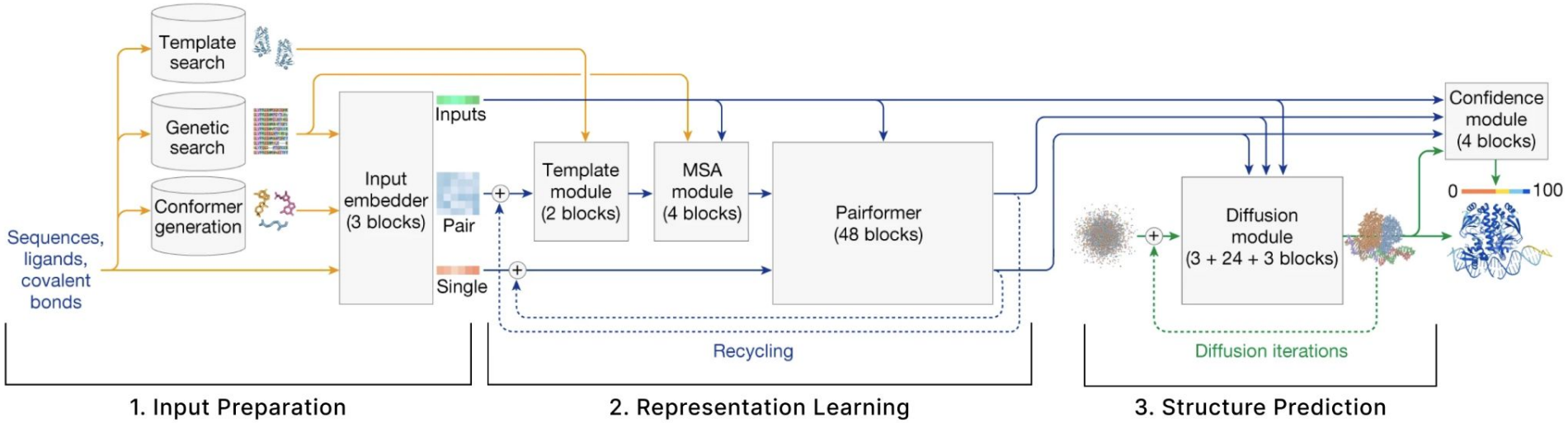
Notes on the variables and diagrams



AlphaFold 3 represents protein complexes using two key tensors: **Single** (per-token/atom features) and **Pair** (relationships between token/atom pairs). These can be at the token or atom level, shown with consistent names and colors.

Diagrams show how activation shapes change—not model weights. Tensor labels match those in the AF3 paper. Names are mostly preserved, but some (e.g., c to q in atom-level single) indicate updates through processing. LayerNorms are used throughout but mostly omitted for clarity.

1. Input Preparation

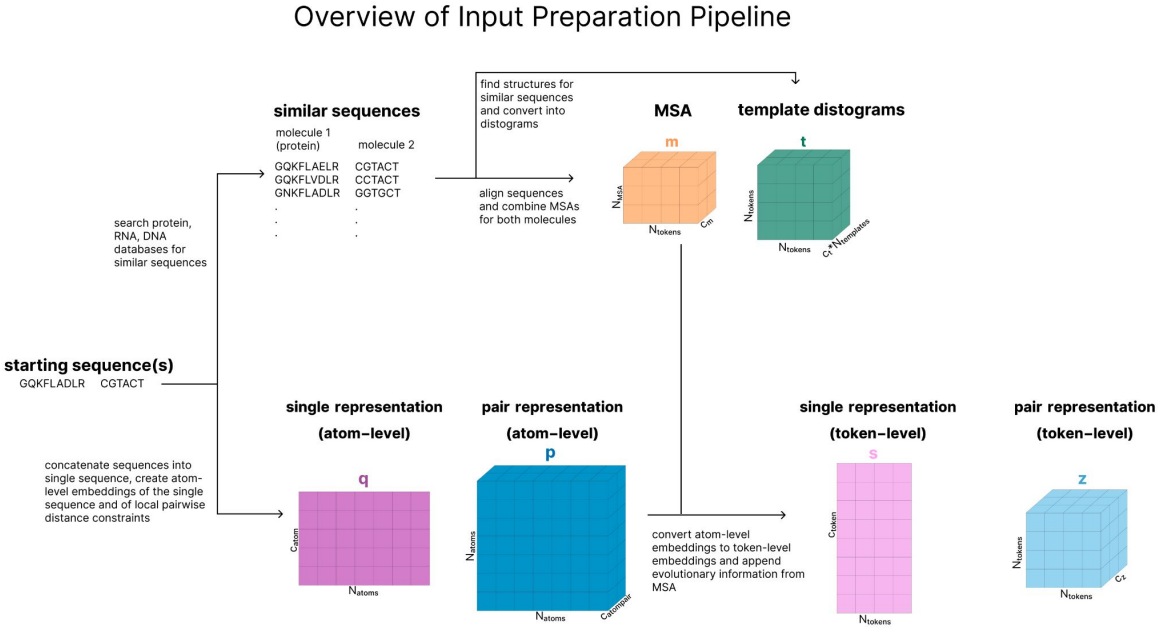


Input Features

Feature & Shape	Description
residue_index $[N_{\text{token}}]$	Residue number in the token's original input chain.
token_index $[N_{\text{token}}]$	Token number. Increases monotonically; does not restart at 1 for new chains.
asym_id $[N_{\text{token}}]$	Unique integer for each distinct chain.
entity_id $[N_{\text{token}}]$	Unique integer for each distinct sequence.
sym_id $[N_{\text{token}}]$	Unique integer within chains of this sequence. E.g. if chains A, B and C share a sequence but D does not, their sym_ids would be [0, 1, 2, 0].
restype $[N_{\text{token}}, 32]$	One-hot encoding of the sequence. 32 possible values: 20 amino acids + unknown, 4 RNA nucleotides + unknown, 4 DNA nucleotides + unknown, and gap. Ligands represented as "unknown amino acid".
is_protein / rna / dna / ligand $[N_{\text{token}}]$	4 masks indicating the molecule type of a particular token.
ref_pos $[N_{\text{atom}}, 3]$	Atom positions in the reference conformer, with a random rotation and translation applied. Atom positions are given in Å.
ref_mask $[N_{\text{atom}}]$	Mask indicating which atom slots are used in the reference conformer.
ref_element $[N_{\text{atom}}, 128]$	One-hot encoding of the element atomic number for each atom in the reference conformer, up to atomic number 128.
ref_charge $[N_{\text{atom}}]$	Charge for each atom in the reference conformer.

Feature & Shape	Description
ref_atom_name_chars $[N_{\text{atom}}, 4, 64]$	One-hot encoding of the unique atom names in the reference conformer. Each character is encoded as $\text{ord}(c) - 32$, and names are padded to length 4.
ref_space_uid $[N_{\text{atom}}]$	Numerical encoding of the chain id and residue index associated with this reference conformer. Each (chain id, residue index) tuple is assigned an integer on first appearance.
msa $[N_{\text{msa}}, N_{\text{token}}, 32]$	One-hot encoding of the processed MSA, using the same classes as restype.
has_deletion $[N_{\text{msa}}, N_{\text{token}}]$	Binary feature indicating if there is a deletion to the left of each position in the MSA.
deletion_value $[N_{\text{msa}}, N_{\text{token}}]$	Raw deletion counts (the number of deletions to the left of each MSA position) are transformed to $[0, 1]$ using $\frac{2}{\pi} \arctan \frac{d}{3}$.
profile $[N_{\text{token}}, 32]$	Distribution across restypes in the main MSA. Computed before MSA processing (subsection 2.3).
deletion_mean $[N_{\text{token}}]$	Mean number of deletions at each position in the main MSA. Computed before MSA processing (subsection 2.3).
template_restype $[N_{\text{templ}}, N_{\text{token}}]$	One-hot encoding of the template sequence, see restype.
template_pseudo_beta_mask $[N_{\text{templ}}, N_{\text{token}}]$	Mask indicating if the C^{β} (C^{α} for glycine) has coordinates for the template at this residue.
template_backbone_frame_mask $[N_{\text{templ}}, N_{\text{token}}]$	Mask indicating if coordinates exist for all atoms required to compute the backbone frame (used in the template_unit_vector feature).
template_distogram $[N_{\text{templ}}, N_{\text{token}}, N_{\text{token}}, 39]$	A one-hot pairwise feature indicating the distance between C^{β} atoms (C^{α} for glycine). Pairwise distances are discretized into 38 bins of equal width between 3.25 Å and 50.75 Å; one more bin contains any larger distances.
template_unit_vector $[N_{\text{templ}}, N_{\text{token}}, N_{\text{token}}, 3]$	The unit vector of the displacement of the C^{α} atom of all residues within the local frame of each residue. Local frames are computed as in [1].
token_bonds $[N_{\text{token}}, N_{\text{token}}]$	A 2D matrix indicating if there is a bond between any atom in token i and token j, restricted to just polymer-ligand and ligand-ligand bonds and bonds less than 2.4 Å during training.

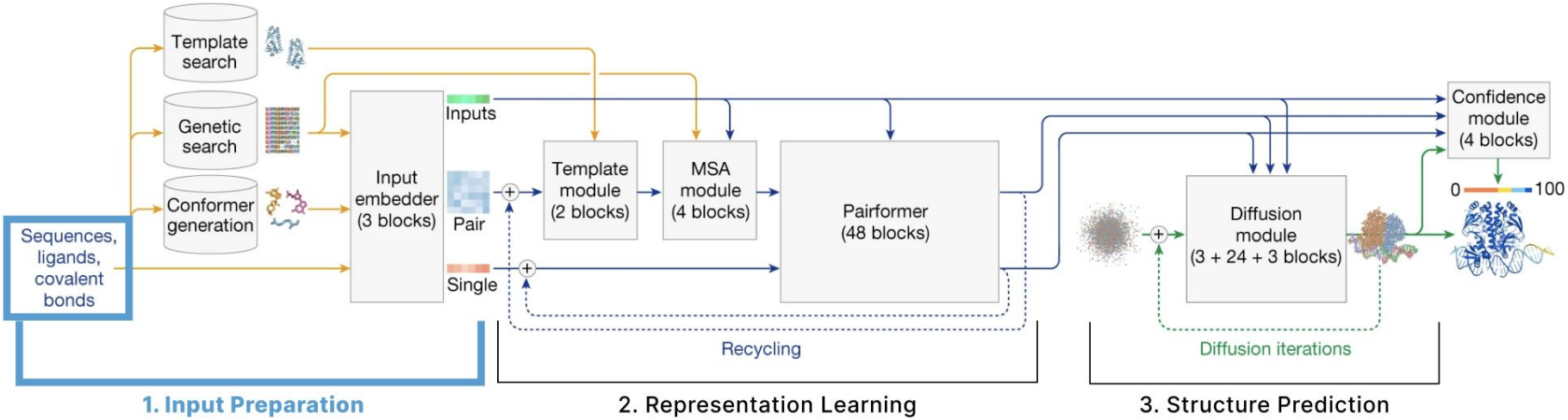
Input Preparation Pipeline



The user provides a protein sequence and optionally other molecules. This section turns inputs into 6 key tensors for the model: **s** (token-level single), **z** (token-level pair), **q** (atom-level single), **p** (atom-level pair), **m** (MSA), and **t** (template). Steps include:

- **Tokenization**: Molecules are split into tokens (e.g., amino acids, atoms); distinguishes atom-level vs. token-level representations.
- **Retrieval (MSA & Templates)**: Uses external databases to find similar sequences (MSA, **m**) and structural templates (**t**) to guide prediction.
- **Create Atom-Level Representations**: Builds **q** and **p** from 3D conformers; encodes atom-level features and inter-atomic relationships.
- **Update Atom-Level Representations**: The Atom Transformer updates **q** using custom attention, LayerNorm, gating, and transition modules.
- **Aggregate Atom- to Token-Level**: Aggregates atom-level **q** and **p** into token-level **s** and **z**, incorporating MSA info and known ligand bonding data.

Tokenization



Tokenize input sequences

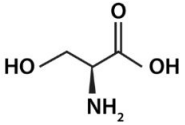
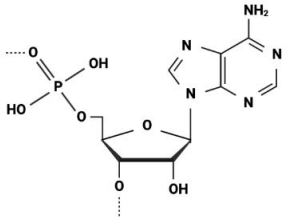
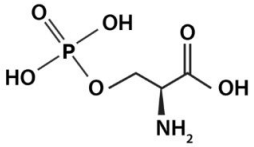
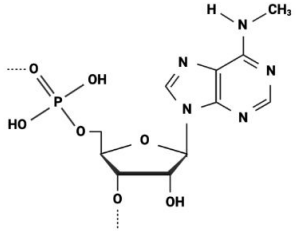
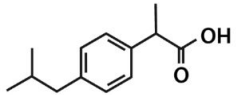
Retrieve similar sequences and structures to create MSA and templates

Create atom-level representation of sequences

Update atom-level representation (Atom Transformer)

Aggregate atom-level representation to token-level

Tokenization

Standard Amino Acid (serine)	**Standard Nucleotide** (adenosine)	**Modified Amino Acid** (phosphoserine)	**Modified Nucleotide** (methyladenosine)	**Ligand** (ibuprofen)
				
7 atoms* 1 tokens	**23 atoms* 1 token**	**11 atoms* 11 tokens**	**24 atoms* 24 tokens**	**15 atoms* 15 tokens**

*atoms=heavy atoms

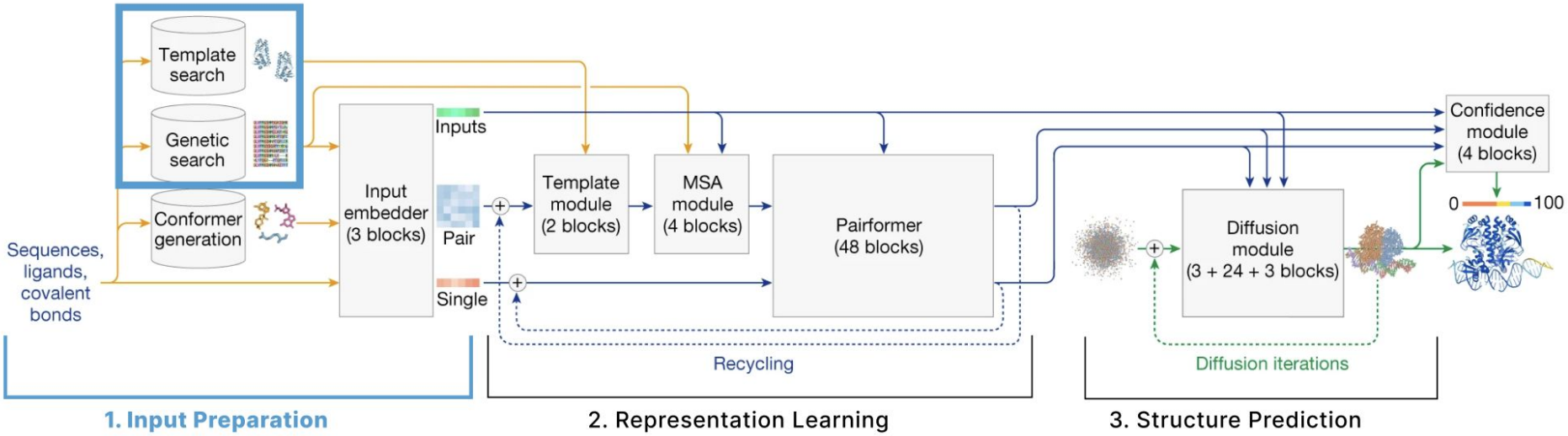
AF2 used one token per amino acid. AF3 extends this:

- **Standard amino acids & nucleotides**: 1 token each
- **Modified residues & other molecules**: 1 token per atom

Thus, tokens may represent multiple atoms (e.g., an amino acid) or a single atom (e.g., a ligand).

- 35 amino acids → 35 tokens (~600 atoms)
- 35-atom ligand → 35 tokens

Retrieval (Create MSA and Templates)



Tokenize input sequences

Retrieve similar sequences and structures to create MSA and templates

Create atom-level representation of sequences

Update atom-level representation (Atom Transformer)

Aggregate atom-level representation to token-level

Retrieval (Create MSA and Templates)

Retrieval as Input

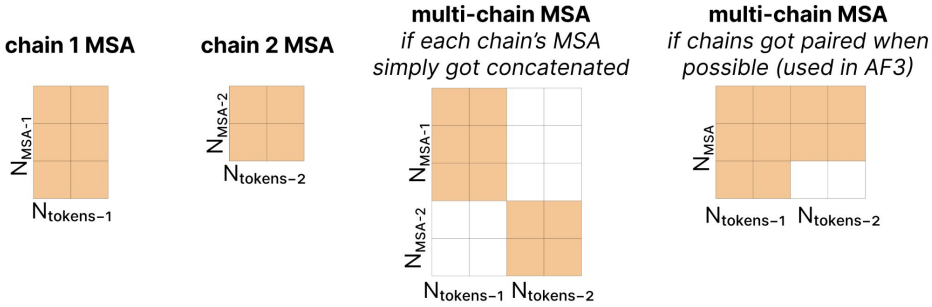
AF3 begins by retrieving similar protein and RNA sequences (MSA) and related structures (templates). Unlike AF-Multimer, RNA sequences are also retrieved.

Why Use MSAs and Templates?

MSAs reveal evolutionary patterns—conserved positions and co-evolving residues—helping infer structure. Known structures (templates) of similar proteins also guide prediction, following the principle of homology modeling.

How Retrieval Works

Protein and RNA chains are searched via HMM-based tools. Hits are aligned into MSAs, with chain pairing used to reduce sparsity. Up to 16384 sequences are included. Templates (up to 4 per chain) are selected from PDB using HMM alignment.

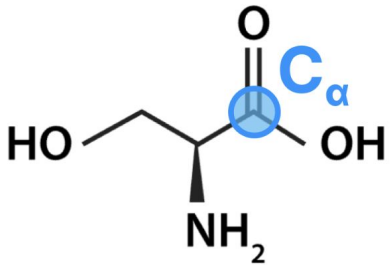


Retrieval (Create MSA and Templates)

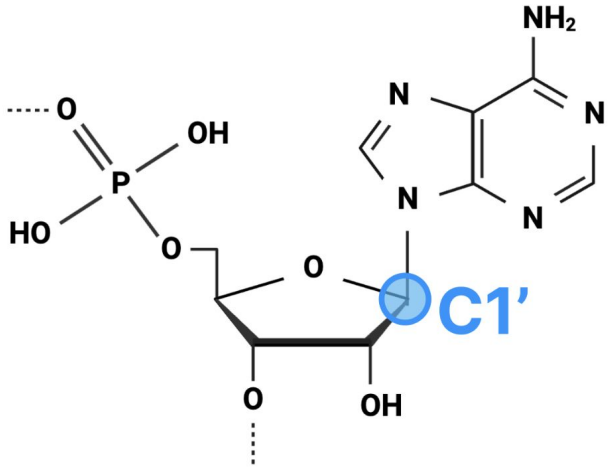
Template Representation

Each template's 3D structure is encoded as a distance matrix between token pairs. Distances are binned into histograms ("distograms"), with added metadata like chain ID and resolution. Only intra-chain distances are used, ignoring inter-chain info.

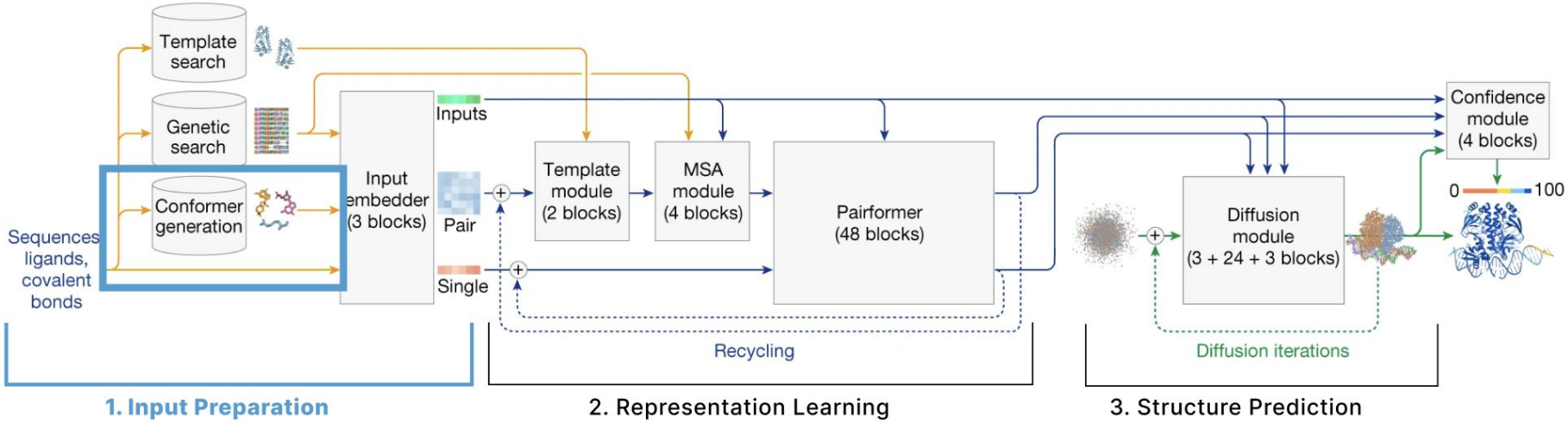
Standard Amino Acid



Standard Nucleotide



Create Atom-Level Representations



Tokenize input sequences

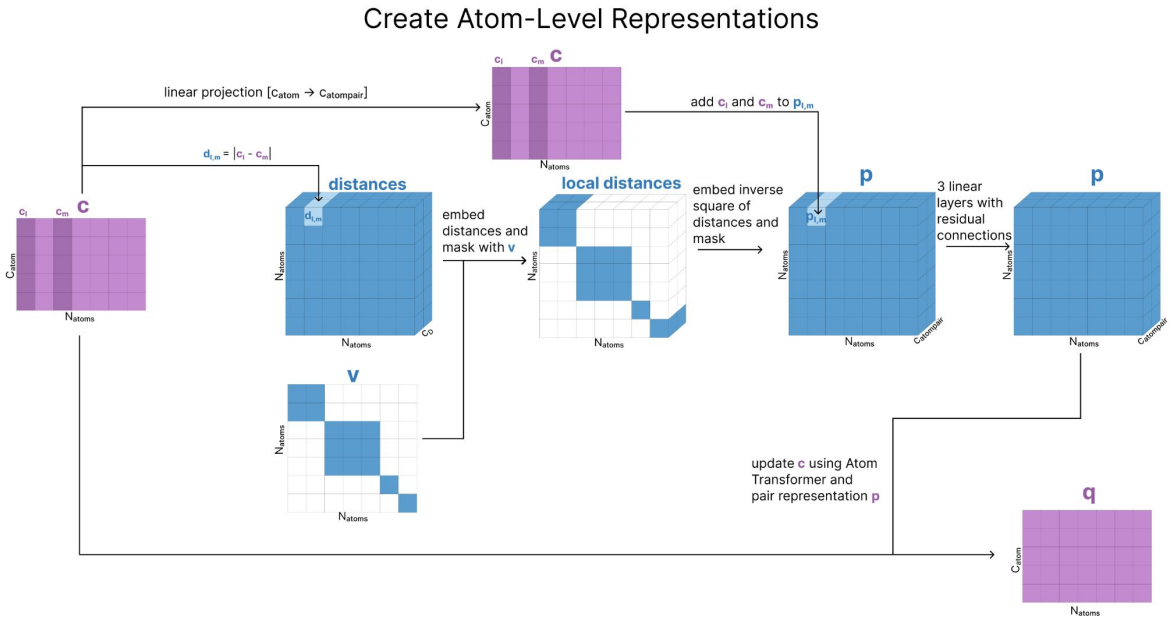
Retrieve similar sequences and structures to create MSA and templates

Create atom-level representation of sequences

Update atom-level representation (Atom Transformer)

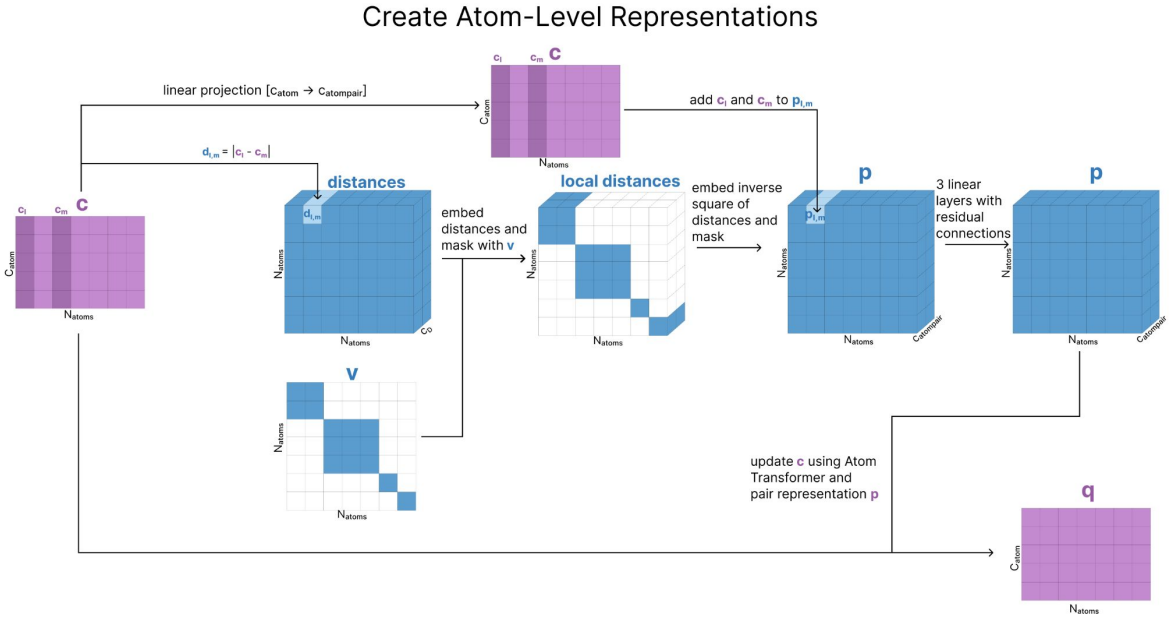
Aggregate atom-level representation to token-level

Create Atom-Level Representations



To create the atom-level representation **q**, we start by generating a “reference conformer” for each amino acid, nucleotide, and ligand—a 3D structure based on known local arrangements. Amino acids use standard conformers, while ligands are generated via RDKit’s ETKDGv3 algorithm. We combine conformer coordinates with atomic features (charge, atomic number) into matrix **c**, which initializes the atom-pair distance matrix **p**. Masking ensures distances only reflect known intra-token values. After processing through embeddings and layers, we copy **c** to **q**, which is updated throughout the model.

Create Atom-Level Representations



Algorithm 5 Atom attention encoder

def AtomAttentionEncoder($\{\mathbf{f}^*\}, \{\mathbf{r}_l\}, \{\mathbf{s}_i^{\text{trunk}}\}, \{\mathbf{z}_{ij}\}, c_{\text{atom}}, c_{\text{atompair}}, c_{\text{token}}$) :

 # Create the atom single conditioning: Embed per-atom meta data

$\mathbf{c}_l = \text{LinearNoBias}(\text{concat}(\vec{\mathbf{f}}_l^{\text{ref\_pos}}, \mathbf{f}_l^{\text{ref\_charge}}, \mathbf{f}_l^{\text{ref\_mask}}, \mathbf{f}_l^{\text{ref\_element}}, \mathbf{f}_l^{\text{ref\_atom\_name\_chars}}))$ $\quad \mathbf{c}_l \in \mathbb{R}^{c_{\text{atom}}}$, $l \in \{1, \ldots, N_{\text{atoms}}\}$

 # Embed offsets between atom reference positions

$\vec{\mathbf{d}}_{lm} = \vec{\mathbf{f}}_l^{\text{ref\_pos}} - \vec{\mathbf{f}}_m^{\text{ref\_pos}}$ $\quad \vec{\mathbf{d}}_{lm} \in \mathbb{R}^3$

$v_{lm} = (\mathbf{f}_l^{\text{ref\_space\_uid}} == \mathbf{f}_m^{\text{ref\_space\_uid}})$ $\quad v_{lm} \in \mathbb{R}$

$\mathbf{p}_{lm} = \text{LinearNoBias}(\vec{\mathbf{d}}_{lm}) \cdot v_{lm}$ $\quad \mathbf{p}_{lm} \in \mathbb{R}^{c_{\text{atompair}}}$

 # Embed pairwise inverse squared distances, and the valid mask.

$\mathbf{p}_{lm}$ += $\text{LinearNoBias}\left(1/\left(1 + \|\vec{\mathbf{d}}_{lm}\|^2\right)\right) \cdot v_{lm}$

$\mathbf{p}_{lm}$ += $\text{LinearNoBias}(v_{lm}) \cdot v_{lm}$

 # Initialise the atom single representation as the single conditioning.

$\mathbf{q}_l = \mathbf{c}_l$ $\quad \mathbf{q}_l \in \mathbb{R}^{c_{\text{atom}}}$

 # If provided, add trunk embeddings and noisy positions.

if $\{\mathbf{r}_l\} \neq \emptyset$ **then**

 # Broadcast the single and pair embedding from the trunk.

$\mathbf{c}_l$ += $\text{LinearNoBias}(\text{LayerNorm}(\mathbf{s}^{\text{trunk}}_{\text{tok\_idx}(l)}))$

$\mathbf{p}_{lm}$ += $\text{LinearNoBias}(\text{LayerNorm}(\mathbf{z}_{\text{tok\_idx}(l)\ \text{tok\_idx}(m)}))$

 # Add the noisy positions.

$\mathbf{q}_l$ += $\text{LinearNoBias}\ (\mathbf{r}_l)$

end if

 # Add the combined single conditioning to the pair representation.

$\mathbf{p}_{lm}$ += $\text{LinearNoBias}(\text{relu}(\mathbf{c}_l)) + \text{LinearNoBias}(\text{relu}(\mathbf{c}_m))$

 # Run a small MLP on the pair activations.

$\mathbf{p}_{lm}$ += $\text{LinearNoBias}(\text{relu}(\text{LinearNoBias}(\text{relu}(\text{LinearNoBias}(\text{relu}(\mathbf{p}_{lm}))))))$

 # Cross attention transformer.

$\{\mathbf{q}_l\} = \text{AtomTransformer}(\{\mathbf{q}_l\}, \{\mathbf{c}_l\}, \{\mathbf{p}_{lm}\}, N_{\text{block}} = 3, N_{\text{head}} = 4)$

 # Aggregate per-atom representation to per-token representation

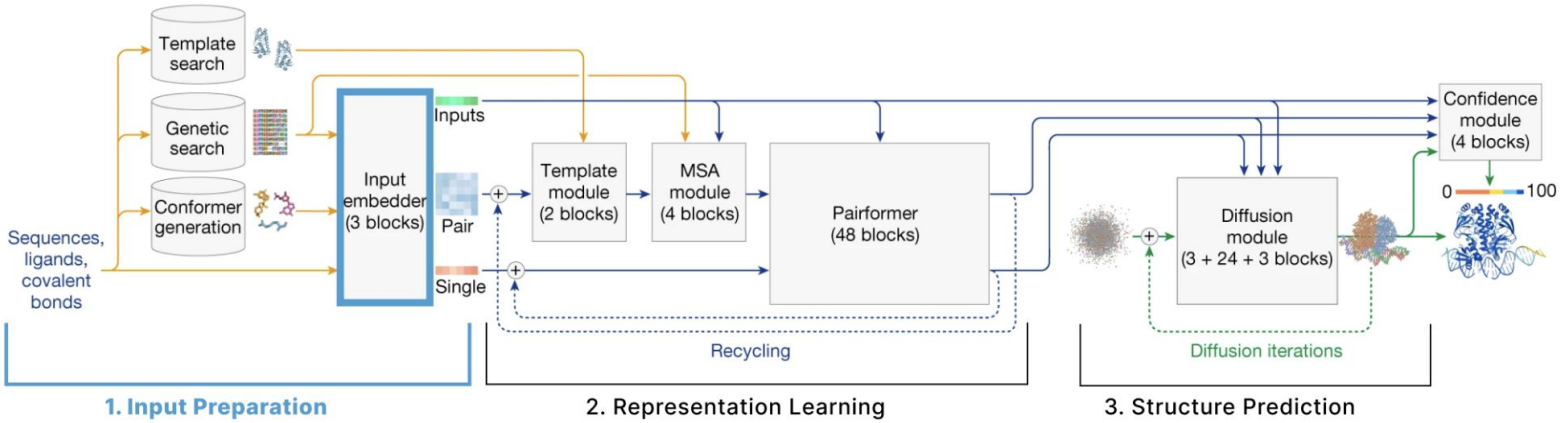
$\mathbf{a}_i = \underset{\substack{l \in \{1,\ldots,N_{\text{atoms}}\} \\ \text{tok\_idx}(l) = i}}{\text{mean}} (\text{relu}(\text{LinearNoBias}(\mathbf{q}_l)))$ $\quad \mathbf{a}_i \in \mathbb{R}^{c_{\text{token}}}$

$\mathbf{q}_l^{\text{skip}}, \mathbf{c}_l^{\text{skip}}, \mathbf{p}_{lm}^{\text{skip}} = \mathbf{q}_l, \mathbf{c}_l, \mathbf{p}_{lm}$

return $\{\mathbf{a}_i\}, \{\mathbf{q}_l^{\text{skip}}\}, \{\mathbf{c}_l^{\text{skip}}\}, \{\mathbf{p}_{lm}^{\text{skip}}\}$

To create the atom-level representation **q**, we start by generating a “reference conformer” for each amino acid, nucleotide, and ligand—a 3D structure based on known local arrangements. Amino acids use standard conformers, while ligands are generated via RDKit’s ETKDGv3 algorithm. We combine conformer coordinates with atomic features (charge, atomic number) into matrix **c**, which initializes the atom-pair distance matrix **p**. Masking ensures distances only reflect known intra-token values. After processing through embeddings and layers, we copy **c** to **q**, which is updated throughout the model.

Update Atom-Level Representations (Atom Transformer)



Tokenize input sequences

Retrieve similar sequences and structures to create MSA and templates

Create atom-level representation of sequences

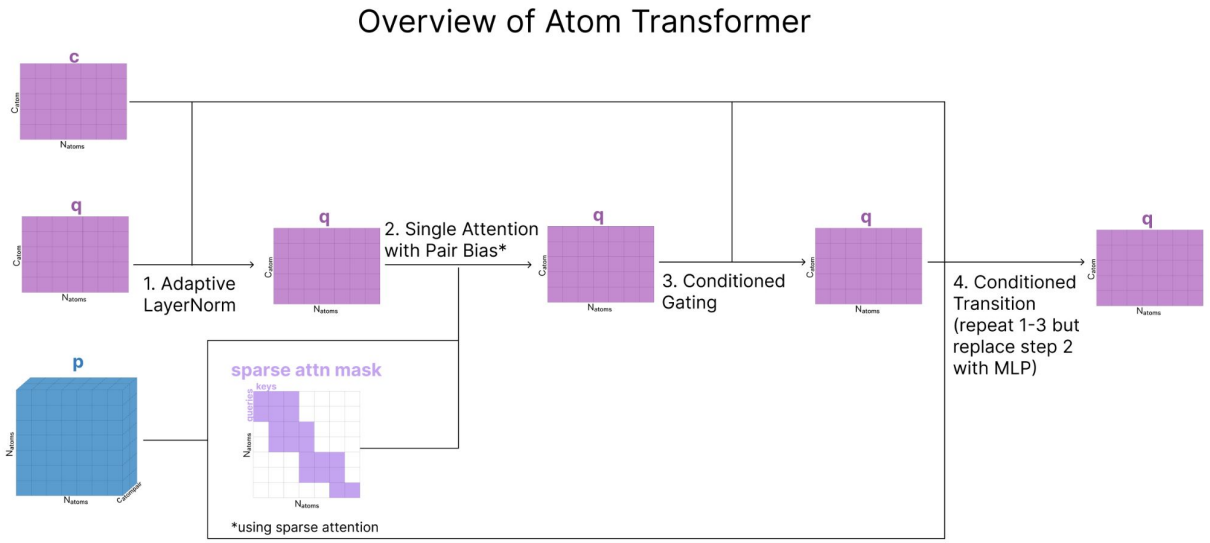
Update atom-level representation (Atom Transformer)

Aggregate atom-level representation to token-level

AtomTransformer introduces building blocks used elsewhere

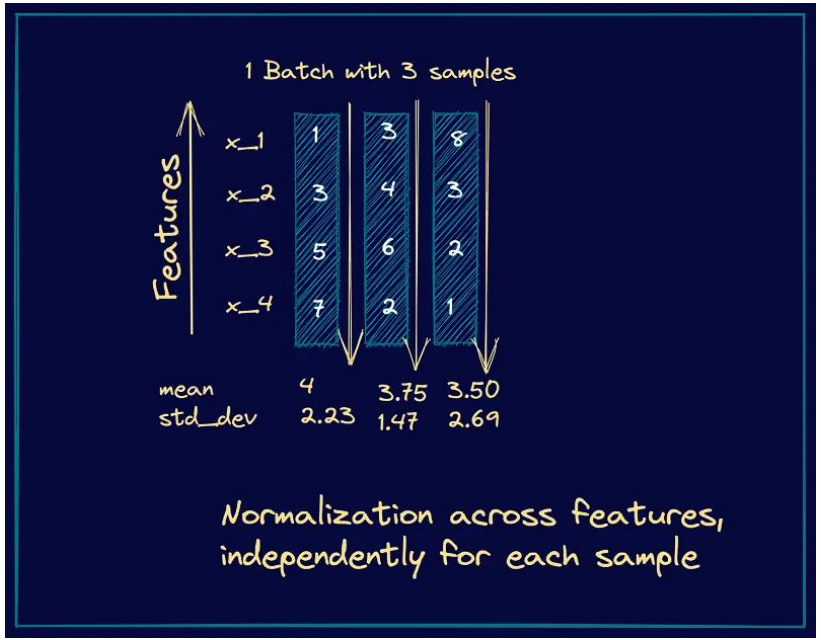
* **Adaptive LayerNorm**
* **Attention with Pair Bias**
* **Sequence-local atom attention**
* **Conditioned Gating**
* **Conditioned Transition blocks**

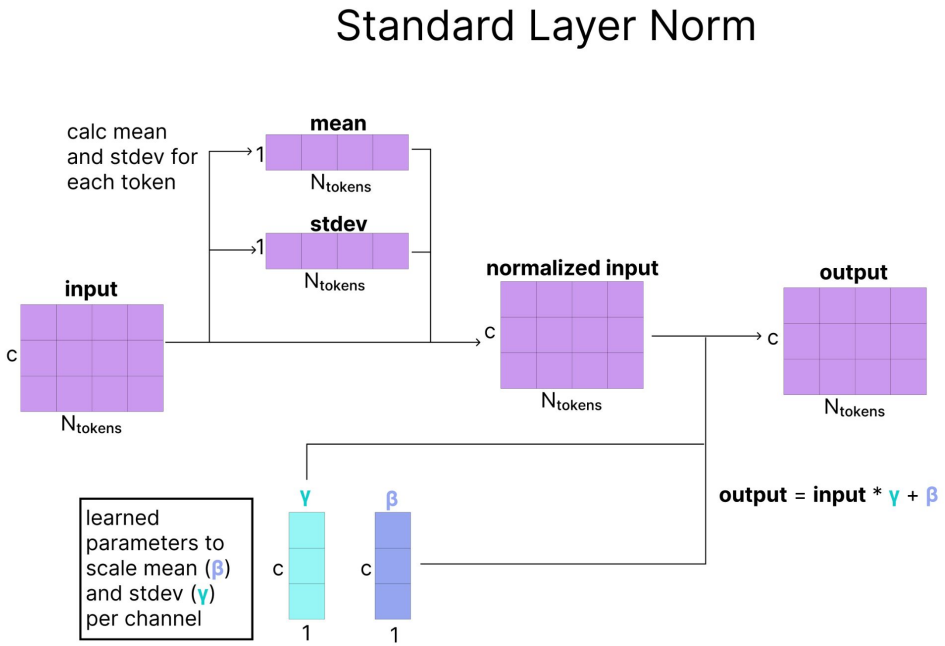
Create Atom-Level Representations



AF3 updates atom-level representations **q** and pair representations **p** using the Atom Transformer, which integrates original features **c** as a residual input.

Atom Transformer - 1. Adaptive LayerNorm (AdaNorm)





Instead of fixed scaling (gamma) and bias (beta), AdaNorm adaptively predicts these parameters from **c** to normalize **q**.

Atom Transformer - 1. Adaptive LayerNorm (AdaNorm)

Algorithm 26 Adaptive LayerNorm

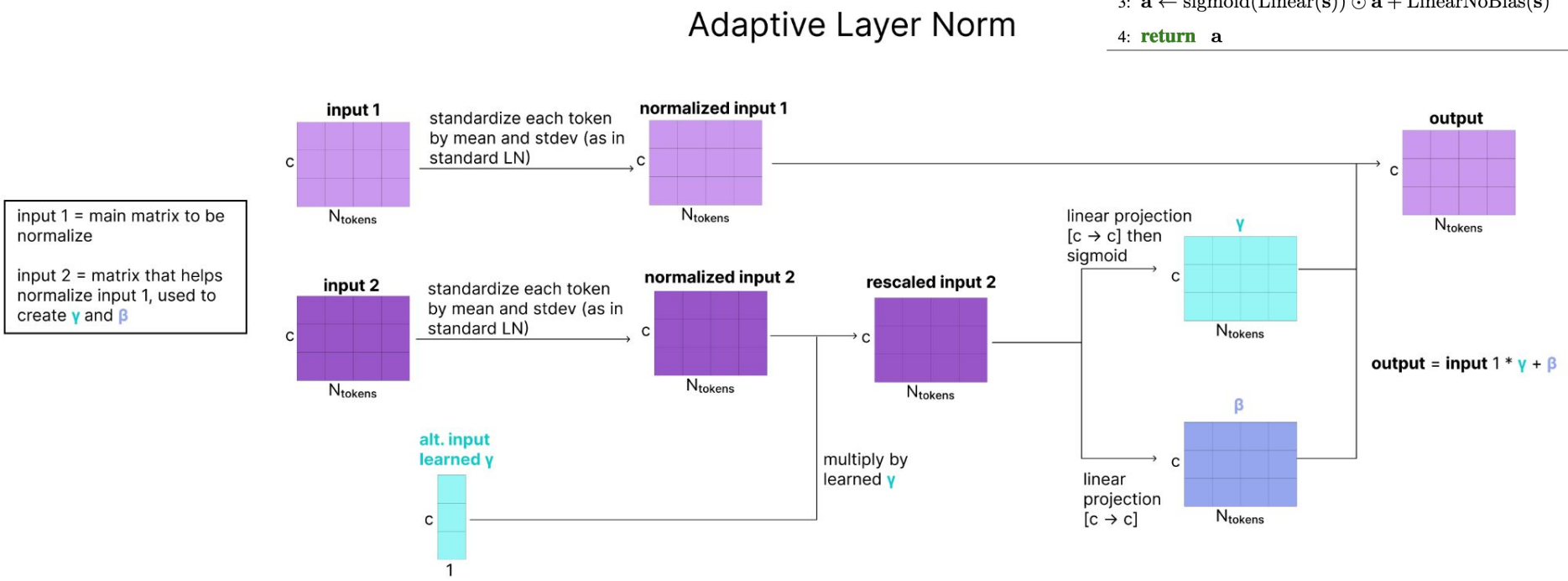
def $\text{AdaLN}(\mathbf{a}, \mathbf{s})$:

1: $\mathbf{a} \leftarrow \text{LayerNorm}(\mathbf{a},\ \text{scale=False, offset=False})$

2: $\mathbf{s} \leftarrow \text{LayerNorm}(\mathbf{s},\ \text{offset=False})$

3: $\mathbf{a} \leftarrow \text{sigmoid}(\text{Linear}(\mathbf{s})) \odot \mathbf{a} + \text{LinearNoBias}(\mathbf{s})$

4: **return** $\mathbf{a}$

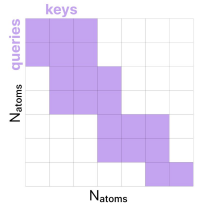


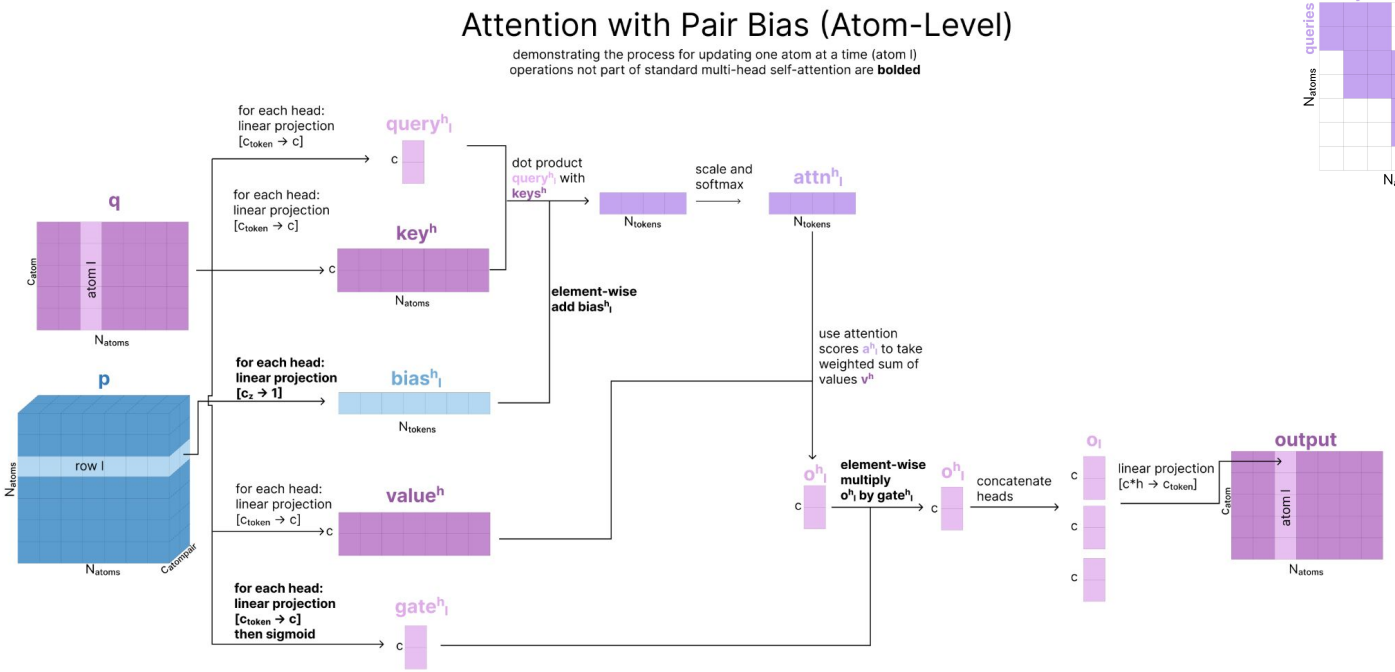
Instead of fixed scaling (gamma) and bias (beta), AdaNorm adaptively predicts these parameters from **c** to normalize **q**.

Atom Transformer - 2. Attention with Pair Bias & Gating

Sequence-local atom attention

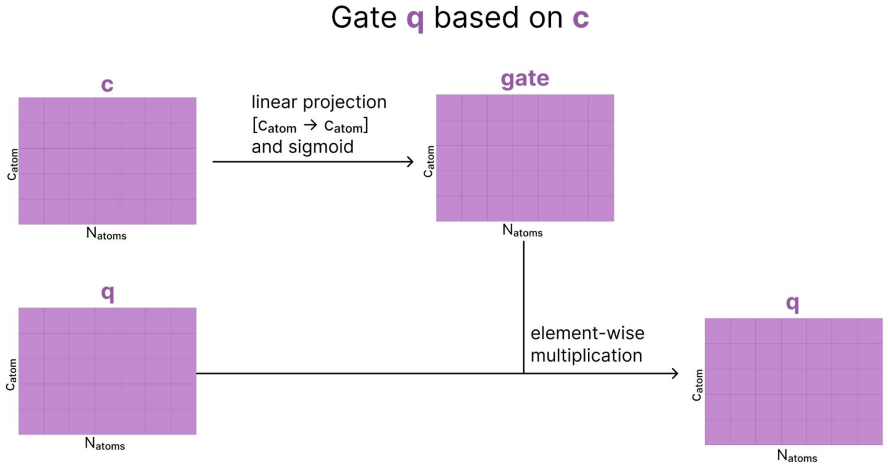
Self-attention run within rectangular blocks where 32 query atoms can attend to 128 nearby key atoms

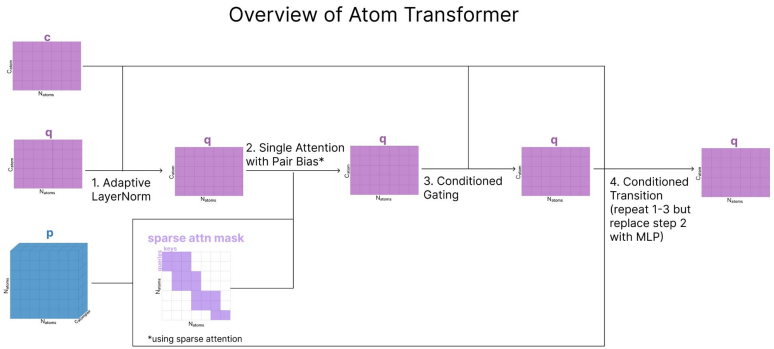




- Attention uses queries, keys, and values from **q** with added pairwise biases from **p**.
- A gating sigmoid filters attention output, controlling information flow like LSTM gates.
- Sparse local attention attends over groups of 32 atoms at a time can all attend to 128 other atoms for efficiency.

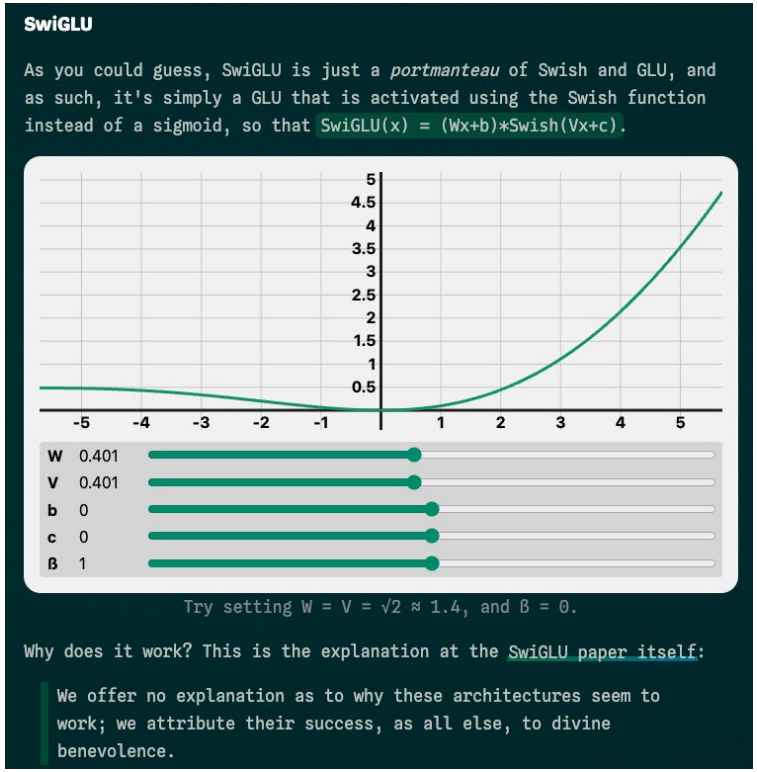
Atom Transformer - 3. Conditioned Gating

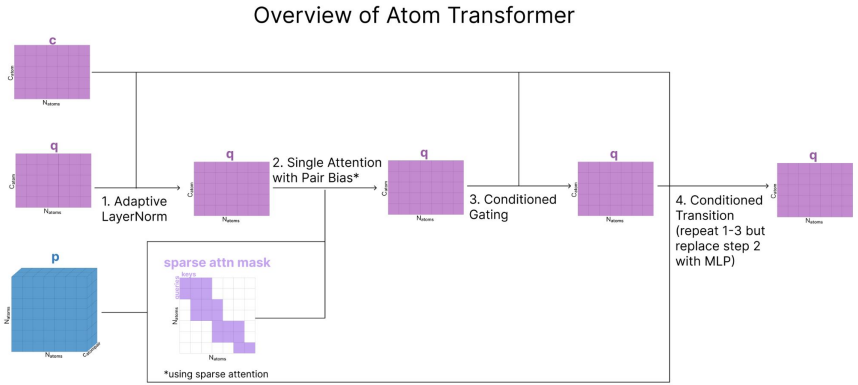




A second gating step is applied, conditioned on **c**, to selectively modulate the representation.

Atom Transformer - 4. Conditioned Transition (MLP with SwiGLU)

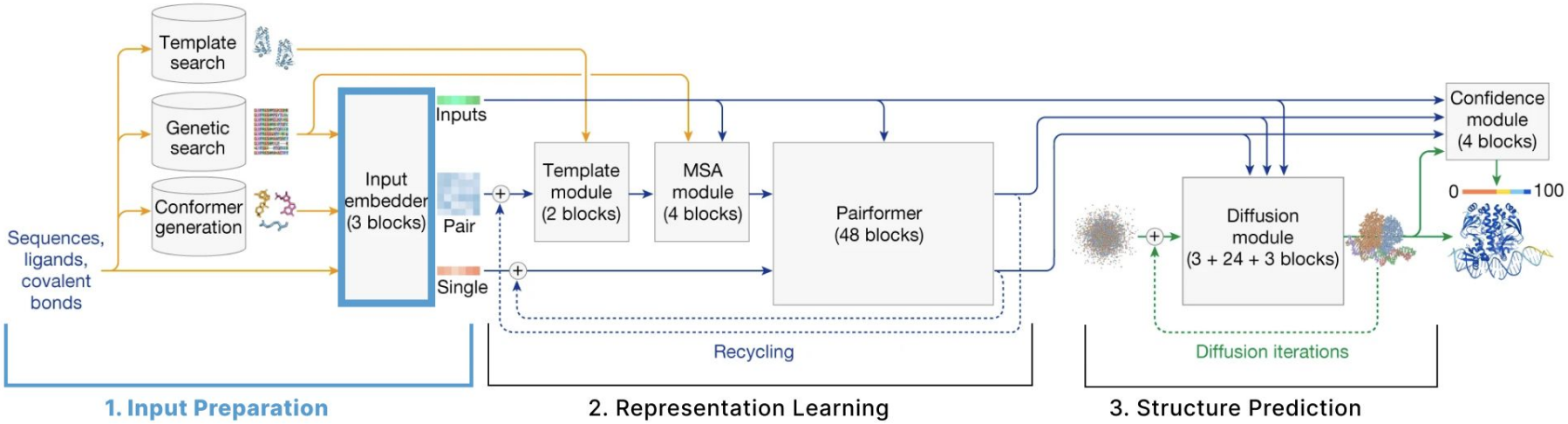




MLP layers are “conditioned” by AdaNorm and gating based on **c**. AF3 replaces ReLU with SwiGLU for improved non-linearity.

https://jcarlosroldan.com/post/348/what-is-swiglu

Aggregate Atom-Level → Token-Level



Tokenize input sequences

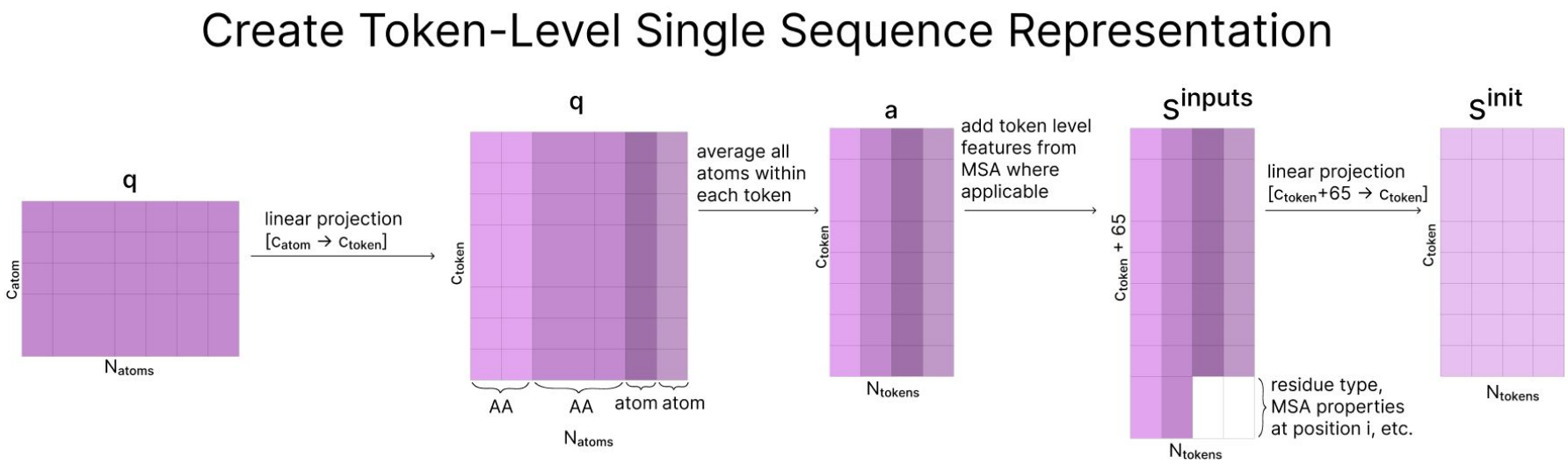
Retrieve similar sequences and structures to create MSA and templates

Create atom-level representation of sequences

Update atom-level representation (Atom Transformer)

Aggregate atom-level representation to token-level

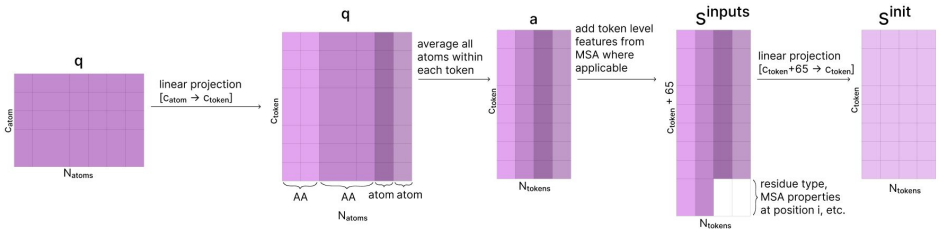
Aggregate Atom-Level → Token-Level



AF3 shifts from atom-level to token-level processing by projecting atom features (**q**) to higher dimensions, then averaging over atoms belonging to each standard amino acid or nucleotide. These become token-level inputs.

MSA-derived features are concatenated, forming **s_inputs**, which is projected to **s_init**, the initialized token representation.

Aggregate Atom-Level → Token-Level



 # Cross attention transformer.

15: $\{\mathbf{q}_l\} = \text{AtomTransformer}(\{\mathbf{q}_l\}, \{\mathbf{c}_l\}, \{\mathbf{p}_{lm}\}, N_{\text{block}} = 3, N_{\text{head}} = 4)$

 # Aggregate per-atom representation to per-token representation

16: $\mathbf{a}_i = \underset{\substack{l \in \{1,\ldots,N_{\text{atoms}}\} \\ \text{tok\_idx}(l)=i}}{\text{mean}} (\text{relu}(\text{LinearNoBias}(\mathbf{q}_l)))$

17: $\mathbf{q}_l^{\text{skip}}, \mathbf{c}_l^{\text{skip}}, \mathbf{p}_{lm}^{\text{skip}} = \mathbf{q}_l, \mathbf{c}_l, \mathbf{p}_{lm}$

18: **return** $\{\mathbf{a}_i\}, \{\mathbf{q}_l^{\text{skip}}\}, \{\mathbf{c}_l^{\text{skip}}\}, \{\mathbf{p}_{lm}^{\text{skip}}\}$

---

def $\text{InputFeatureEmbedder}(\{\mathbf{f}^*\})$:

 # Embed per-atom features.

1: $\{\mathbf{a}_i\}, \_, \_, \_ = \text{AtomAttentionEncoder}(\{\mathbf{f}^*\}, \emptyset, \emptyset, \emptyset, c_{\text{atom}} = 128, c_{\text{atompair}} = 16, c_{\text{token}} = 384)$

 # Concatenate the per-token features.

2: $\mathbf{s}_i = \text{concat}(\mathbf{a}_i, \mathbf{f}_i^{\text{restype}}, \mathbf{f}_i^{\text{profile}}, f_i^{\text{deletion\_mean}})$

3: **return** $\{\mathbf{s}_i\}$

---

def $\text{MainInferenceLoop}(\{\mathbf{f}^*\}, N_{\text{cycle}} = 4, c_{\text{s}} = 384, c_{\text{z}} = 128)$:

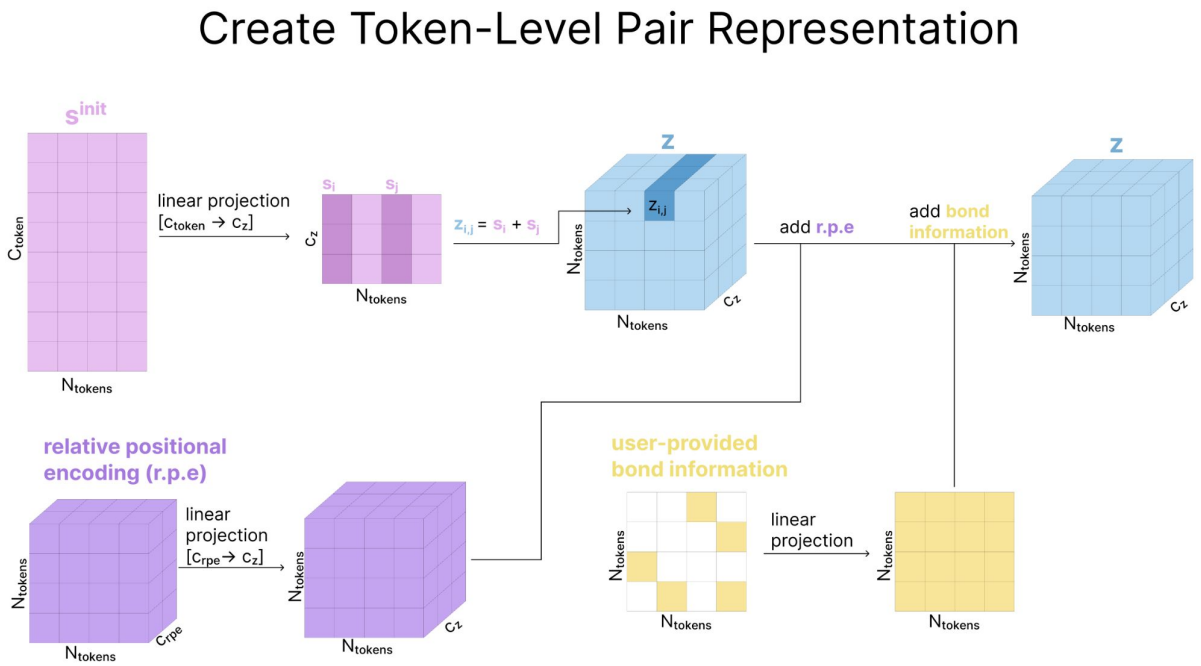
1: $\{\mathbf{s}_i^{\text{inputs}}\} = \text{InputFeatureEmbedder}(\{\mathbf{f}^*\})$

2: $\mathbf{s}_i^{\text{init}} = \text{LinearNoBias}(\mathbf{s}_i^{\text{inputs}})$

AF3 shifts from atom-level to token-level processing by projecting atom features (**q**) to higher dimensions, then averaging over atoms belonging to each standard amino acid or nucleotide. These become token-level inputs.

MSA-derived features are concatenated, forming **s_inputs**, which is projected to **s_init**, the initialized token representation.

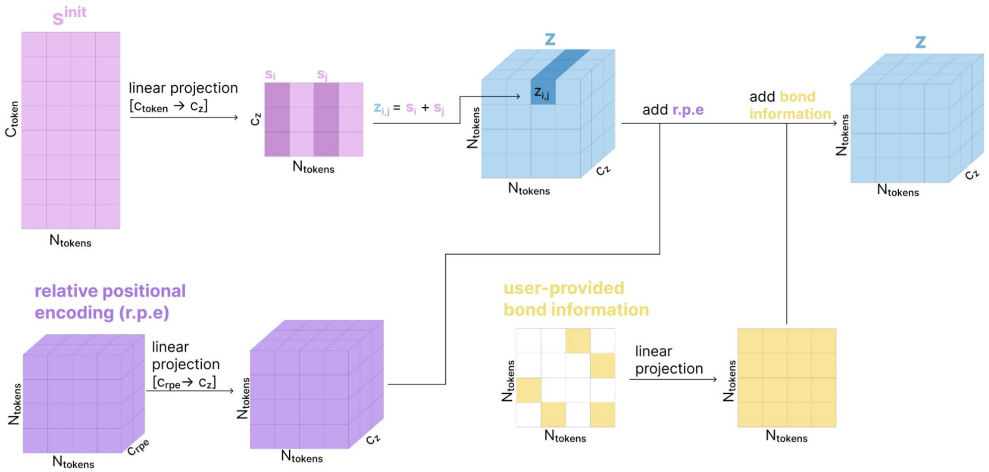
Aggregate Atom-Level → Token-Level



Token pair representations **z_init** (shape: tokens × tokens × 128) are initialized by summing projected token features (**si** + **sj**) with relative positional encodings and optional bond info.

Aggregate Atom-Level → Token-Level

Create Token-Level Pair Representation



$$\textbf{def}\ \text{MainInferenceLoop}(\{\mathbf{f}^*\}, N_{\text{cycle}} = 4, c_s = 384, c_z = 128):$$

1: $\{\mathbf{s}_i^{\text{inputs}}\} = \text{InputFeatureEmbedder}(\{\mathbf{f}^*\})$

2: $\mathbf{s}_i^{\text{init}} = \text{LinearNoBias}(\mathbf{s}_i^{\text{inputs}})$

3: $\mathbf{z}_{ij}^{\text{init}} = \text{LinearNoBias}(\mathbf{s}_i^{\text{inputs}}) + \text{LinearNoBias}(\mathbf{s}_j^{\text{inputs}})$

4: $\mathbf{z}_{ij}^{\text{init}} \mathrel{+}= \text{RelativePositionEncoding}(\{\mathbf{f}^*\})$

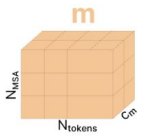
5: $\mathbf{z}_{ij}^{\text{init}} \mathrel{+}= \text{LinearNoBias}(\mathbf{f}_{ij}^{\text{token\_bonds}})$

Token pair representations **z_init** (shape: tokens × tokens × 128) are initialized by summing projected token features (**si** + **sj**) with relative positional encodings and optional bond info.

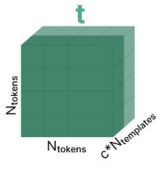
Aggregate Atom-Level → Token-LeveL

Information about related sequences and their structures

Multiple Sequence Alignment

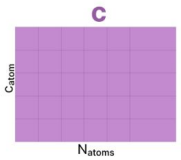


Structure Templates

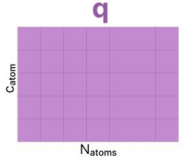


Information about all the atoms (“single”)

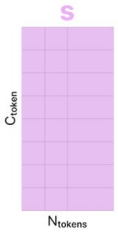
Original Atom-Level Single Representation



Updated Atom-Level Single Representation

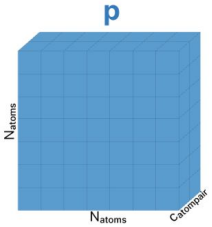


Token-Level Single Representation

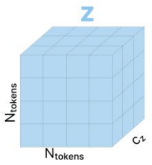


Information about all the pairs of atoms (“pair”)

Atom-Level Pair Representation

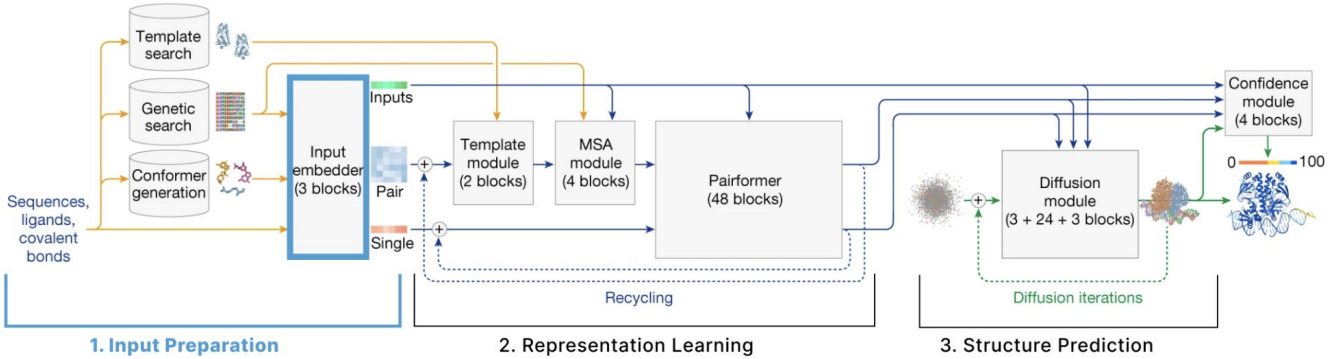


Token-Level Pair Representation



For Step 2, we will 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**).

1. Input Preparation Recap



Tokenize input sequences

Retrieve similar sequences and structures to create MSA and templates

Create atom-level representation of sequences

Update atom-level representation (Atom Transformer)

Aggregate atom-level representation to token-level

AtomTransformer introduces building blocks used elsewhere

* **Adaptive LayerNorm**
* **Attention with Pair Bias**
* **Sequence-local atom attention**
* **Conditioned Gating**
* **Conditioned Transition blocks**

End of Part 1

Thanks for listening!

Questions - Comments?

Algorithm 1 Main Inference Loop

def MainInferenceLoop($\{\mathbf{f}^*\}, N_{\text{cycle}} = 4, c_s = 384, c_z = 128$) :

$\{\mathbf{s}_i^{\text{inputs}}\} = \text{InputFeatureEmbedder}(\{\mathbf{f}^*\})$

$\mathbf{s}_i^{\text{init}} = \text{LinearNoBias}(\mathbf{s}_i^{\text{inputs}})$ $\quad \mathbf{s}_i^{\text{init}} \in \mathbb{R}^{c_s}$

$\mathbf{z}_{ij}^{\text{init}} = \text{LinearNoBias}(\mathbf{s}_i^{\text{inputs}}) + \text{LinearNoBias}(\mathbf{s}_j^{\text{inputs}})$ $\quad \mathbf{z}_{ij}^{\text{init}} \in \mathbb{R}^{c_z}$

$\mathbf{z}_{ij}^{\text{init}} \mathrel{+}= \text{RelativePositionEncoding}(\{\mathbf{f}^*\})$

$\mathbf{z}_{ij}^{\text{init}} \mathrel{+}= \text{LinearNoBias}(\mathbf{f}_{ij}^{\text{token\_bonds}})$

$\{\hat{\mathbf{z}}_{ij}\}, \{\hat{\mathbf{s}}_i\} = \mathbf{0}, \mathbf{0}$

for all $c \in [1, \ldots, N_{\text{cycle}}]$ **do**

  $\mathbf{z}_{ij} = \mathbf{z}_{ij}^{\text{init}} + \text{LinearNoBias}(\text{LayerNorm}(\hat{\mathbf{z}}_{ij}))$ $\quad \mathbf{z}_{ij} \in \mathbb{R}^{c_z}$

  $\{\mathbf{z}_{ij}\} \mathrel{+}= \text{TemplateEmbedder}(\{\mathbf{f}^*\}, \{\mathbf{z}_{ij}\})$

  $\{\mathbf{z}_{ij}\} \mathrel{+}= \text{MsaModule}(\{\mathbf{f}_{Si}^{\text{msa}}\}, \{\mathbf{z}_{ij}\}, \{\mathbf{s}_i^{\text{inputs}}\})$

  $\mathbf{s}_i = \mathbf{s}_i^{\text{init}} + \text{LinearNoBias}(\text{LayerNorm}(\hat{\mathbf{s}}_i))$ $\quad \mathbf{s}_i \in \mathbb{R}^{c_s}$

  $\{\mathbf{s}_i\}, \{\mathbf{z}_{ij}\} = \text{PairformerStack}(\{\mathbf{s}_i\}, \{\mathbf{z}_{ij}\})$

  $\{\hat{\mathbf{s}}_i\}, \{\hat{\mathbf{z}}_{ij}\} \leftarrow \{\mathbf{s}_i\}, \{\mathbf{z}_{ij}\}$

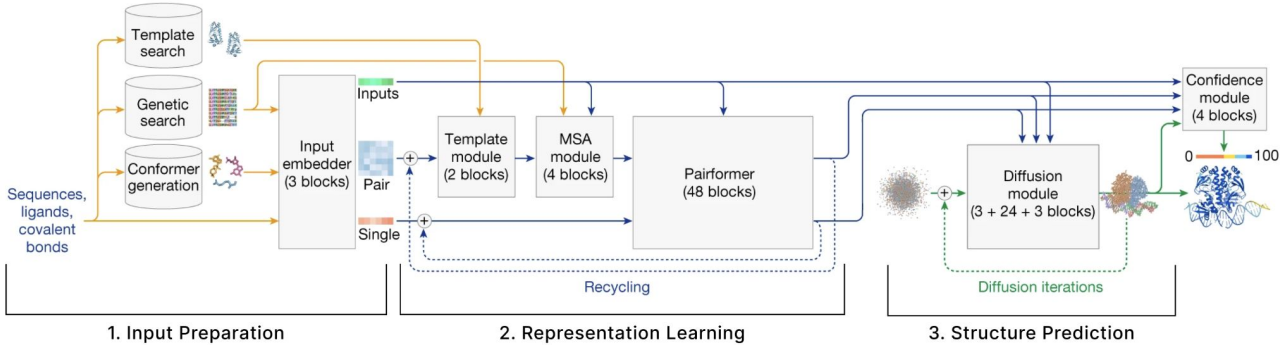
end for

$\{\vec{\mathbf{x}}_l^{\text{pred}}\} = \text{SampleDiffusion}(\{\mathbf{f}^*\}, \{\mathbf{s}_i^{\text{inputs}}\}, \{\mathbf{s}_i\}, \{\mathbf{z}_{ij}\})$

$\{\mathbf{p}_l^{\text{plddt}}\}, \{\mathbf{p}_{ij}^{\text{pae}}\}, \{\mathbf{p}_{ij}^{\text{pde}}\}, \{\mathbf{p}_l^{\text{resolved}}\} = \text{ConfidenceHead}(\{\mathbf{s}_i^{\text{inputs}}\}, \{\mathbf{s}_i\}, \{\mathbf{z}_{ij}\}, \{\vec{\mathbf{x}}_l^{\text{pred}}\})$

$\mathbf{p}_{ij}^{\text{distogram}} = \text{DistogramHead}(\mathbf{z}_{ij})$

return $\{\vec{\mathbf{x}}_l^{\text{pred}}\}, \{\mathbf{p}_l^{\text{plddt}}\}, \{\mathbf{p}_{ij}^{\text{pae}}\}, \{\mathbf{p}_{ij}^{\text{pde}}\}, \{\mathbf{p}_l^{\text{resolved}}\}, \{\mathbf{p}_{ij}^{\text{distogram}}\}$



Accurate structure prediction of biomolecular interactions with AlphaFold 3 (Part 2)

Abramson, J., Adler, J., Dunger, J. et al.

https://www.nature.com/articles/s41586-024-07487-w

Burak Suyunu

LifeLU Reading Group

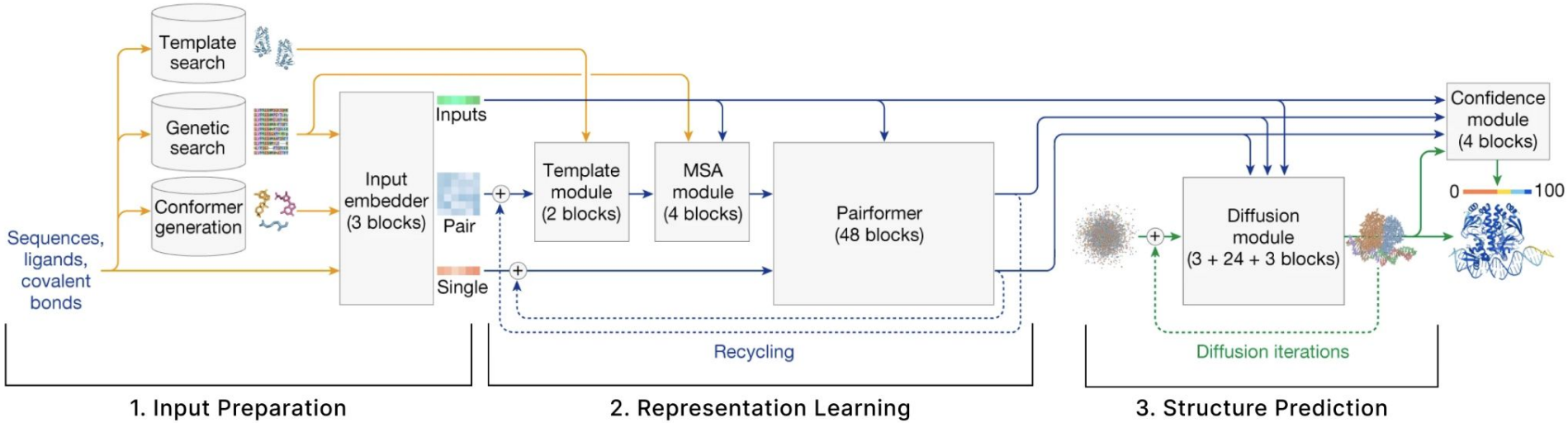
22.05.2025

Previously on Part 1

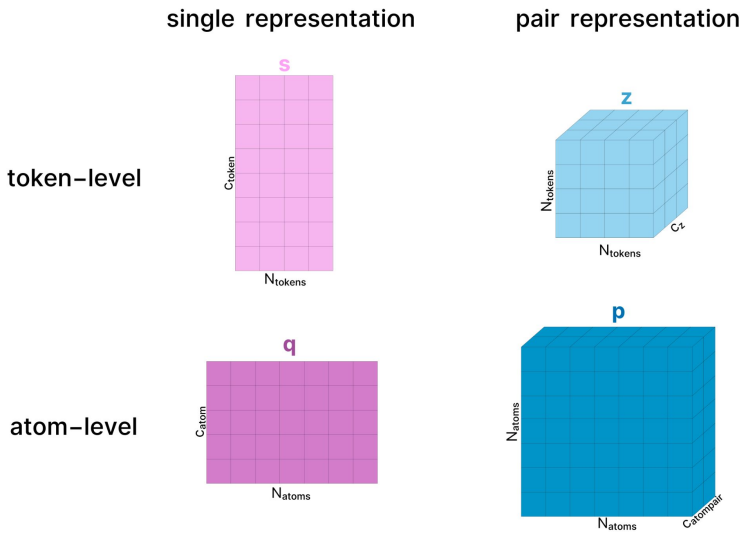
From AlphaFold2 to AlphaFold3 – A Paradigm Shift

Feature	AlphaFold2 (2021)	AlphaFold-Multimer (2021)	AlphaFold3 (2024)
Input types	Protein sequences	Protein sequences	Proteins, RNA, DNA, ligands
Complex prediction	No	Yes (proteins only)	Yes (all molecule types)
Ligand/RNA/DNA modeling	❌	❌	✅
Underlying architecture	Evoformer + structure module	Extended Evoformer	**Pairformer + Diffusion**
Training task	Supervised (PDB structures)	Supervised (PDB structures)	**Conditional diffusion**

Architecture Overview



Notes on the variables and diagrams



AlphaFold 3 represents protein complexes using two key tensors: **Single** (per-token/atom features) and **Pair** (relationships between token/atom pairs). These can be at the token or atom level, shown with consistent names and colors.

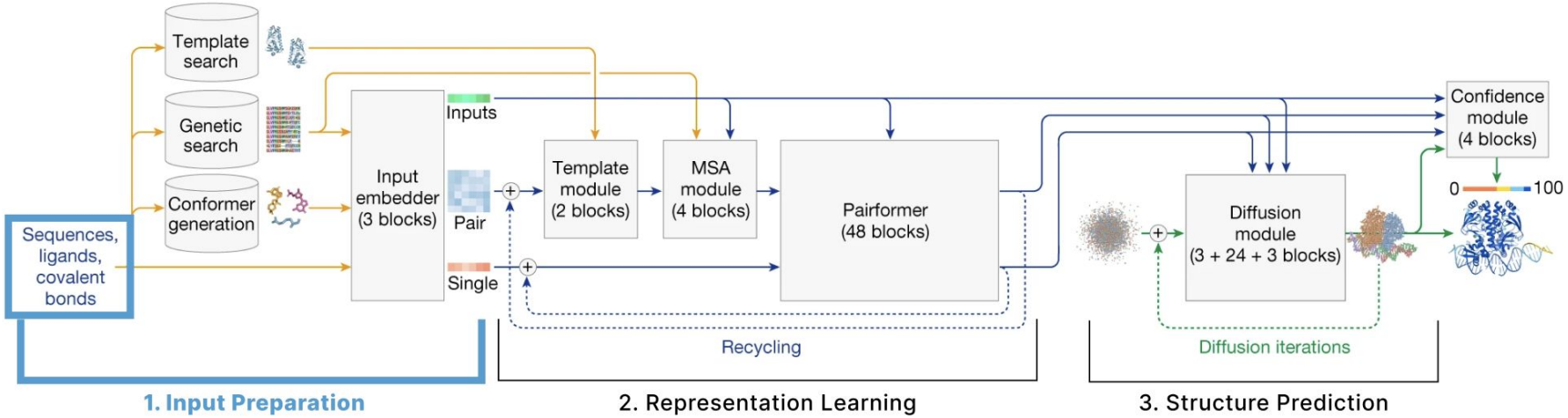
Diagrams show how activation shapes change—not model weights. Tensor labels match those in the AF3 paper. Names are mostly preserved, but some (e.g., c to q in atom-level single) indicate updates through processing. LayerNorms are used throughout but mostly omitted for clarity.

Input Features

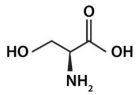
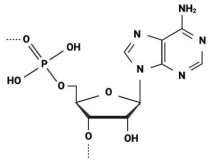
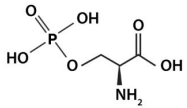
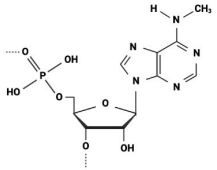
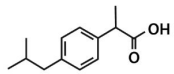
Feature & Shape	Description
residue_index $[N_{\text{token}}]$	Residue number in the token's original input chain.
token_index $[N_{\text{token}}]$	Token number. Increases monotonically; does not restart at 1 for new chains.
asym_id $[N_{\text{token}}]$	Unique integer for each distinct chain.
entity_id $[N_{\text{token}}]$	Unique integer for each distinct sequence.
sym_id $[N_{\text{token}}]$	Unique integer within chains of this sequence. E.g. if chains A, B and C share a sequence but D does not, their sym_ids would be [0, 1, 2, 0].
restype $[N_{\text{token}}, 32]$	One-hot encoding of the sequence. 32 possible values: 20 amino acids + unknown, 4 RNA nucleotides + unknown, 4 DNA nucleotides + unknown, and gap. Ligands represented as "unknown amino acid".
is_protein / rna / dna / ligand $[N_{\text{token}}]$	4 masks indicating the molecule type of a particular token.
ref_pos $[N_{\text{atom}}, 3]$	Atom positions in the reference conformer, with a random rotation and translation applied. Atom positions are given in Å.
ref_mask $[N_{\text{atom}}]$	Mask indicating which atom slots are used in the reference conformer.
ref_element $[N_{\text{atom}}, 128]$	One-hot encoding of the element atomic number for each atom in the reference conformer, up to atomic number 128.
ref_charge $[N_{\text{atom}}]$	Charge for each atom in the reference conformer.

Feature & Shape	Description
ref_atom_name_chars $[N_{\text{atom}}, 4, 64]$	One-hot encoding of the unique atom names in the reference conformer. Each character is encoded as $\text{ord}(c) - 32$, and names are padded to length 4.
ref_space_uid $[N_{\text{atom}}]$	Numerical encoding of the chain id and residue index associated with this reference conformer. Each (chain id, residue index) tuple is assigned an integer on first appearance.
msa $[N_{\text{msa}}, N_{\text{token}}, 32]$	One-hot encoding of the processed MSA, using the same classes as restype.
has_deletion $[N_{\text{msa}}, N_{\text{token}}]$	Binary feature indicating if there is a deletion to the left of each position in the MSA.
deletion_value $[N_{\text{msa}}, N_{\text{token}}]$	Raw deletion counts (the number of deletions to the left of each MSA position) are transformed to $[0, 1]$ using $\frac{2}{\pi} \arctan \frac{d}{3}$.
profile $[N_{\text{token}}, 32]$	Distribution across restypes in the main MSA. Computed before MSA processing (subsection 2.3).
deletion_mean $[N_{\text{token}}]$	Mean number of deletions at each position in the main MSA. Computed before MSA processing (subsection 2.3).
template_restype $[N_{\text{templ}}, N_{\text{token}}]$	One-hot encoding of the template sequence, see restype.
template_pseudo_beta_mask $[N_{\text{templ}}, N_{\text{token}}]$	Mask indicating if the C^{β} (C^{α} for glycine) has coordinates for the template at this residue.
template_backbone_frame_mask $[N_{\text{templ}}, N_{\text{token}}]$	Mask indicating if coordinates exist for all atoms required to compute the backbone frame (used in the template_unit_vector feature).
template_distogram $[N_{\text{templ}}, N_{\text{token}}, N_{\text{token}}, 39]$	A one-hot pairwise feature indicating the distance between C^{β} atoms (C^{α} for glycine). Pairwise distances are discretized into 38 bins of equal width between 3.25 Å and 50.75 Å; one more bin contains any larger distances.
template_unit_vector $[N_{\text{templ}}, N_{\text{token}}, N_{\text{token}}, 3]$	The unit vector of the displacement of the C^{α} atom of all residues within the local frame of each residue. Local frames are computed as in [1].
token_bonds $[N_{\text{token}}, N_{\text{token}}]$	A 2D matrix indicating if there is a bond between any atom in token i and token j, restricted to just polymer-ligand and ligand-ligand bonds and bonds less than 2.4 Å during training.

Tokenization

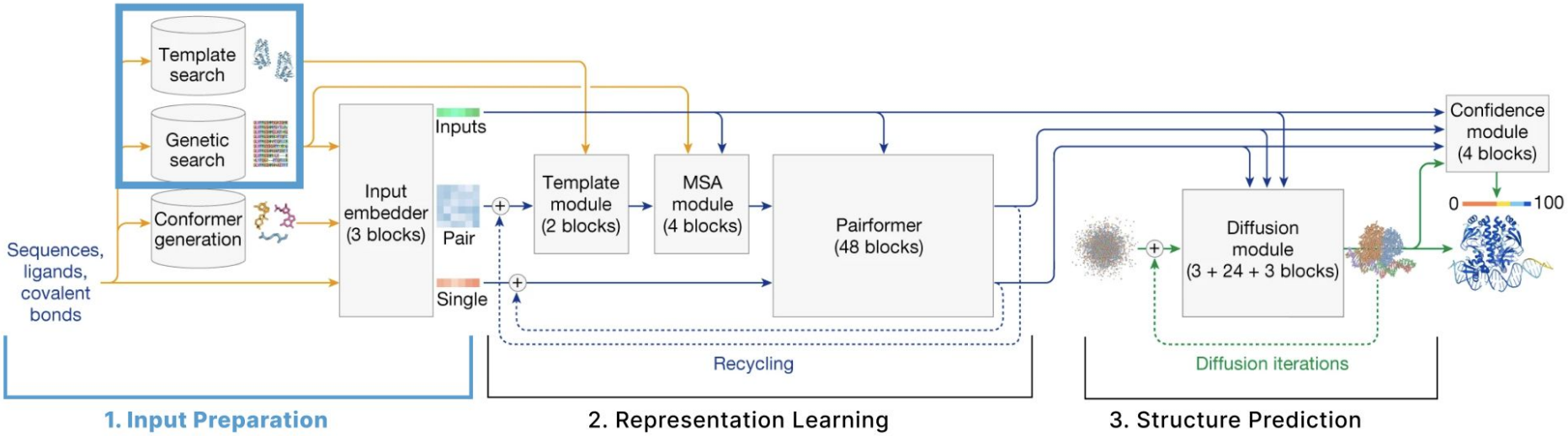


Tokenize input sequences

Standard Amino Acid (serine)	**Standard Nucleotide** (adenosine)	**Modified Amino Acid** (phosphoserine)	**Modified Nucleotide** (methyladenosine)	**Ligand** (ibuprofen)
				
7 atoms* 1 tokens	**23 atoms* 1 token**	**11 atoms* 11 tokens**	**24 atoms* 24 tokens**	**15 atoms* 15 tokens**

*atoms=heavy atoms

Retrieval (Create MSA and Templates)



Tokenize input sequences

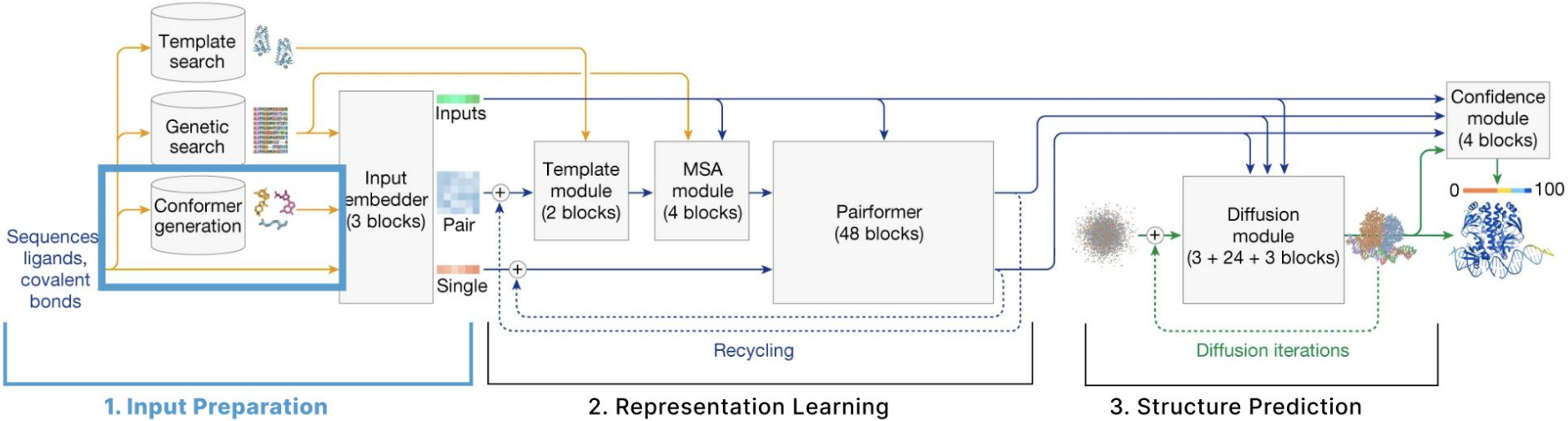
Retrieve similar sequences and structures to create MSA and templates

Create atom-level representation of sequences

Update atom-level representation (Atom Transformer)

Aggregate atom-level representation to token-level

Create Atom-Level Representations



Tokenize input sequences

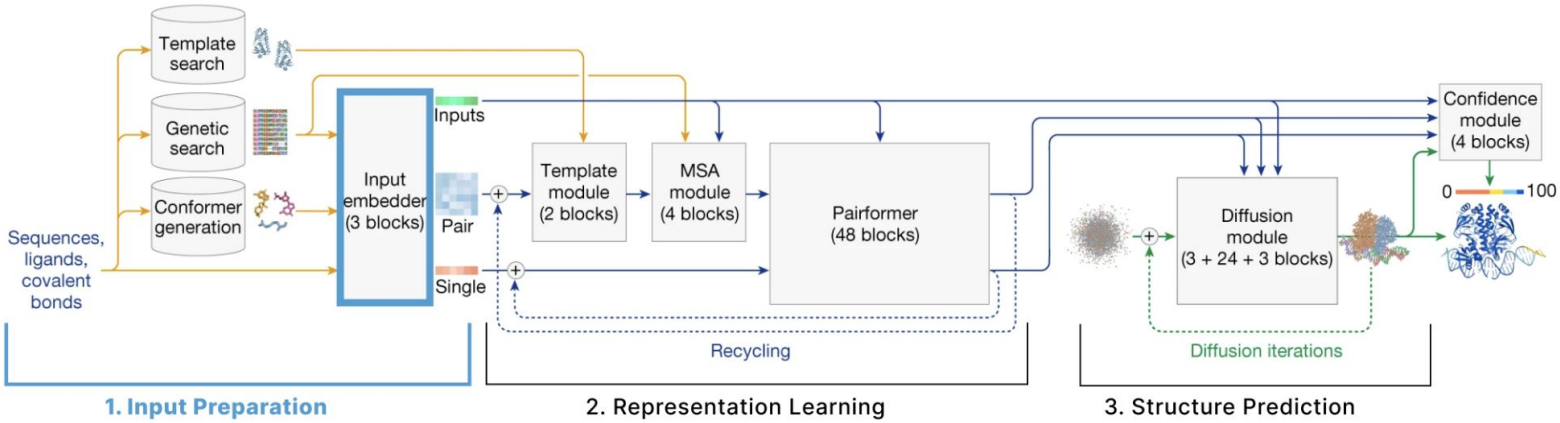
Retrieve similar sequences and structures to create MSA and templates

Create atom-level representation of sequences

Update atom-level representation (Atom Transformer)

Aggregate atom-level representation to token-level

Update Atom-Level Representations (Atom Transformer)



Tokenize input sequences

Retrieve similar sequences and structures to create MSA and templates

Create atom-level representation of sequences

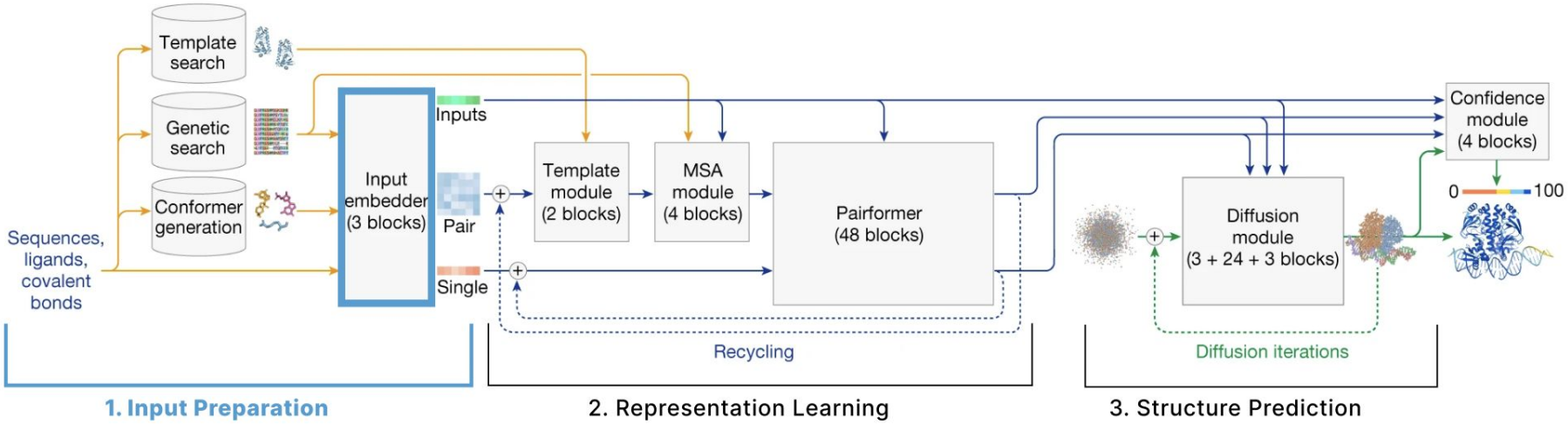
Update atom-level representation (Atom Transformer)

Aggregate atom-level representation to token-level

AtomTransformer introduces building blocks used elsewhere

* **Adaptive LayerNorm**
* **Attention with Pair Bias**
* **Sequence-local atom attention**
* **Conditioned Gating**
* **Conditioned Transition blocks**

Aggregate Atom-Level → Token-Level



Tokenize input sequences

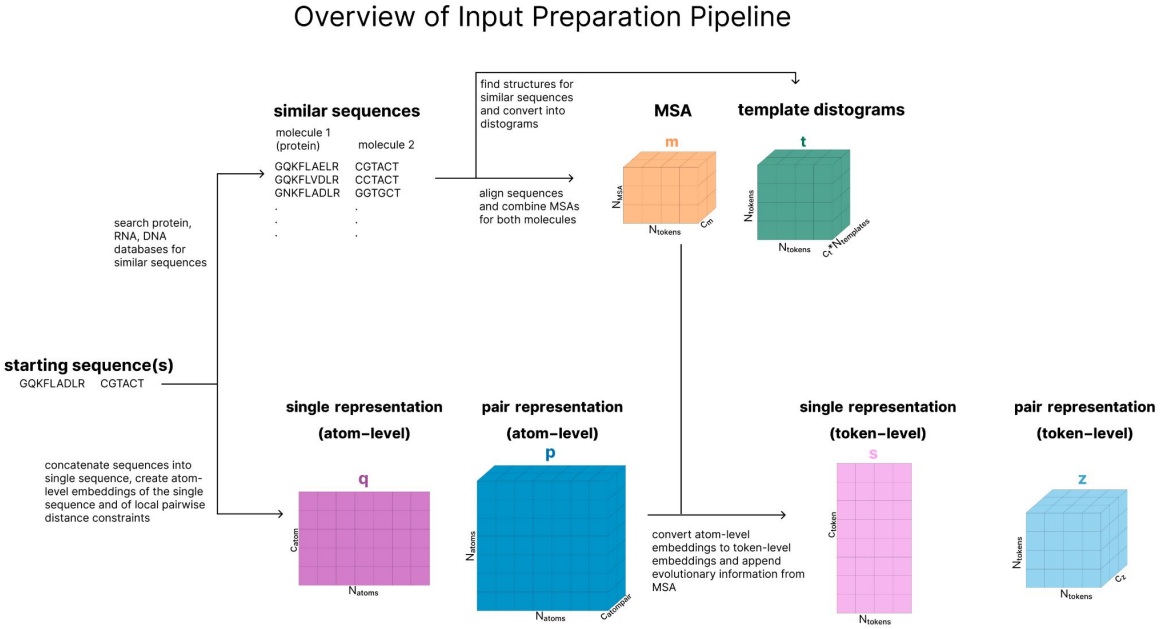
Retrieve similar sequences and structures to create MSA and templates

Create atom-level representation of sequences

Update atom-level representation (Atom Transformer)

Aggregate atom-level representation to token-level

Input Preparation Pipeline



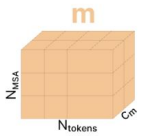
The user provides a protein sequence and optionally other molecules. This section turns inputs into 6 key tensors for the model: **s** (token-level single), **z** (token-level pair), **q** (atom-level single), **p** (atom-level pair), **m** (MSA), and **t** (template). Steps include:

- **Tokenization**: Molecules are split into tokens (e.g., amino acids, atoms); distinguishes atom-level vs. token-level representations.
- **Retrieval (MSA & Templates)**: Uses external databases to find similar sequences (MSA, **m**) and structural templates (**t**) to guide prediction.
- **Create Atom-Level Representations**: Builds **q** and **p** from 3D conformers; encodes atom-level features and inter-atomic relationships.
- **Update Atom-Level Representations**: The Atom Transformer updates **q** using custom attention, LayerNorm, gating, and transition modules.
- **Aggregate Atom- to Token-Level**: Aggregates atom-level **q** and **p** into token-level **s** and **z**, incorporating MSA info and known ligand bonding data.

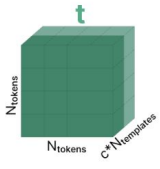
Aggregate Atom-Level → Token-LeveL

Information about related sequences and their structures

Multiple Sequence Alignment

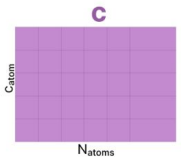


Structure Templates

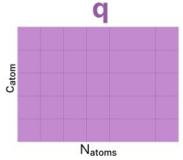


Information about all the atoms (“single”)

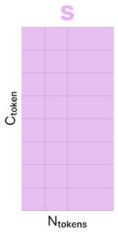
Original Atom-Level Single Representation



Updated Atom-Level Single Representation

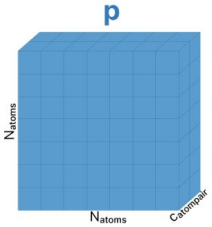


Token-Level Single Representation

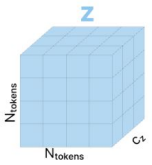


Information about all the pairs of atoms (“pair”)

Atom-Level Pair Representation



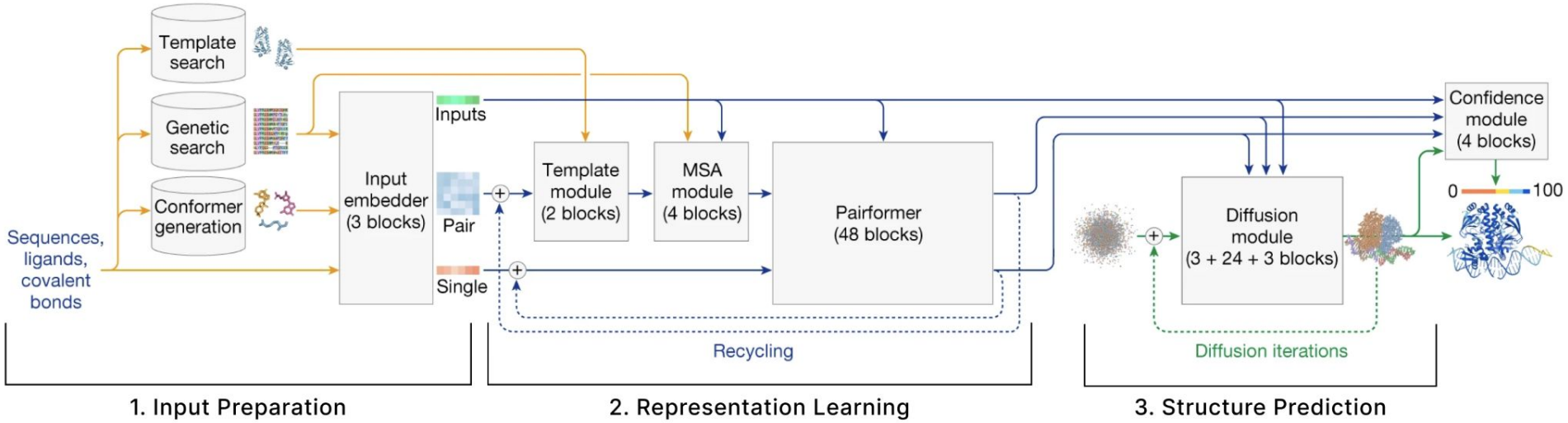
Token-Level Pair Representation



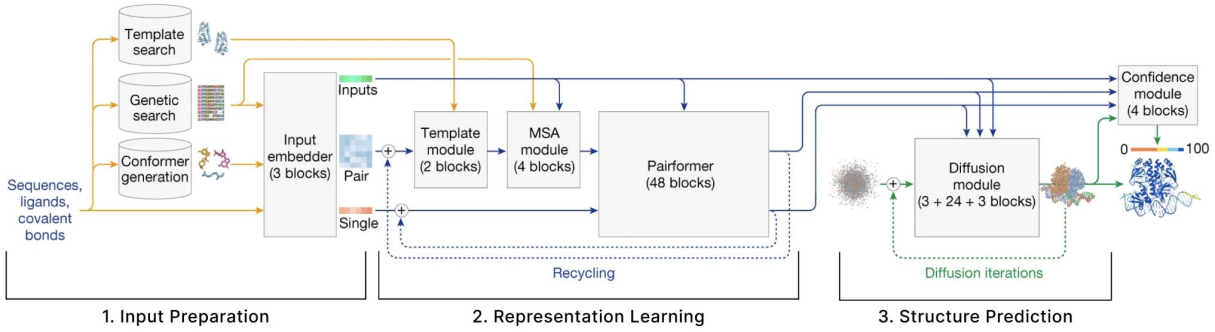
For Step 2, we will set aside the atom-level representations (**c**, **q**, **p**) and focus on updating our token-level representations **s** and **z** in the next section (with the help of **m** and **t**).

Let’s Start Part 2

2. Representation Learning



Representation Learning ("Trunk") Overview

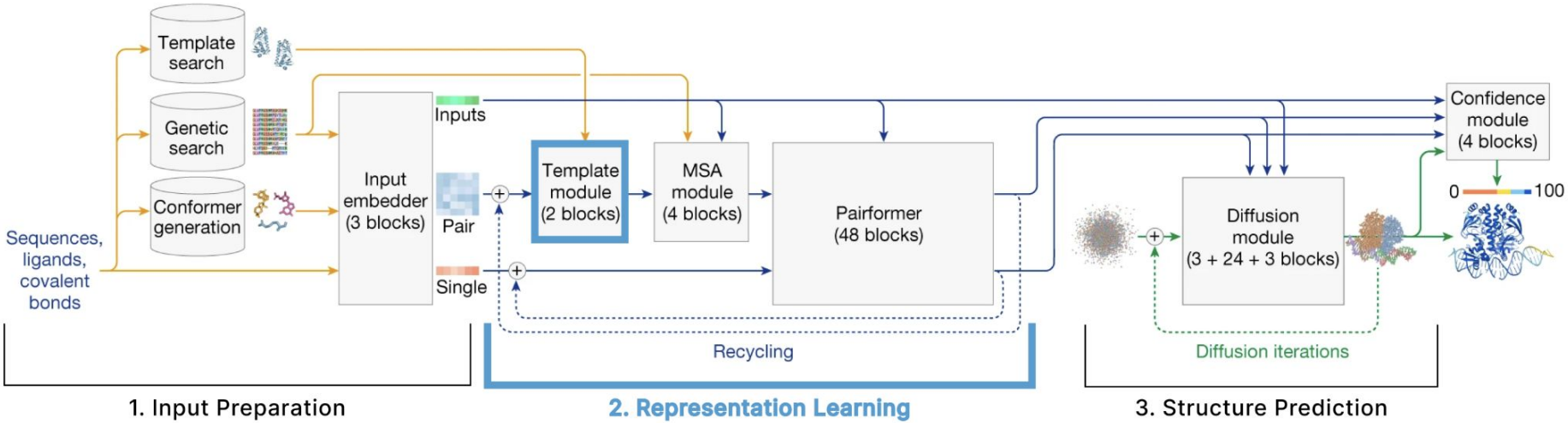


This is the main computational block, where token-level single (**s**) and pair (**z**) representations are iteratively refined. It includes:

- **Template Module**: Updates **z** using distance and identity features from structure templates (**t**).
- **MSA Module**: First updates the MSA representation (**m**), then incorporates it into **z**:
    - **Outer Product Mean**: Captures co-evolution by projecting interactions in **m** into **z**.
    - **MSA Row-wise Gated Self-Attention w/ Pair Bias**: Updates **m** using information from **z**, applying attention gated by **z** to reduce complexity.
- **Pairformer**: Refines **s** and **z** using geometric reasoning based on triangle relationships:
    - **Triangle Updates & Triangle Attention**: Two attention mechanisms updating **z**, inspired by the triangle inequality and structural motifs.
    - **Why Triangles?**: Models indirect interactions (i → j → k) crucial in 3D structures.
    - **Single Attention with Pair Bias**: Updates **s** using information in **z**, letting each token attend more to structurally important partners.

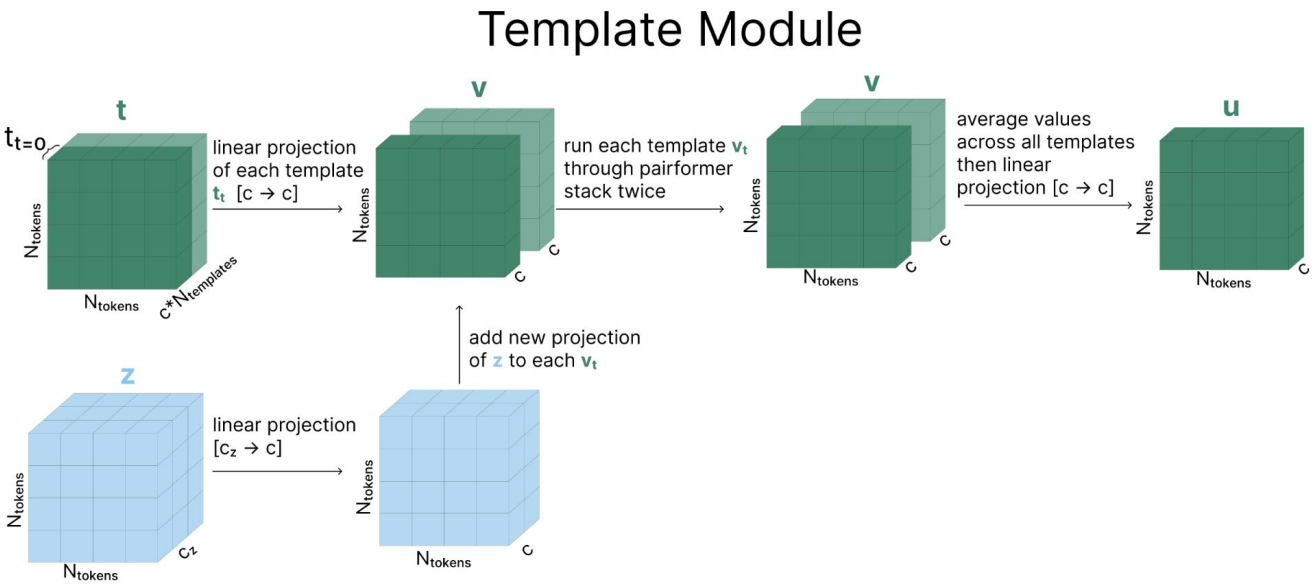
These modules are stacked and **recycled** multiple times, allowing the model to iteratively improve its internal structural representations.

Template Module



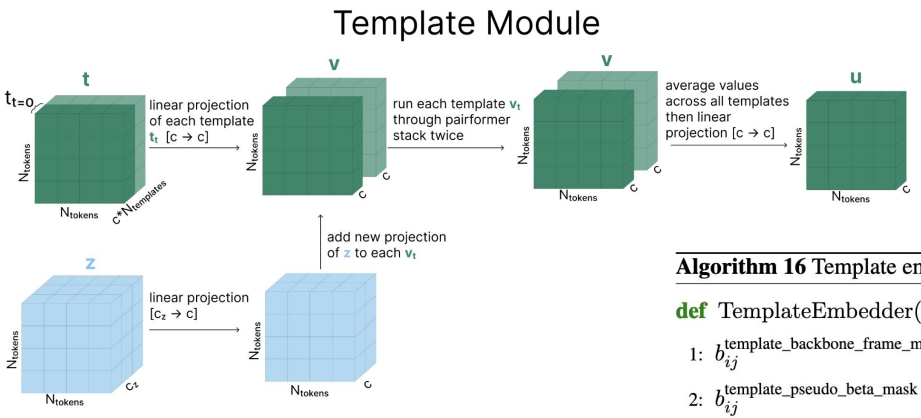
Template Module MSA module Pairformer

Template Module



- Each of the input templates (e.g., 2) is linearly projected and added to a projection of the pair representation **z**.
- The result is passed through a **Pairformer Stack** to update structural relationships.
- Outputs from all templates are averaged and passed through a **ReLU-activated linear layer** (one of only two ReLUs in AF3).
- This module injects structural priors from templates into the pairwise representation.

Template Module



Algorithm 16 Template embedder

def $\text{TemplateEmbedder}(\{\mathbf{f}^*\}, \{\mathbf{z}_{ij}\}, N_{\text{block}} = 2, c = 64)$:

$b_{ij}^{\text{template\_backbone\_frame\_mask}} = \mathrm{f}_{ti}^{\text{template\_backbone\_frame\_mask}} \cdot \mathrm{f}_{tj}^{\text{template\_backbone\_frame\_mask}}$

$b_{ij}^{\text{template\_pseudo\_beta\_mask}} = \mathrm{f}_{ti}^{\text{template\_pseudo\_beta\_mask}} \cdot \mathrm{f}_{tj}^{\text{template\_pseudo\_beta\_mask}}$

$\mathbf{a}_{tij} = \text{concat}(\mathbf{f}_{tij}^{\text{template\_distogram}}, b_{ij}^{\text{template\_backbone\_frame\_mask}}, \mathbf{f}_{tij}^{\text{template\_unit\_vector}}, b_{ij}^{\text{template\_pseudo\_beta\_mask}})$

$\mathbf{a}_{tij} \leftarrow \mathbf{a}_{tij} \odot (\mathrm{f}_i^{\text{asym\_id}} == \mathrm{f}_j^{\text{asym\_id}})$

$\mathbf{a}_{tij} \leftarrow \text{concat}(\mathbf{a}_{tij}, \mathbf{f}_{ti}^{\text{template\_restype}}, \mathbf{f}_{tj}^{\text{template\_restype}})$

$\mathbf{u}_{ij} = \mathbf{0}$ $\quad \mathbf{u}_{ij} \in \mathbb{R}^c$

for all $t \in [1, \ldots, N_{\text{templates}}]$ **do**

  $\mathbf{v}_{ij} = \text{LinearNoBias}(\text{LayerNorm}(\mathbf{z}_{ij})) + \text{LinearNoBias}(\mathbf{a}_{tij})$ $\quad \mathbf{v}_{ij} \in \mathbb{R}^c$

  $\{\mathbf{v}_{ij}\}$ += $\text{PairformerStack}(\{\mathbf{v}_{ij}\}, N_{\text{block}})$

  $\mathbf{u}_{ij}$ += $\text{LayerNorm}(\mathbf{v}_{ij})$

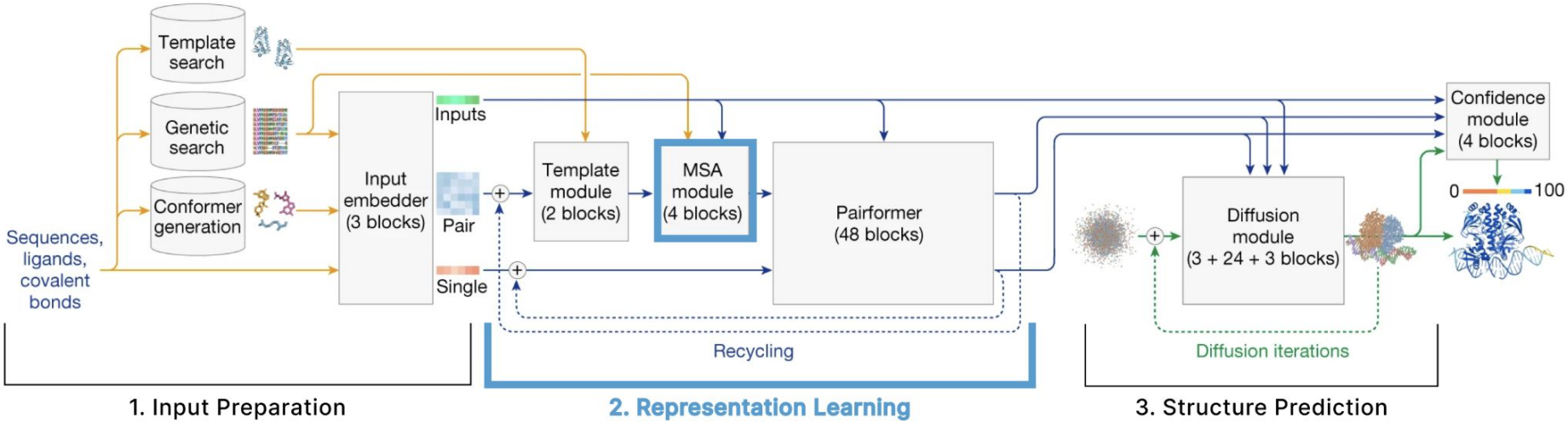
end for

$\mathbf{u}_{ij} \leftarrow \frac{\mathbf{u}_{ij}}{N_{\text{templates}}}$

$\mathbf{u}_{ij} \leftarrow \text{LinearNoBias}(\text{ReLU}(\mathbf{u}_{ij}))$ $\quad \mathbf{u}_{ij} \in \mathbb{R}^c$

return $\{\mathbf{u}_{ij}\}$

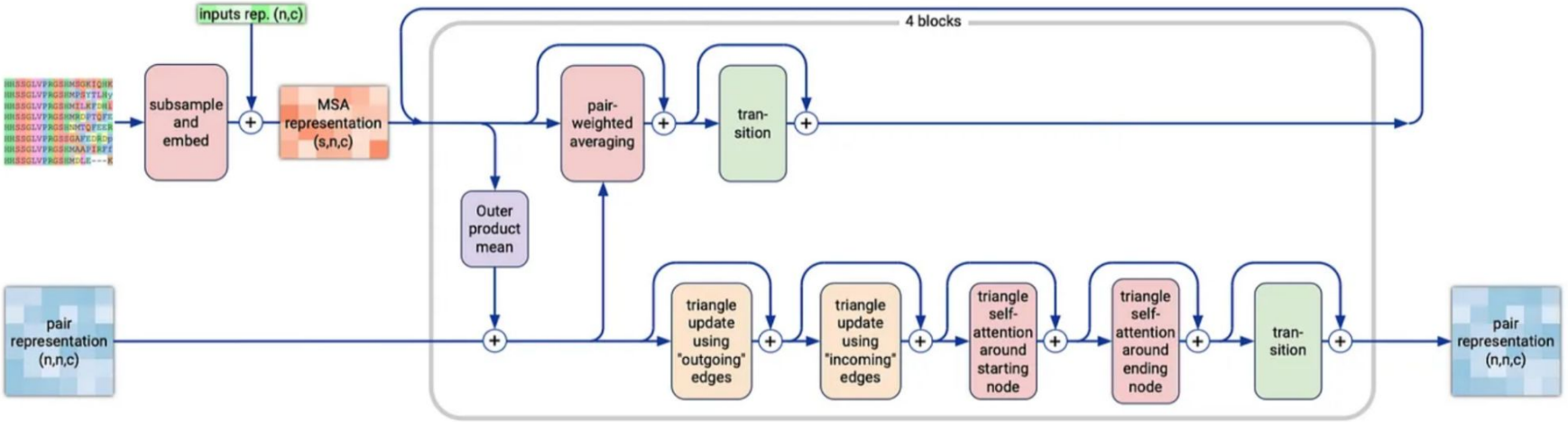
MSA Module



Template Module **MSA module** Pairformer

- **Outer product mean**
- **Row-wise gated attention using only pair bias**

MSA Module

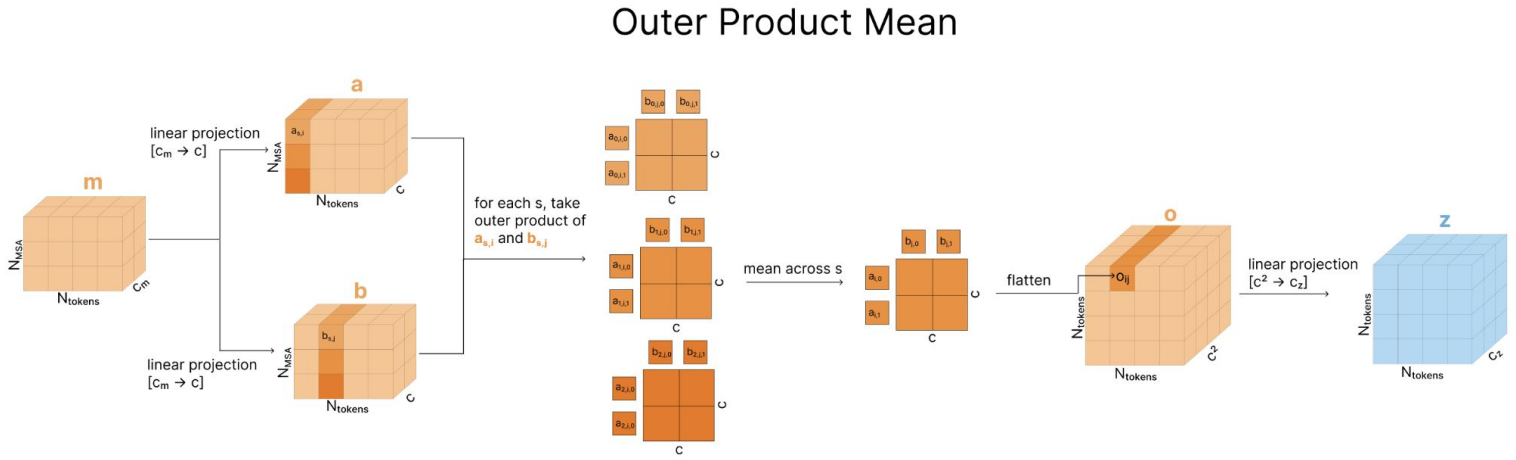


The MSA Module is the evolutionary reasoning engine of the model. It resembles the **Evoformer** from AlphaFold 2 but is optimized for speed and scalability by limiting cross-sequence operations.

➢ **Step 1: MSA Subsampling and Initialization**

- Instead of using the full MSA (which may have **up to 16k sequences**), AlphaFold 3 subsamples it to a smaller, computationally manageable set.
- The token-level single representation **s** is projected and **added to each row of the subsampled MSA**, seeding each sequence with positional context.

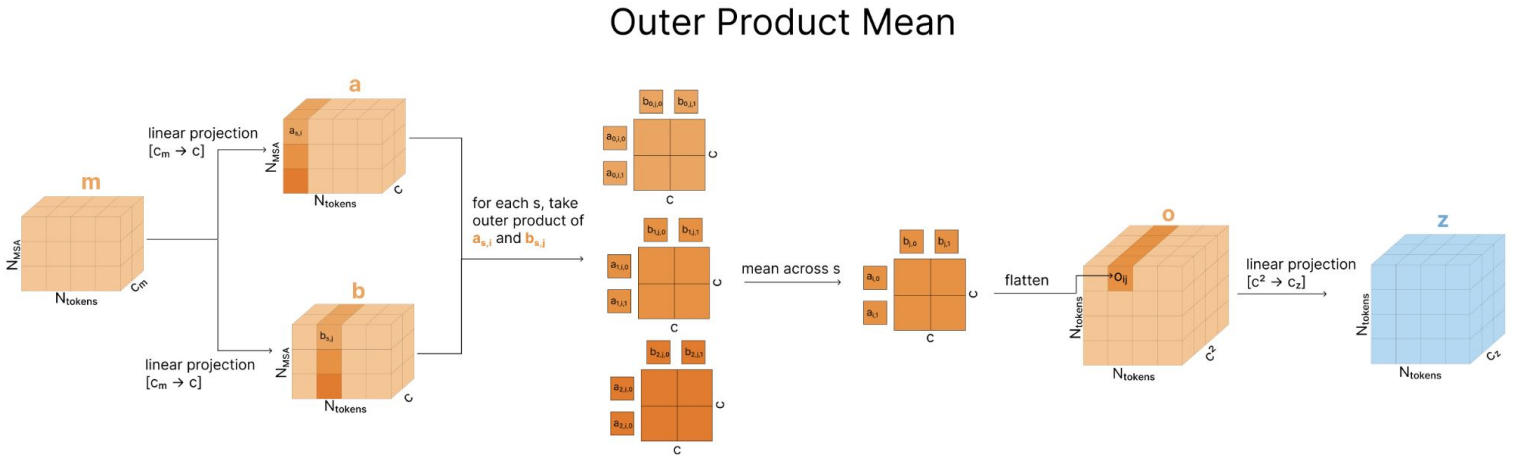
MSA Module - Outer Product Mean



Step 2: Outer Product Mean (MSA → z)

- This step injects evolutionary coupling signals from the MSA into the pair representation **z**:
  - For each sequence and token pair (i, j), compute the **outer product** of their MSA embeddings: `outer(ms,i, ms,j)`.
  - Average this across all MSA sequences to get a single feature per token pair.
  - Flatten and project this tensor back down to the pair channel size (e.g., cz = 128).
  - **Add to z_i,j**
- Crucially, **this is the only point** where cross-sequence information from the MSA is mixed — reducing memory costs compared to AF2.

MSA Module - Outer Product Mean



Algorithm 9 Outer product mean

def $\text{OuterProductMean}(\{\mathbf{m}_{si}\}, c = 32, c_z = 128)$:

1: $\mathbf{m}_{si} \leftarrow \text{LayerNorm}(\mathbf{m}_{si})$

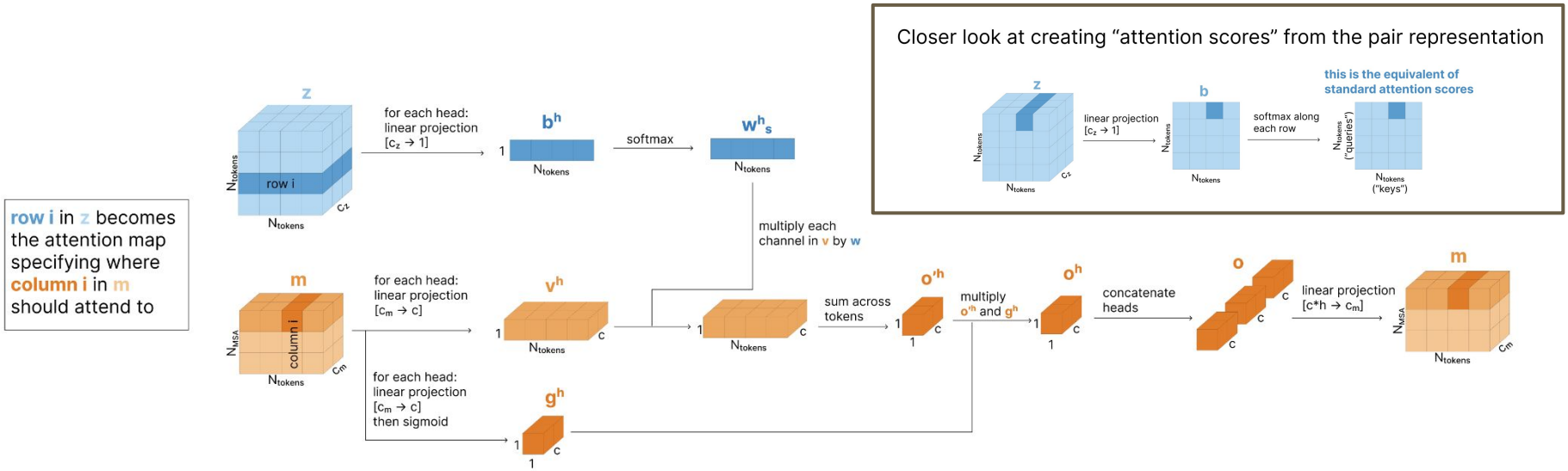
2: $\mathbf{a}_{si}, \mathbf{b}_{si} = \text{LinearNoBias}(\mathbf{m}_{si})$

3: $\mathbf{o}_{ij} = \text{flatten}\left(\text{mean}_s(\mathbf{a}_{si} \otimes \mathbf{b}_{sj})\right)$

4: $\mathbf{z}_{ij} = \text{Linear}(\mathbf{o}_{ij})$

5: **return** $\{\mathbf{z}_{ij}\}$

MSA Module - Row-wise gated self-attention using only pair bias

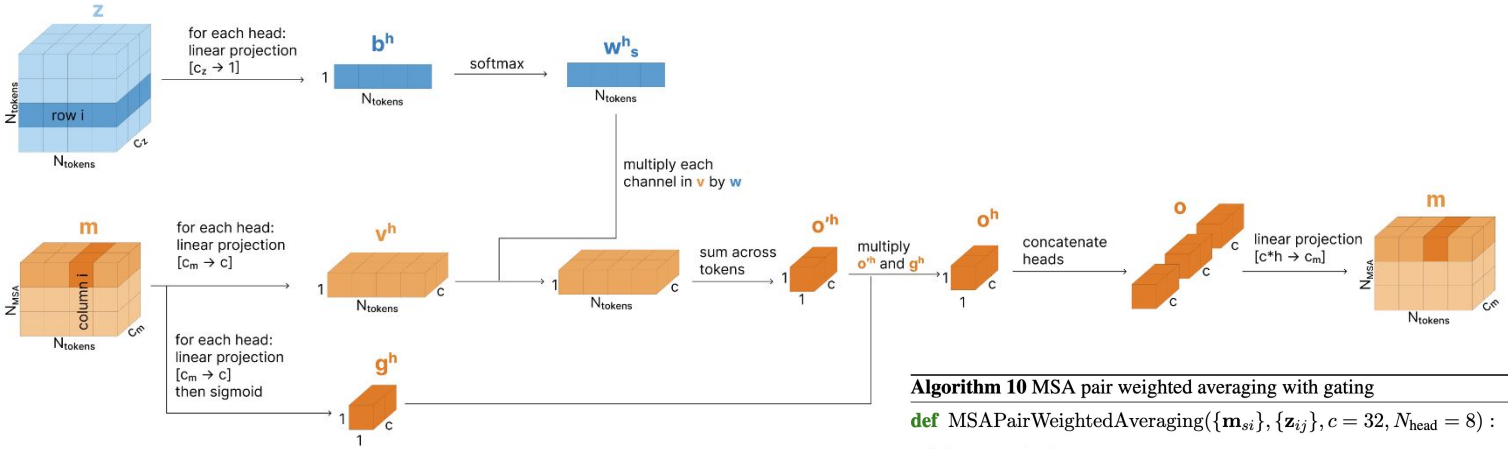


➢ **Step 3: Row-wise Gated Self-Attention Using Only Pair Bias (z → m)**

- Updates each row of the MSA independently based on the current pairwise representation **z**:
  - Rather than using queries/keys as in standard attention, **z_i,j** directly biases attention scores.
  - Each z_i,j is projected to a scalar and passed through a **row-wise softmax** to act as attention weights.
  - The resulting scores are used to aggregate value vectors within each MSA row.
  - A **gating mechanism** (sigmoid) controls how much of the attention output is retained.
- Note: **No information is exchanged across different MSA rows** (i.e., no inter-sequence communication here).

MSA Module - Row-wise gated self-attention using only pair bias

row i in z becomes the attention map specifying where column i in m should attend to



Algorithm 10 MSA pair weighted averaging with gating

def MSAPairWeightedAveraging($\{\mathbf{m}_{si}\}, \{\mathbf{z}_{ij}\}, c = 32, N_{\text{head}} = 8$) :

 # Input projections

$\mathbf{m}_{si} \leftarrow \text{LayerNorm}(\mathbf{m}_{si})$

$\mathbf{v}^h_{si} = \text{LinearNoBias}(\mathbf{m}_{si})$

$b^h_{ij} = \text{LinearNoBias}(\text{LayerNorm}(\mathbf{z}_{ij}))$

$\mathbf{g}^h_{si} = \text{sigmoid}\left(\text{LinearNoBias}(\mathbf{m}_{si})\right)$

 # Weighted average with gating

$w^h_{ij} = \text{softmax}_j\left(b^h_{ij}\right)$

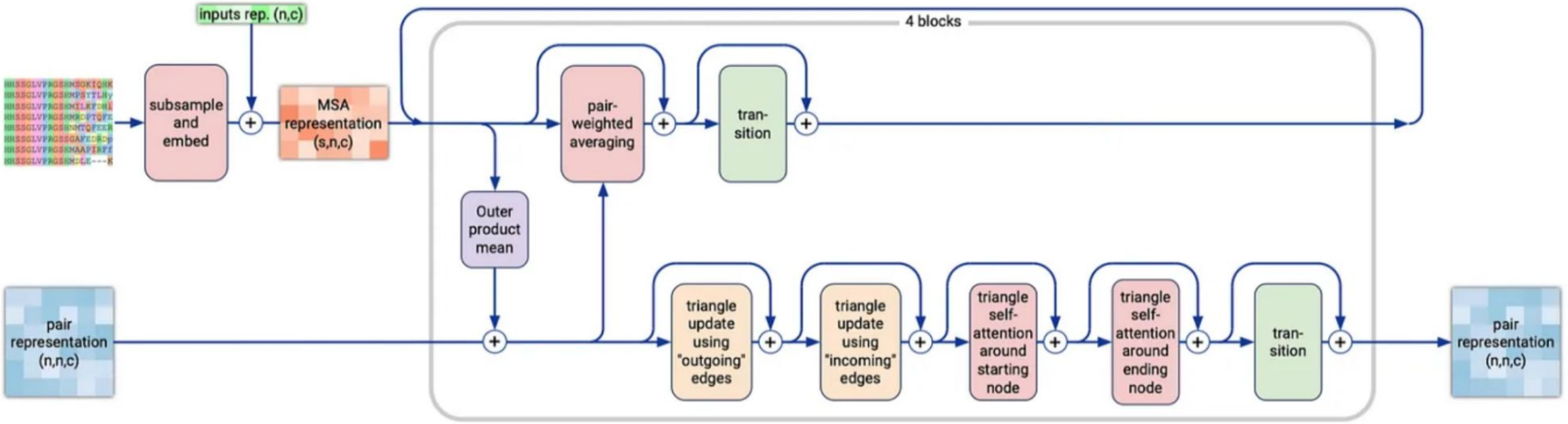
$\mathbf{o}^h_{si} = \mathbf{g}^h_{si} \odot \sum_j w^h_{ij} \mathbf{v}^h_{sj}$

 # Output projection

$\tilde{\mathbf{m}}_{si} = \text{LinearNoBias}\left(\text{concat}_h(\mathbf{o}^h_{si})\right)$

return $\{\tilde{\mathbf{m}}_{si}\}$

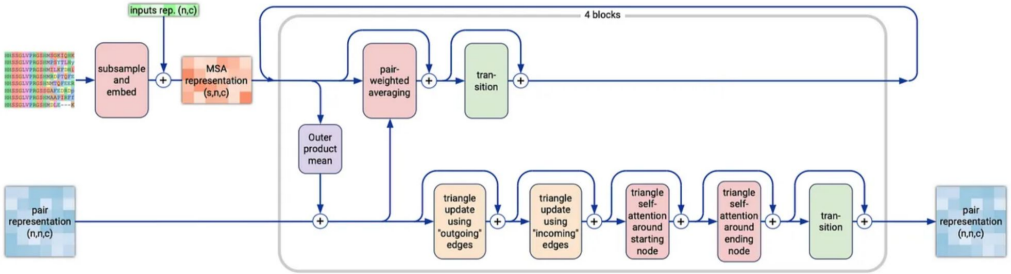
MSA Module - Updates to pair representation



➢ **Step 4: Triangle Updates and SwiGLU Transitions (m + z)**

- After mutual updates between **m** and **z**, a final refinement to **z** is performed:
  - **Triangle Multiplication, Triangle Attention** — use geometric reasoning to propagate indirect residue relationships (e.g., if i interacts with j and j with k, then i and k may interact).
  - **SwiGLU MLPs** refine feature dimensions post-attention.

MSA Module



Algorithm 8 MSA Module

def MsaModule($\{\mathbf{f}^*\}, \{\mathbf{z}_{ij}\}, \{\mathbf{s}_i^{\text{inputs}}\}, N_{\text{block}} = 4, c_m = 64$) :

$\mathbf{m}_{Si} = \text{concat}(\mathbf{f}_{Si}^{\text{msa}}, \mathbf{f}_{Si}^{\text{has\_deletion}}, \mathbf{f}_{Si}^{\text{deletion\_value}})$

$\{s\} = \text{SampleRandomWithoutReplacement}(\{S\})$

$\mathbf{m}_{si} \leftarrow \text{LinearNoBias}(\mathbf{m}_{si})$

$\mathbf{m}_{si}$ += $\text{LinearNoBias}(\{\mathbf{s}_i^{\text{inputs}}\})$

for all $l \in [1, \ldots, N_{\text{block}}]$ **do**

\# *Communication*

$\{\mathbf{z}_{ij}\}$ += $\text{OuterProductMean}(\{\mathbf{m}_{si}\})$

\# *MSA stack*

$\{\mathbf{m}_{si}\}$ += $\text{DropoutRowwise}_{0.15}(\text{MSAPairWeightedAveraging}(\{\mathbf{m}_{si}\}, \{\mathbf{z}_{ij}\}, c = 8))$

$\{\mathbf{m}_{si}\}$ += $\text{Transition}(\{\mathbf{m}_{si}\})$

\# *Pair stack*

$\{\mathbf{z}_{ij}\}$ += $\text{DropoutRowwise}_{0.25}(\text{TriangleMultiplicationOutgoing}(\{\mathbf{z}_{ij}\}))$

$\{\mathbf{z}_{ij}\}$ += $\text{DropoutRowwise}_{0.25}(\text{TriangleMultiplicationIncoming}(\{\mathbf{z}_{ij}\}))$

$\{\mathbf{z}_{ij}\}$ += $\text{DropoutRowwise}_{0.25}(\text{TriangleAttentionStartingNode}(\{\mathbf{z}_{ij}\}))$

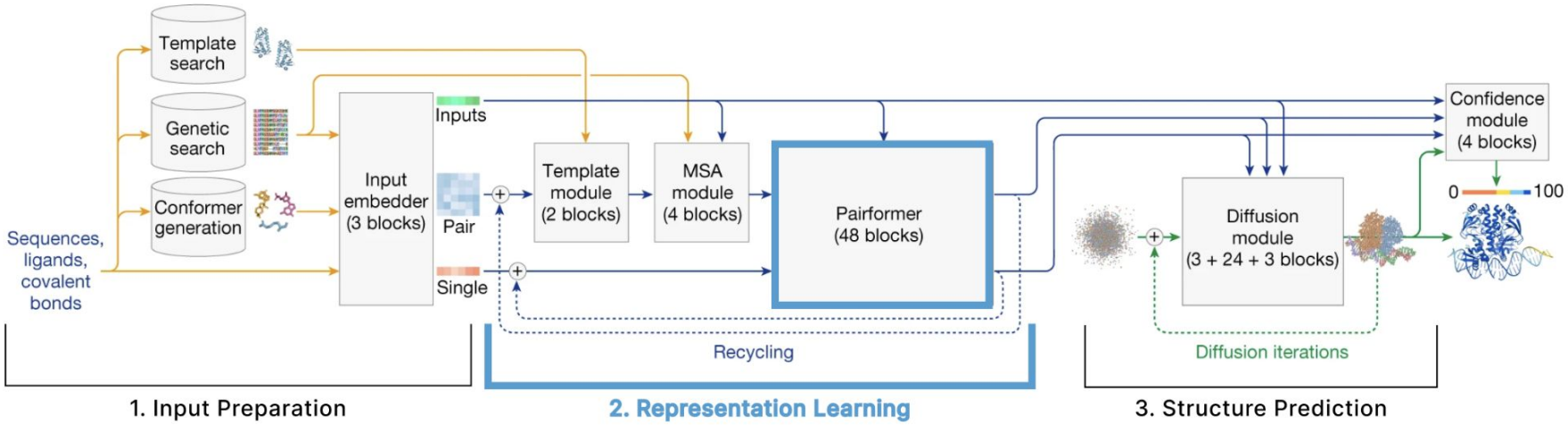
$\{\mathbf{z}_{ij}\}$ += $\text{DropoutColumnwise}_{0.25}(\text{TriangleAttentionEndingNode}(\{\mathbf{z}_{ij}\}))$

$\{\mathbf{z}_{ij}\}$ += $\text{Transition}(\{\mathbf{z}_{ij}\})$

end for

return $\{\mathbf{z}_{ij}\}$

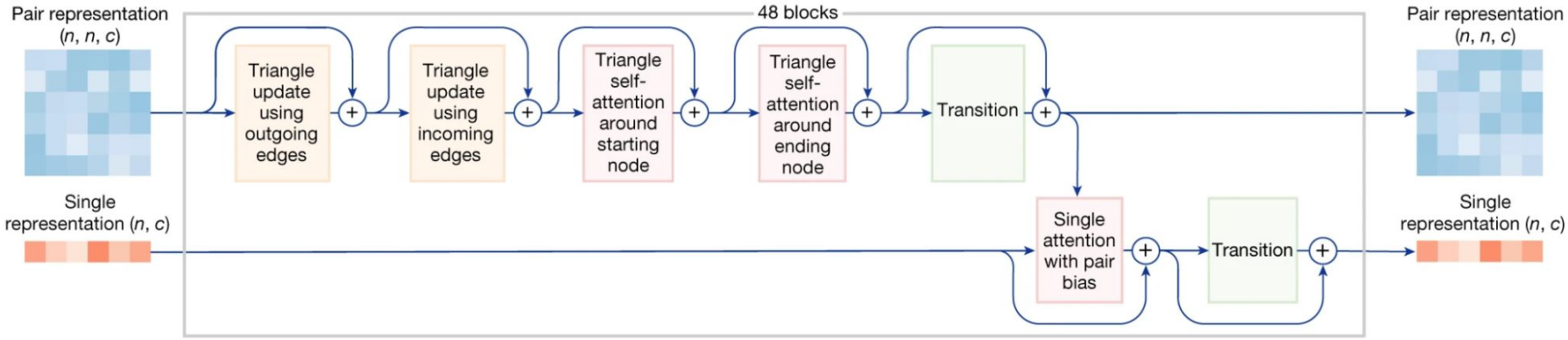
Pairformer Module



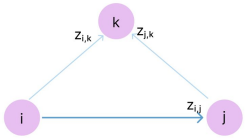
Template Module MSA module **Pairformer**

- **Triangle Updates**
- **Triangle Attention**
- **Attention with Pair Bias (token-level)**

Pairformer



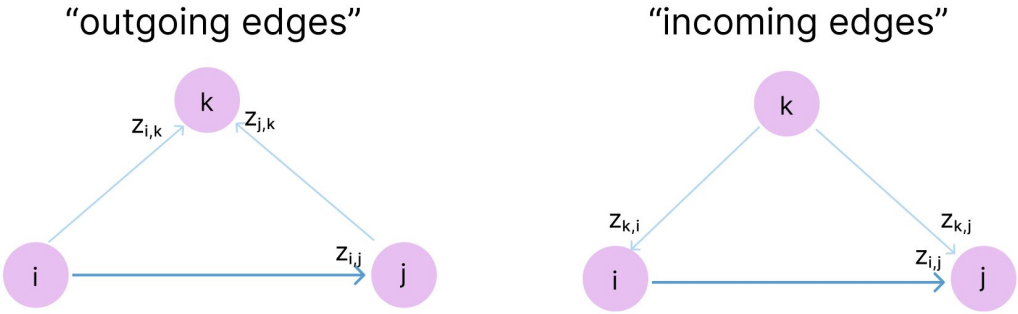
After using templates and MSA to update the pair representation (**z**), AF3 discards them and focuses only on **z** and the single representation **s**. These are now updated via the **Pairformer**, a core architectural block rich with triangle-based reasoning, a key idea from AF2 that remains central in AF3.



Why Triangles?

- **Inspired by the triangle inequality**: If you know the relation (e.g., distance) between i→j and j→k, you can infer constraints on i→k.
- Though **not physically enforced**, triangle consistency is encouraged through updates that consider all triplets (i, j, k).
- This helps encode **geometric constraints** and encourages physical plausibility in structural predictions.

Pairformer



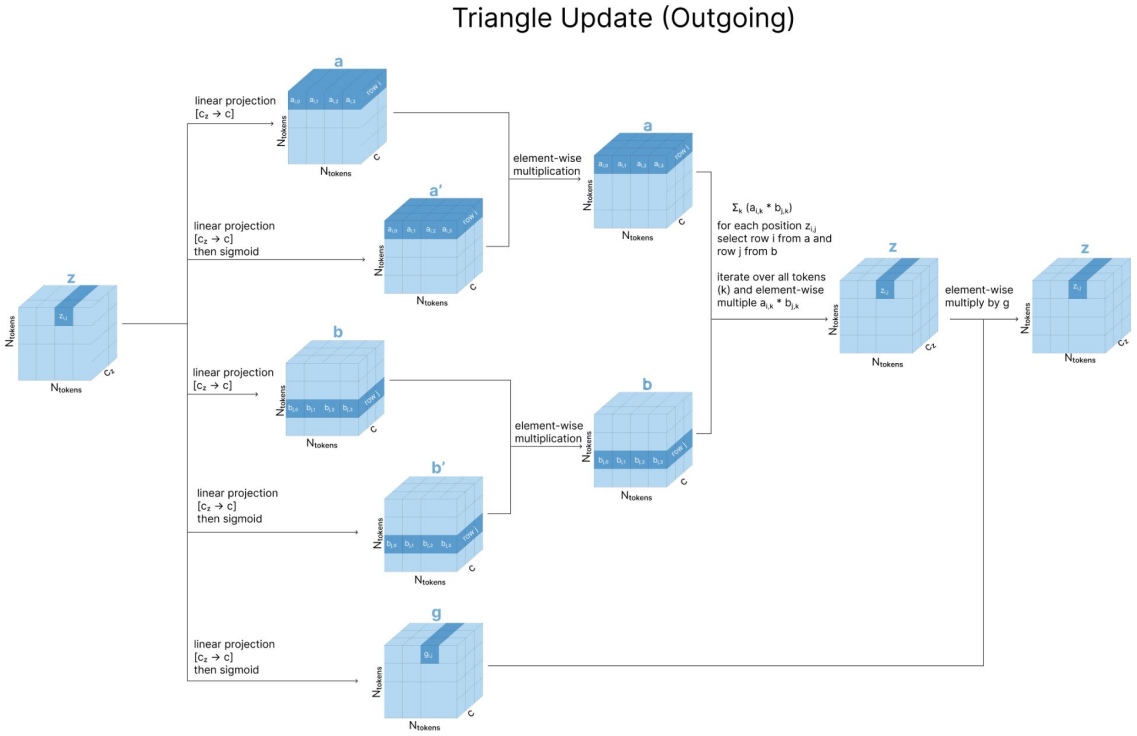
🔁 **Triangle Operations:**

- For each pairwise representation **z**$_{(i,\square)}$, the model considers **intermediate nodes k** to update it:
  - **Outgoing Triangle Attention/Update**: Uses $z_{(i,\square)}$ and $z_{(\square,\square)}$.
  - **Incoming Triangle Attention/Update**: Uses $z_{(\square,i)}$ and $z_{(\square,\square)}$.
- These updates treat **z as a directed graph**, where attention flows along incoming or outgoing edges, encouraging relational consistency across triplets.

🔄 **Single Attention with Pair Bias:**

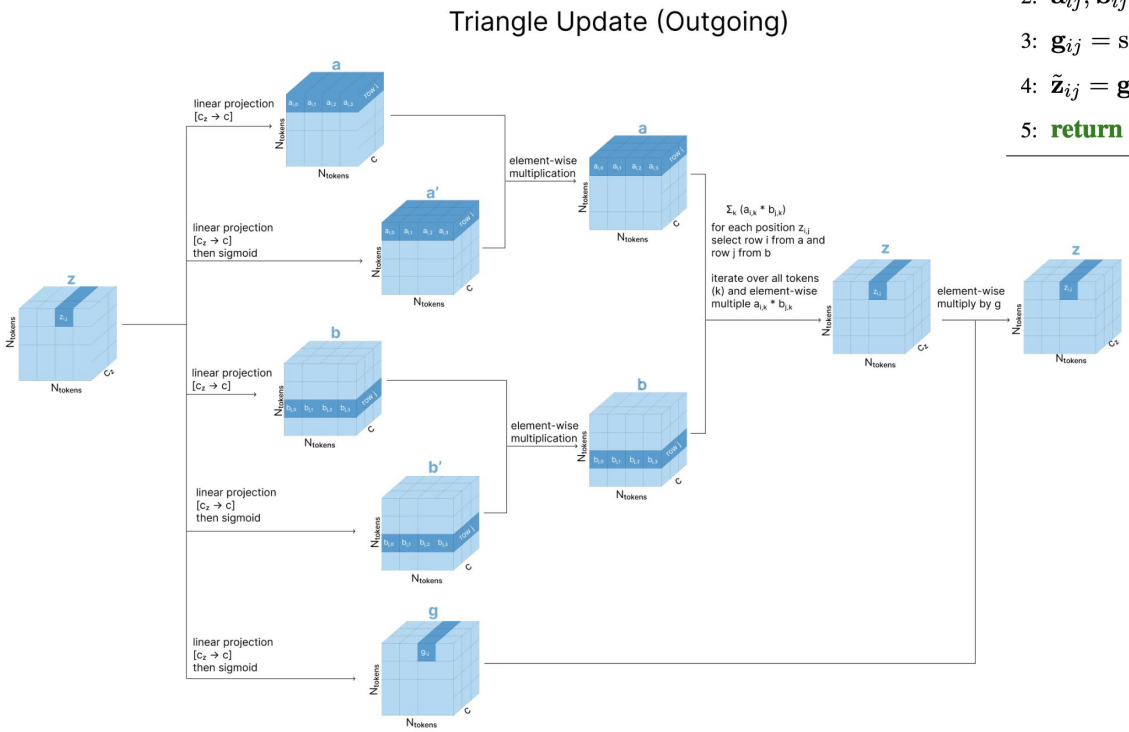
- Similar to earlier attention with pair bias (from Atom Transformer), but now operates at the **token-level**.
- **s** (single) is updated using **z** (pair) as attention bias.
- Helps infuse sequence-level features with geometric and relational cues.

Pairformer - Triangle Updates



Practically, we take three linear projections of **z** (called a, b, and g). To update **zi,j**, we take an element-wise multiplication of **row i** from **a** and **row j** from **b**. We then sum over all these rows (different values of **k**), and **gate** with our **g** projection.

Pairformer - Triangle Updates



Algorithm 12 Triangular multiplicative update using "outgoing" edges

def $\text{TriangleMultiplicationOutgoing}(\{\mathbf{z}_{ij}\}, c = 128)$:

1: $\mathbf{z}_{ij} \leftarrow \text{LayerNorm}(\mathbf{z}_{ij})$

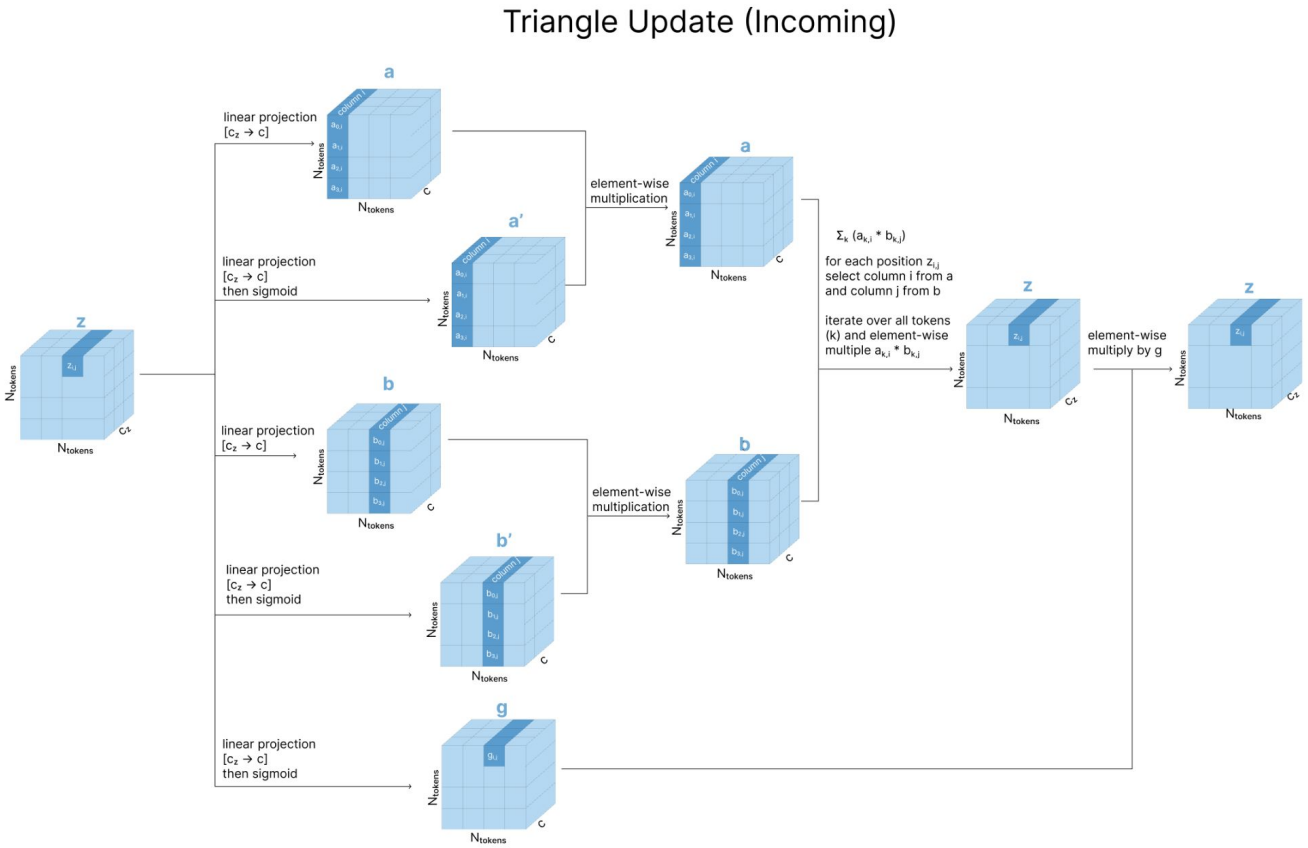
2: $\mathbf{a}_{ij}, \mathbf{b}_{ij} = \text{sigmoid}\left(\text{LinearNoBias}(\mathbf{z}_{ij})\right) \odot \text{LinearNoBias}(\mathbf{z}_{ij})$

3: $\mathbf{g}_{ij} = \text{sigmoid}\left(\text{LinearNoBias}(\mathbf{z}_{ij})\right)$

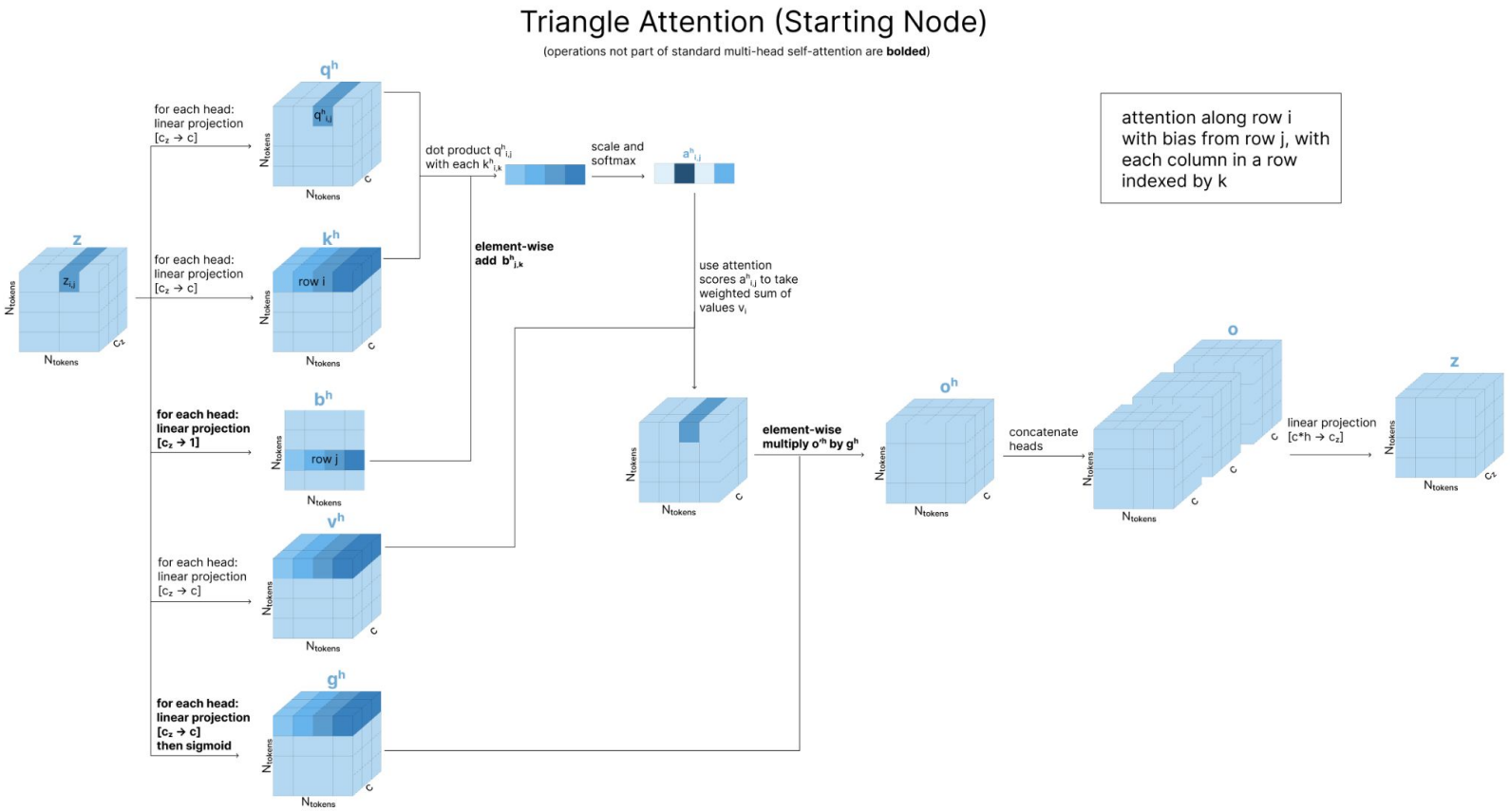
4: $\tilde{\mathbf{z}}_{ij} = \mathbf{g}_{ij} \odot \text{LinearNoBias}(\text{LayerNorm}(\sum_k \mathbf{a}_{ik} \odot \mathbf{b}_{jk}))$

5: **return** $\{\tilde{\mathbf{z}}_{ij}\}$

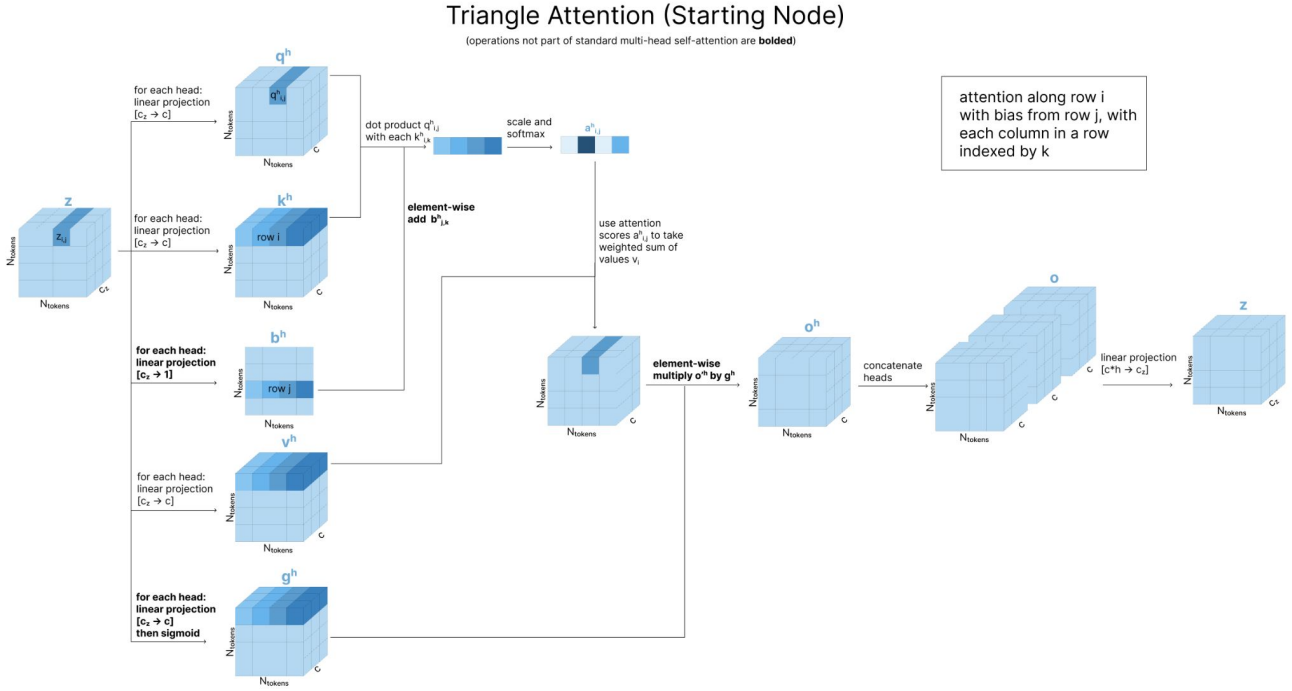
Pairformer - Triangle Updates



Pairformer - Triangle Attention

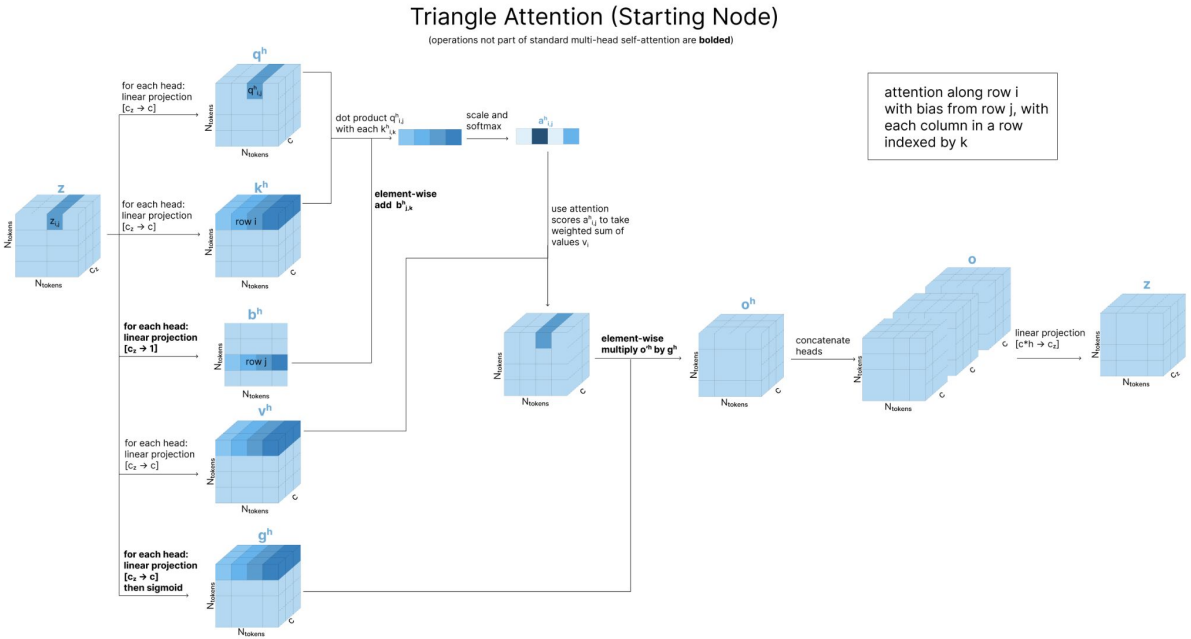


Pairformer - Triangle Attention



- **Built on standard self-attention:** Queries, keys, and values are derived from **z**.
- **Adds triangle bias**: This means the attention score between two edges (e.g., z_{i}□ and z_{i}□) is **biased** based on a third relationship (z□□).
- Think of it as **attention over a 2D matrix (z)** guided by a third point in the triangle.

Pairformer - Triangle Attention



Algorithm 14 Triangular gated self-attention around starting node

def TriangleAttentionStartingNode($\{\mathbf{z}_{ij}\}, c = 32, N_{\text{head}} = 4$) :

 # Input projections

$\mathbf{z}_{ij} \leftarrow \text{LayerNorm}(\mathbf{z}_{ij})$

$\mathbf{q}^h_{ij}, \mathbf{k}^h_{ij}, \mathbf{v}^h_{ij} = \text{LinearNoBias}(\mathbf{z}_{ij})$

$b^h_{ij} = \text{LinearNoBias}(\mathbf{z}_{ij})$

$\mathbf{g}^h_{ij} = \text{sigmoid}\left(\text{LinearNoBias}(\mathbf{z}_{ij})\right)$

 # Attention

$a^h_{ijk} = \text{softmax}_k\left(\frac{1}{\sqrt{c}}\,\mathbf{q}^{h\top}_{ij}\mathbf{k}^h_{ik} + b^h_{jk}\right)$

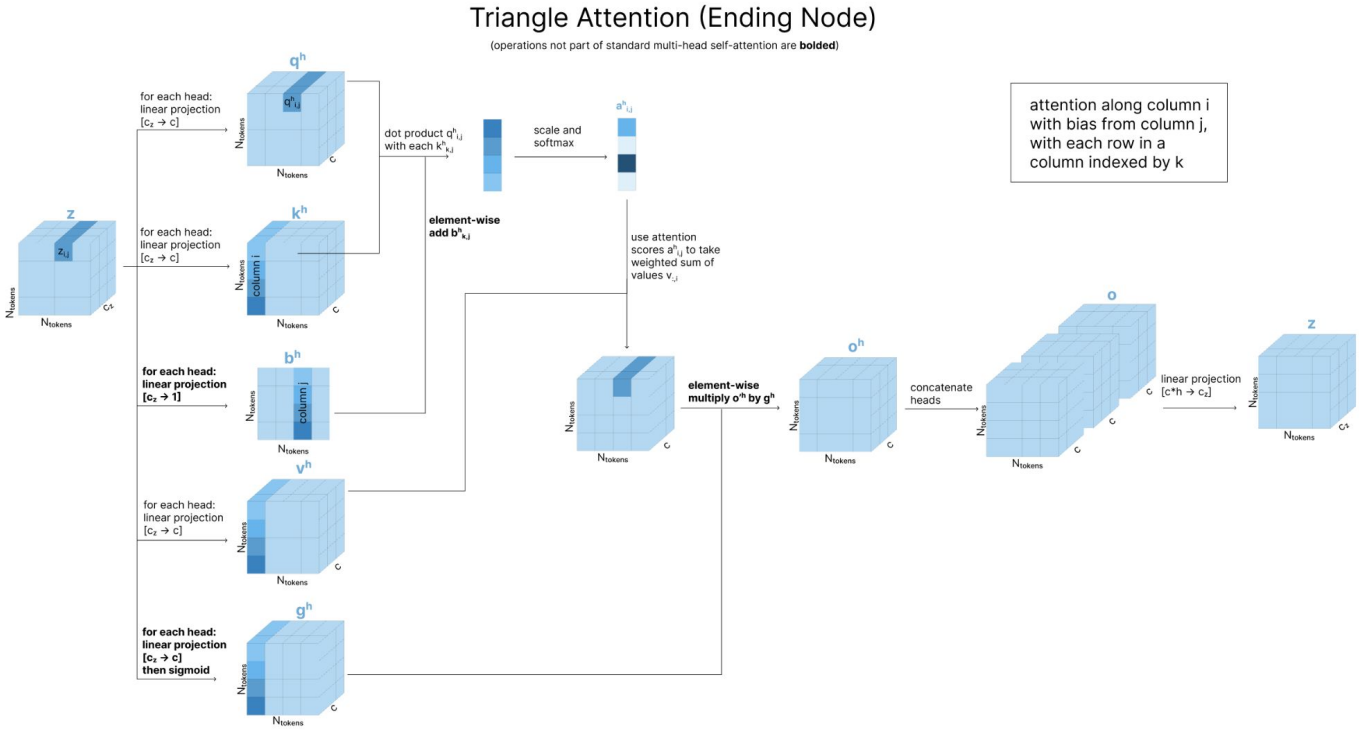
$\mathbf{o}^h_{ij} = \mathbf{g}^h_{ij} \odot \sum_k a^h_{ijk}\mathbf{v}^h_{ik}$

 # Output projection

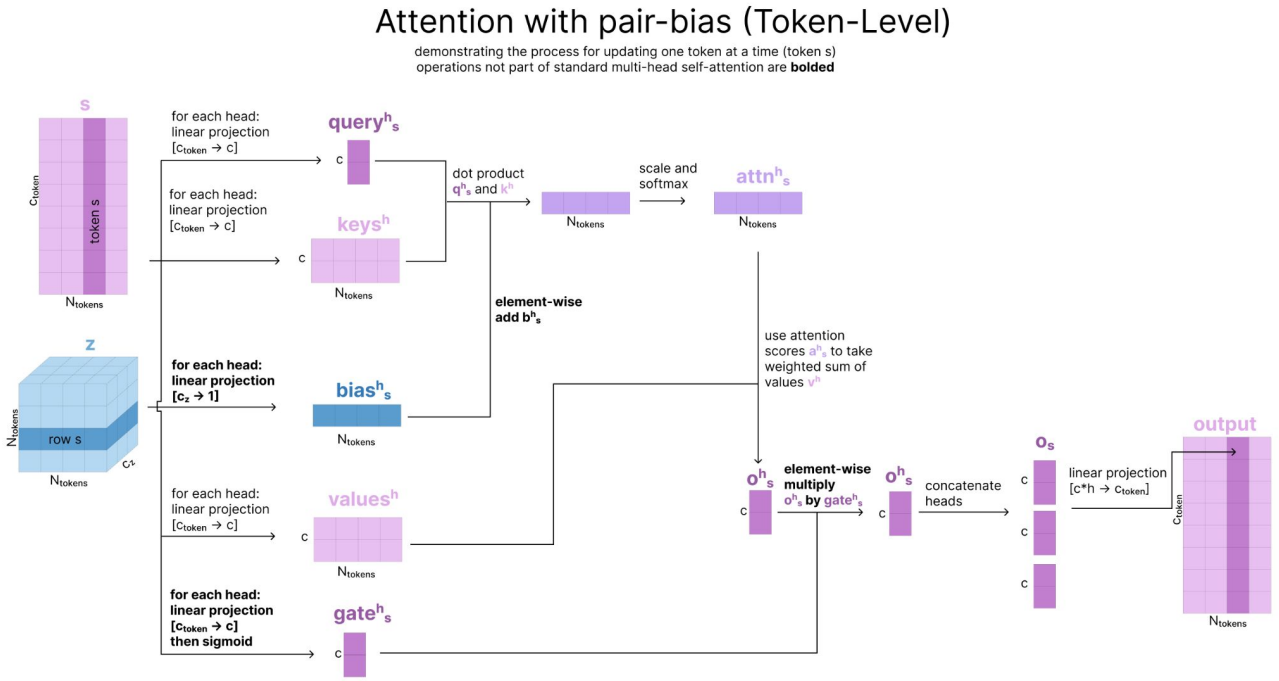
$\tilde{\mathbf{z}}_{ij} = \text{LinearNoBias}\left(\text{concat}_h(\mathbf{o}^h_{ij})\right)$

return $\{\tilde{\mathbf{z}}_{ij}\}$

Pairformer - Triangle Attention



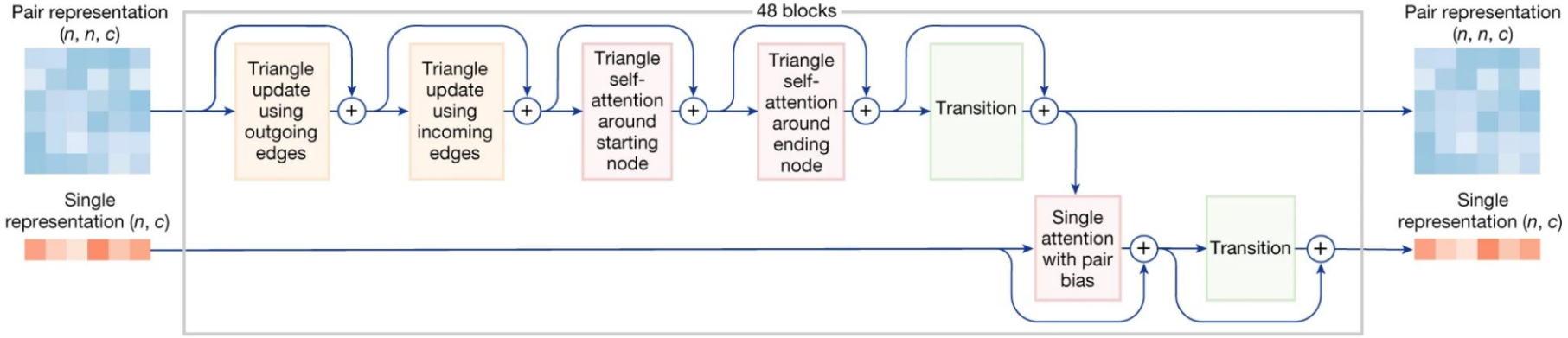
Pairformer - Single Attention with Pair Bias



After the triangle updates and attentions, the pair representation **z** goes through a Transition block. Then, the single representation **s** is updated using **Single Attention with Pair Bias** at the token-level with full attention (unlike sparse atom-level attention).

This Pairformer block is repeated **48 times**, producing the final refined representations **s_trunk** and **z_trunk**.

Pairformer Module



Algorithm 17 Pairformer stack

def PairformerStack($\{\mathbf{s}_i\}, \{\mathbf{z}_{ij}\}, N_{\text{block}} = 48$) :

for all $l \in [1, \ldots, N_{\text{block}}]$ **do**

 # Pair stack

$\{\mathbf{z}_{ij}\}$ += DropoutRowwise$_{0.25}$(TriangleMultiplicationOutgoing($\{\mathbf{z}_{ij}\}$))

$\{\mathbf{z}_{ij}\}$ += DropoutRowwise$_{0.25}$(TriangleMultiplicationIncoming($\{\mathbf{z}_{ij}\}$))

$\{\mathbf{z}_{ij}\}$ += DropoutRowwise$_{0.25}$(TriangleAttentionStartingNode($\{\mathbf{z}_{ij}\}$))

$\{\mathbf{z}_{ij}\}$ += DropoutColumnwise$_{0.25}$(TriangleAttentionEndingNode($\{\mathbf{z}_{ij}\}$))

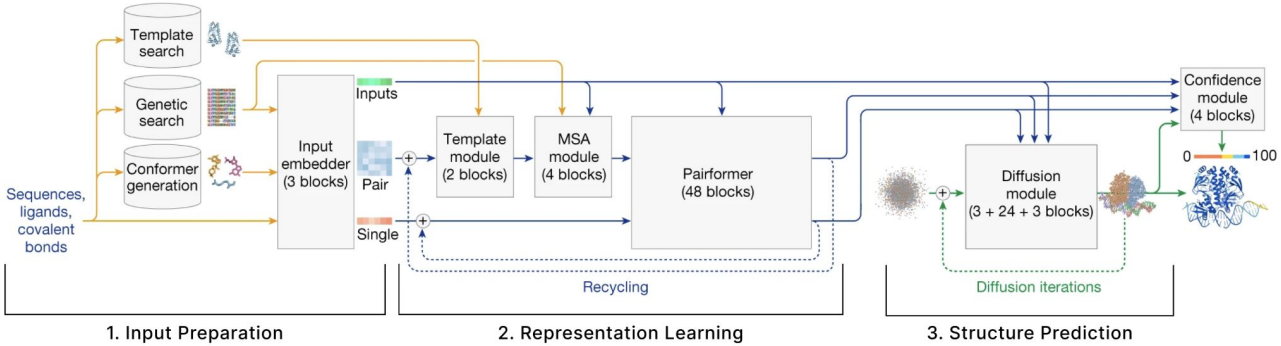
$\{\mathbf{z}_{ij}\}$ += Transition($\{\mathbf{z}_{ij}\}$)

$\{\mathbf{s}_i\}$ += AttentionPairBias($\{\mathbf{s}_i\}, \emptyset, \{\mathbf{z}_{ij}\}, \beta_{ij} = 0, N_{\text{head}} = 16$)

$\{\mathbf{s}_i\}$ += Transition($\{\mathbf{s}_i\}$)

end for

return $\{\mathbf{s}_i\}, \{\mathbf{z}_{ij}\}$



Algorithm 1 Main Inference Loop

def MainInferenceLoop($\{\mathbf{f}^*\}, N_{\text{cycle}} = 4, c_s = 384, c_z = 128$) :

$\{\mathbf{s}_i^{\text{inputs}}\} = \text{InputFeatureEmbedder}(\{\mathbf{f}^*\})$

$\mathbf{s}_i^{\text{init}} = \text{LinearNoBias}(\mathbf{s}_i^{\text{inputs}})$ $\quad \mathbf{s}_i^{\text{init}} \in \mathbb{R}^{c_s}$

$\mathbf{z}_{ij}^{\text{init}} = \text{LinearNoBias}(\mathbf{s}_i^{\text{inputs}}) + \text{LinearNoBias}(\mathbf{s}_j^{\text{inputs}})$ $\quad \mathbf{z}_{ij}^{\text{init}} \in \mathbb{R}^{c_z}$

$\mathbf{z}_{ij}^{\text{init}} \mathrel{+}= \text{RelativePositionEncoding}(\{\mathbf{f}^*\})$

$\mathbf{z}_{ij}^{\text{init}} \mathrel{+}= \text{LinearNoBias}(\mathbf{f}_{ij}^{\text{token\_bonds}})$

$\{\hat{\mathbf{z}}_{ij}\}, \{\hat{\mathbf{s}}_i\} = \mathbf{0}, \mathbf{0}$

for all $c \in [1, \ldots, N_{\text{cycle}}]$ **do**

  $\mathbf{z}_{ij} = \mathbf{z}_{ij}^{\text{init}} + \text{LinearNoBias}(\text{LayerNorm}(\hat{\mathbf{z}}_{ij}))$ $\quad \mathbf{z}_{ij} \in \mathbb{R}^{c_z}$

  $\{\mathbf{z}_{ij}\} \mathrel{+}= \text{TemplateEmbedder}(\{\mathbf{f}^*\}, \{\mathbf{z}_{ij}\})$

  $\{\mathbf{z}_{ij}\} \mathrel{+}= \text{MsaModule}(\{\mathbf{f}_{Si}^{\text{msa}}\}, \{\mathbf{z}_{ij}\}, \{\mathbf{s}_i^{\text{inputs}}\})$

  $\mathbf{s}_i = \mathbf{s}_i^{\text{init}} + \text{LinearNoBias}(\text{LayerNorm}(\hat{\mathbf{s}}_i))$ $\quad \mathbf{s}_i \in \mathbb{R}^{c_s}$

  $\{\mathbf{s}_i\}, \{\mathbf{z}_{ij}\} = \text{PairformerStack}(\{\mathbf{s}_i\}, \{\mathbf{z}_{ij}\})$

  $\{\hat{\mathbf{s}}_i\}, \{\hat{\mathbf{z}}_{ij}\} \leftarrow \{\mathbf{s}_i\}, \{\mathbf{z}_{ij}\}$

end for

$\{\vec{\mathbf{x}}_l^{\text{pred}}\} = \text{SampleDiffusion}(\{\mathbf{f}^*\}, \{\mathbf{s}_i^{\text{inputs}}\}, \{\mathbf{s}_i\}, \{\mathbf{z}_{ij}\})$

$\{\mathbf{p}_l^{\text{plddt}}\}, \{\mathbf{p}_{ij}^{\text{pae}}\}, \{\mathbf{p}_{ij}^{\text{pde}}\}, \{\mathbf{p}_l^{\text{resolved}}\} = \text{ConfidenceHead}(\{\mathbf{s}_i^{\text{inputs}}\}, \{\mathbf{s}_i\}, \{\mathbf{z}_{ij}\}, \{\vec{\mathbf{x}}_l^{\text{pred}}\})$

$\mathbf{p}_{ij}^{\text{distogram}} = \text{DistogramHead}(\mathbf{z}_{ij})$

return $\{\vec{\mathbf{x}}_l^{\text{pred}}\}, \{\mathbf{p}_l^{\text{plddt}}\}, \{\mathbf{p}_{ij}^{\text{pae}}\}, \{\mathbf{p}_{ij}^{\text{pde}}\}, \{\mathbf{p}_l^{\text{resolved}}\}, \{\mathbf{p}_{ij}^{\text{distogram}}\}$

End of Part 2

Thanks for listening!

Questions - Comments?

Algorithm 1 Main Inference Loop

def MainInferenceLoop($\{\mathbf{f}^*\}, N_{\text{cycle}} = 4, c_s = 384, c_z = 128$) :

$\{\mathbf{s}_i^{\text{inputs}}\} = \text{InputFeatureEmbedder}(\{\mathbf{f}^*\})$

$\mathbf{s}_i^{\text{init}} = \text{LinearNoBias}(\mathbf{s}_i^{\text{inputs}})$ $\mathbf{s}_i^{\text{init}} \in \mathbb{R}^{c_s}$

$\mathbf{z}_{ij}^{\text{init}} = \text{LinearNoBias}(\mathbf{s}_i^{\text{inputs}}) + \text{LinearNoBias}(\mathbf{s}_j^{\text{inputs}})$ $\mathbf{z}_{ij}^{\text{init}} \in \mathbb{R}^{c_z}$

$\mathbf{z}_{ij}^{\text{init}} \mathrel{+}= \text{RelativePositionEncoding}(\{\mathbf{f}^*\})$

$\mathbf{z}_{ij}^{\text{init}} \mathrel{+}= \text{LinearNoBias}(\mathbf{f}_{ij}^{\text{token\_bonds}})$

$\{\hat{\mathbf{z}}_{ij}\}, \{\hat{\mathbf{s}}_i\} = \mathbf{0}, \mathbf{0}$

for all $c \in [1, \ldots, N_{\text{cycle}}]$ **do**

  $\mathbf{z}_{ij} = \mathbf{z}_{ij}^{\text{init}} + \text{LinearNoBias}(\text{LayerNorm}(\hat{\mathbf{z}}_{ij}))$ $\mathbf{z}_{ij} \in \mathbb{R}^{c_z}$

  $\{\mathbf{z}_{ij}\} \mathrel{+}= \text{TemplateEmbedder}(\{\mathbf{f}^*\}, \{\mathbf{z}_{ij}\})$

  $\{\mathbf{z}_{ij}\} \mathrel{+}= \text{MsaModule}(\{\mathbf{f}_{Si}^{\text{msa}}\}, \{\mathbf{z}_{ij}\}, \{\mathbf{s}_i^{\text{inputs}}\})$

  $\mathbf{s}_i = \mathbf{s}_i^{\text{init}} + \text{LinearNoBias}(\text{LayerNorm}(\hat{\mathbf{s}}_i))$ $\mathbf{s}_i \in \mathbb{R}^{c_s}$

  $\{\mathbf{s}_i\}, \{\mathbf{z}_{ij}\} = \text{PairformerStack}(\{\mathbf{s}_i\}, \{\mathbf{z}_{ij}\})$

  $\{\hat{\mathbf{s}}_i\}, \{\hat{\mathbf{z}}_{ij}\} \leftarrow \{\mathbf{s}_i\}, \{\mathbf{z}_{ij}\}$

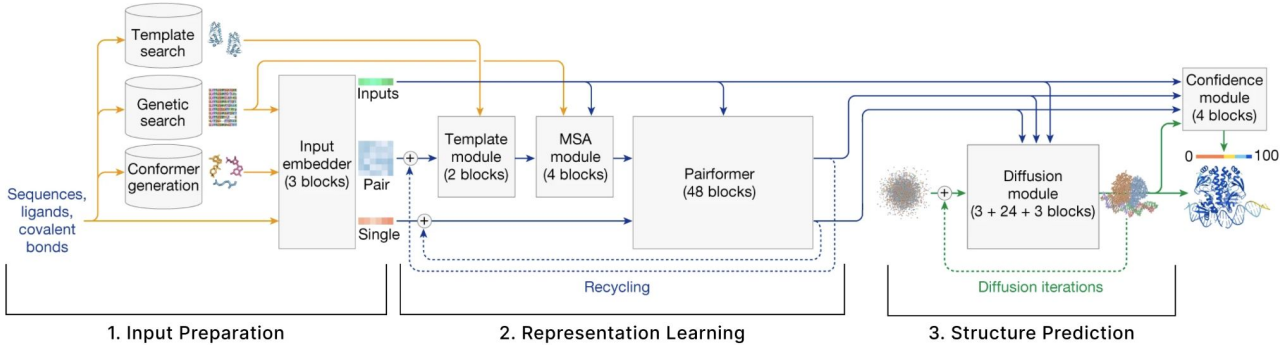
end for

$\{\vec{\mathbf{x}}_l^{\text{pred}}\} = \text{SampleDiffusion}(\{\mathbf{f}^*\}, \{\mathbf{s}_i^{\text{inputs}}\}, \{\mathbf{s}_i\}, \{\mathbf{z}_{ij}\})$

$\{\mathbf{p}_l^{\text{plddt}}\}, \{\mathbf{p}_{ij}^{\text{pae}}\}, \{\mathbf{p}_{ij}^{\text{pde}}\}, \{\mathbf{p}_l^{\text{resolved}}\} = \text{ConfidenceHead}(\{\mathbf{s}_i^{\text{inputs}}\}, \{\mathbf{s}_i\}, \{\mathbf{z}_{ij}\}, \{\vec{\mathbf{x}}_l^{\text{pred}}\})$

$\mathbf{p}_{ij}^{\text{distogram}} = \text{DistogramHead}(\mathbf{z}_{ij})$

return $\{\vec{\mathbf{x}}_l^{\text{pred}}\}, \{\mathbf{p}_l^{\text{plddt}}\}, \{\mathbf{p}_{ij}^{\text{pae}}\}, \{\mathbf{p}_{ij}^{\text{pde}}\}, \{\mathbf{p}_l^{\text{resolved}}\}, \{\mathbf{p}_{ij}^{\text{distogram}}\}$



Accurate structure prediction of biomolecular interactions with AlphaFold 3 (Part 3)

Abramson, J., Adler, J., Dunger, J. et al.

https://www.nature.com/articles/s41586-024-07487-w

Burak Suyunu

LifeLU Reading Group

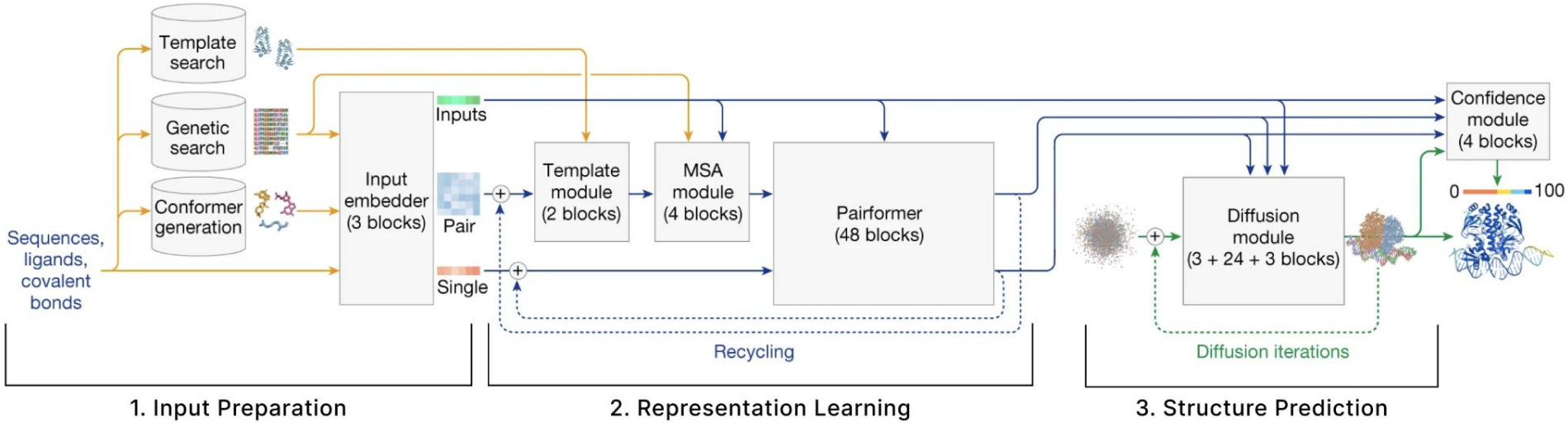
19.06.2025

Previously on Part 2

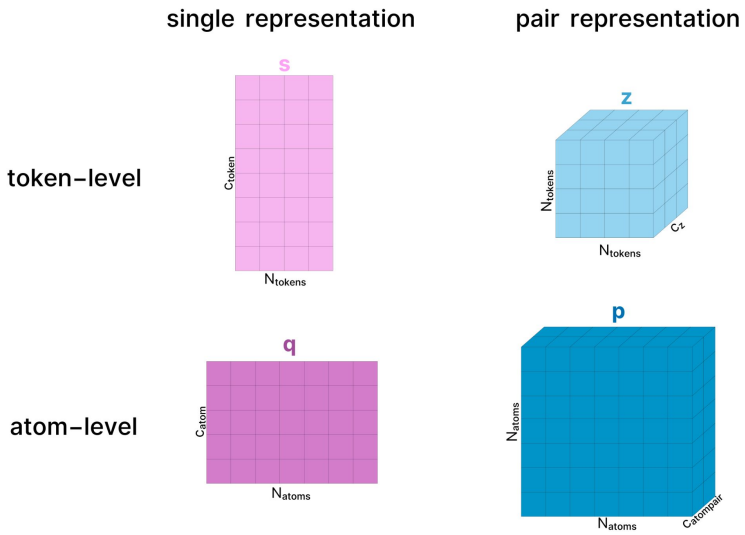
From AlphaFold2 to AlphaFold3 – A Paradigm Shift

Feature	AlphaFold2 (2021)	AlphaFold-Multimer (2021)	AlphaFold3 (2024)
Input types	Protein sequences	Protein sequences	Proteins, RNA, DNA, ligands
Complex prediction	No	Yes (proteins only)	Yes (all molecule types)
Ligand/RNA/DNA modeling	❌	❌	✅
Underlying architecture	Evoformer + structure module	Extended Evoformer	**Pairformer + Diffusion**
Training task	Supervised (PDB structures)	Supervised (PDB structures)	**Conditional diffusion**

Architecture Overview



Notes on the variables and diagrams



AlphaFold 3 represents protein complexes using two key tensors: **Single** (per-token/atom features) and **Pair** (relationships between token/atom pairs). These can be at the token or atom level, shown with consistent names and colors.

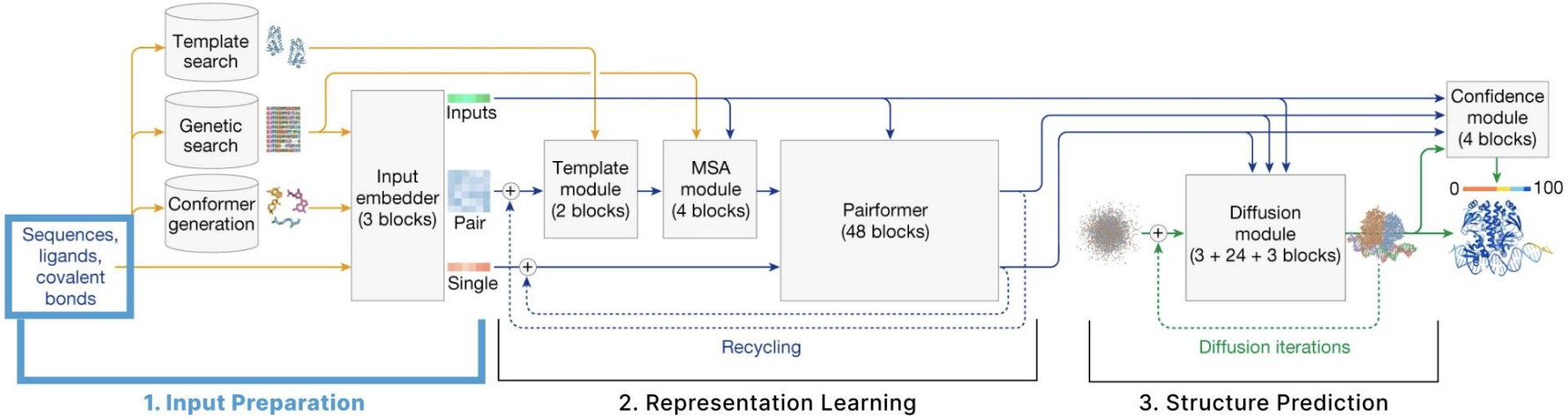
Diagrams show how activation shapes change—not model weights. Tensor labels match those in the AF3 paper. Names are mostly preserved, but some (e.g., c to q in atom-level single) indicate updates through processing. LayerNorms are used throughout but mostly omitted for clarity.

Input Features

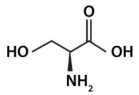
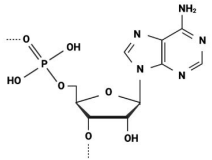
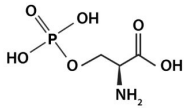
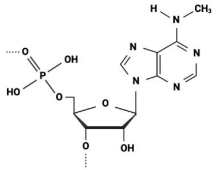
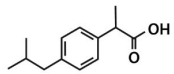
Feature & Shape	Description
residue_index $[N_{\text{token}}]$	Residue number in the token's original input chain.
token_index $[N_{\text{token}}]$	Token number. Increases monotonically; does not restart at 1 for new chains.
asym_id $[N_{\text{token}}]$	Unique integer for each distinct chain.
entity_id $[N_{\text{token}}]$	Unique integer for each distinct sequence.
sym_id $[N_{\text{token}}]$	Unique integer within chains of this sequence. E.g. if chains A, B and C share a sequence but D does not, their sym_ids would be [0, 1, 2, 0].
restype $[N_{\text{token}}, 32]$	One-hot encoding of the sequence. 32 possible values: 20 amino acids + unknown, 4 RNA nucleotides + unknown, 4 DNA nucleotides + unknown, and gap. Ligands represented as "unknown amino acid".
is_protein / rna / dna / ligand $[N_{\text{token}}]$	4 masks indicating the molecule type of a particular token.
ref_pos $[N_{\text{atom}}, 3]$	Atom positions in the reference conformer, with a random rotation and translation applied. Atom positions are given in Å.
ref_mask $[N_{\text{atom}}]$	Mask indicating which atom slots are used in the reference conformer.
ref_element $[N_{\text{atom}}, 128]$	One-hot encoding of the element atomic number for each atom in the reference conformer, up to atomic number 128.
ref_charge $[N_{\text{atom}}]$	Charge for each atom in the reference conformer.

Feature & Shape	Description
ref_atom_name_chars $[N_{\text{atom}}, 4, 64]$	One-hot encoding of the unique atom names in the reference conformer. Each character is encoded as $\text{ord}(c) - 32$, and names are padded to length 4.
ref_space_uid $[N_{\text{atom}}]$	Numerical encoding of the chain id and residue index associated with this reference conformer. Each (chain id, residue index) tuple is assigned an integer on first appearance.
msa $[N_{\text{msa}}, N_{\text{token}}, 32]$	One-hot encoding of the processed MSA, using the same classes as restype.
has_deletion $[N_{\text{msa}}, N_{\text{token}}]$	Binary feature indicating if there is a deletion to the left of each position in the MSA.
deletion_value $[N_{\text{msa}}, N_{\text{token}}]$	Raw deletion counts (the number of deletions to the left of each MSA position) are transformed to $[0, 1]$ using $\frac{2}{\pi} \arctan \frac{d}{3}$.
profile $[N_{\text{token}}, 32]$	Distribution across restypes in the main MSA. Computed before MSA processing (subsection 2.3).
deletion_mean $[N_{\text{token}}]$	Mean number of deletions at each position in the main MSA. Computed before MSA processing (subsection 2.3).
template_restype $[N_{\text{templ}}, N_{\text{token}}]$	One-hot encoding of the template sequence, see restype.
template_pseudo_beta_mask $[N_{\text{templ}}, N_{\text{token}}]$	Mask indicating if the C^{β} (C^{α} for glycine) has coordinates for the template at this residue.
template_backbone_frame_mask $[N_{\text{templ}}, N_{\text{token}}]$	Mask indicating if coordinates exist for all atoms required to compute the backbone frame (used in the template_unit_vector feature).
template_distogram $[N_{\text{templ}}, N_{\text{token}}, N_{\text{token}}, 39]$	A one-hot pairwise feature indicating the distance between C^{β} atoms (C^{α} for glycine). Pairwise distances are discretized into 38 bins of equal width between 3.25 Å and 50.75 Å; one more bin contains any larger distances.
template_unit_vector $[N_{\text{templ}}, N_{\text{token}}, N_{\text{token}}, 3]$	The unit vector of the displacement of the C^{α} atom of all residues within the local frame of each residue. Local frames are computed as in [1].
token_bonds $[N_{\text{token}}, N_{\text{token}}]$	A 2D matrix indicating if there is a bond between any atom in token i and token j, restricted to just polymer-ligand and ligand-ligand bonds and bonds less than 2.4 Å during training.

Tokenization

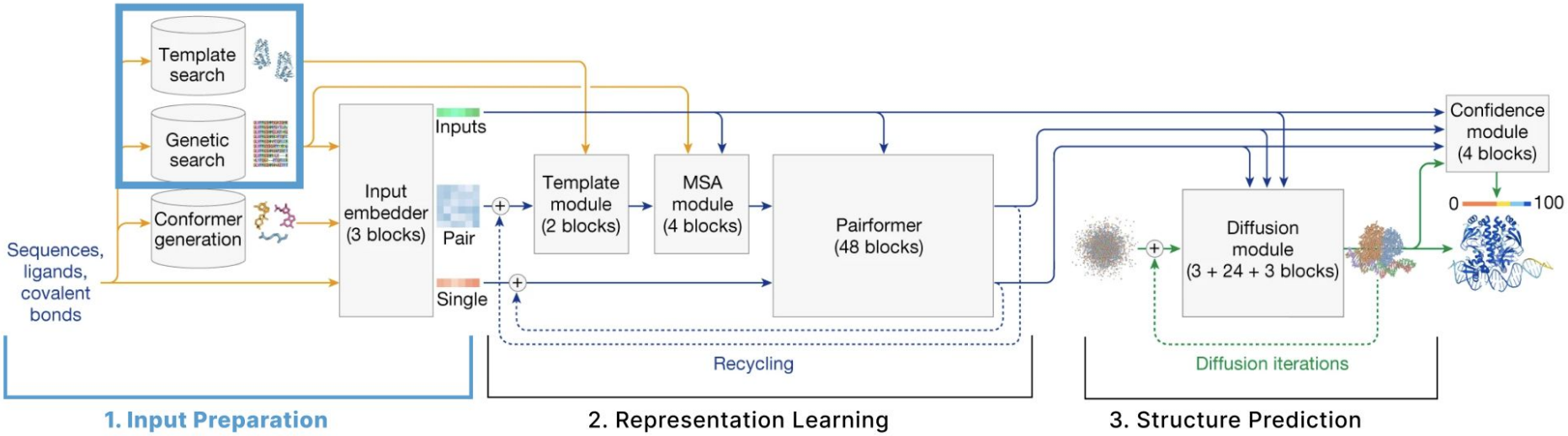


Tokenize input sequences

Standard Amino Acid (serine)	**Standard Nucleotide** (adenosine)	**Modified Amino Acid** (phosphoserine)	**Modified Nucleotide** (methyladenosine)	**Ligand** (ibuprofen)
				
7 atoms* 1 tokens	**23 atoms* 1 token**	**11 atoms* 11 tokens**	**24 atoms* 24 tokens**	**15 atoms* 15 tokens**

*atoms=heavy atoms

Retrieval (Create MSA and Templates)



Tokenize input sequences

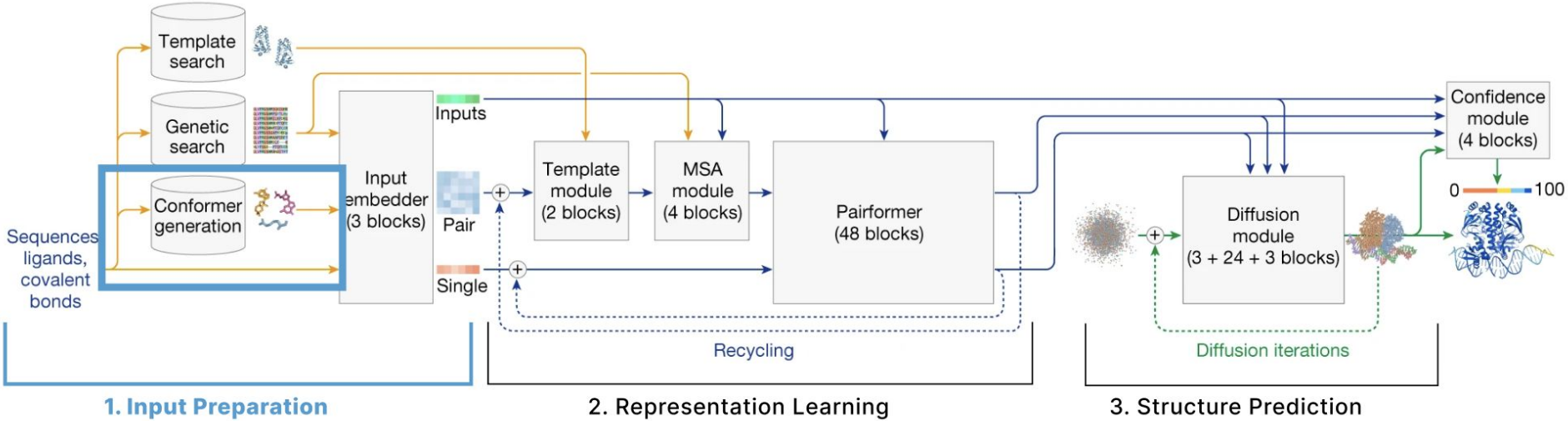
Retrieve similar sequences and structures to create MSA and templates

Create atom-level representation of sequences

Update atom-level representation (Atom Transformer)

Aggregate atom-level representation to token-level

Create Atom-Level Representations



Tokenize input sequences

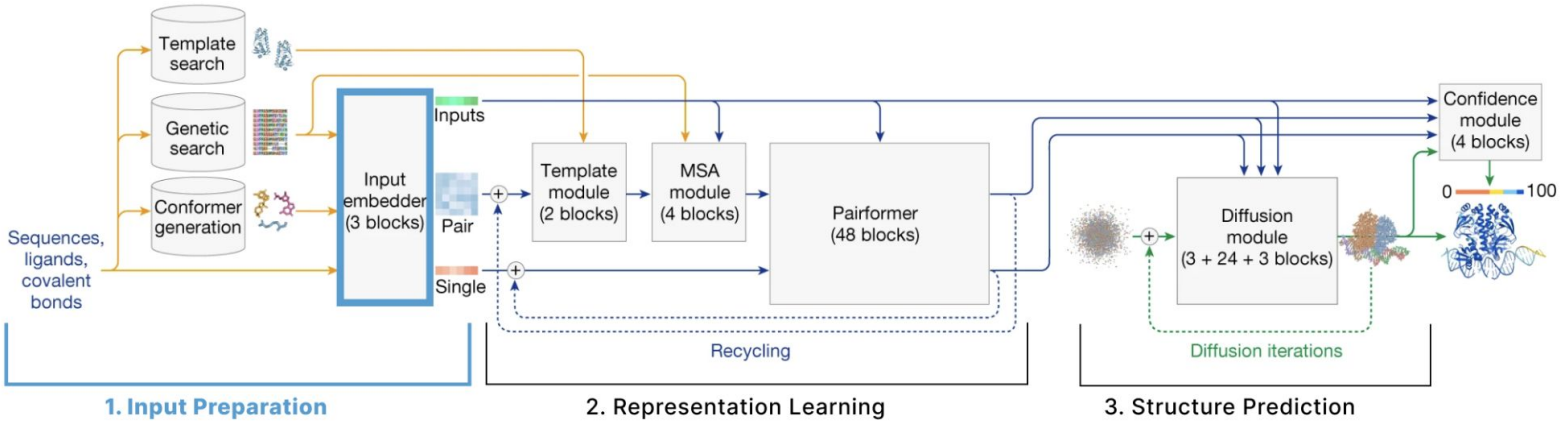
Retrieve similar sequences and structures to create MSA and templates

Create atom-level representation of sequences

Update atom-level representation (Atom Transformer)

Aggregate atom-level representation to token-level

Update Atom-Level Representations (Atom Transformer)



Tokenize input sequences

Retrieve similar sequences and structures to create MSA and templates

Create atom-level representation of sequences

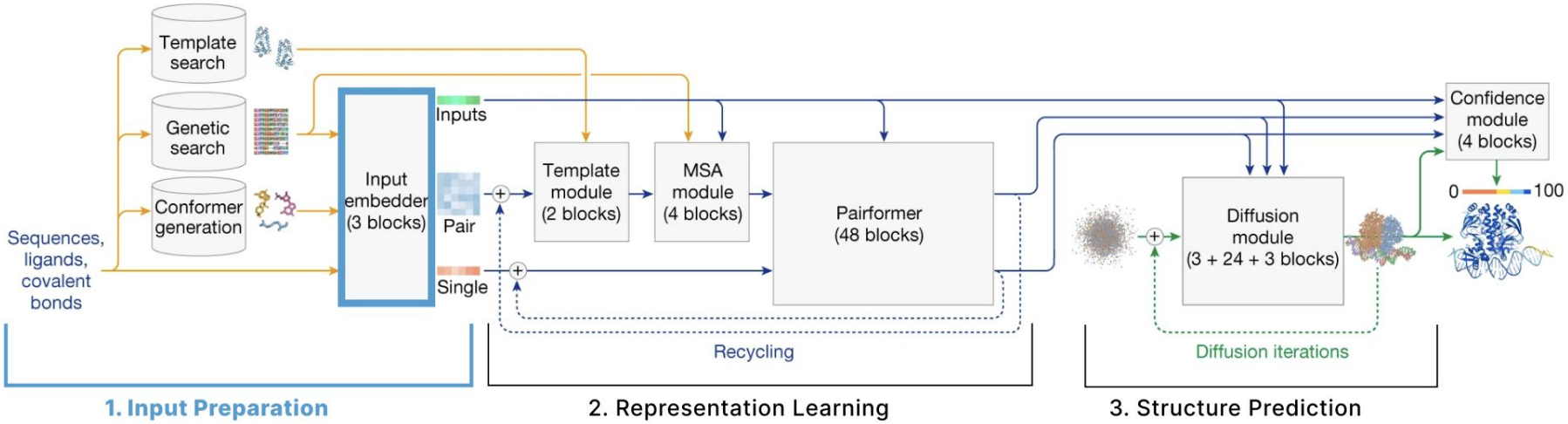
Update atom-level representation (Atom Transformer)

Aggregate atom-level representation to token-level

AtomTransformer introduces building blocks used elsewhere

* **Adaptive LayerNorm**
* **Attention with Pair Bias**
* **Sequence-local atom attention**
* **Conditioned Gating**
* **Conditioned Transition blocks**

Aggregate Atom-Level → Token-Level



Tokenize input sequences

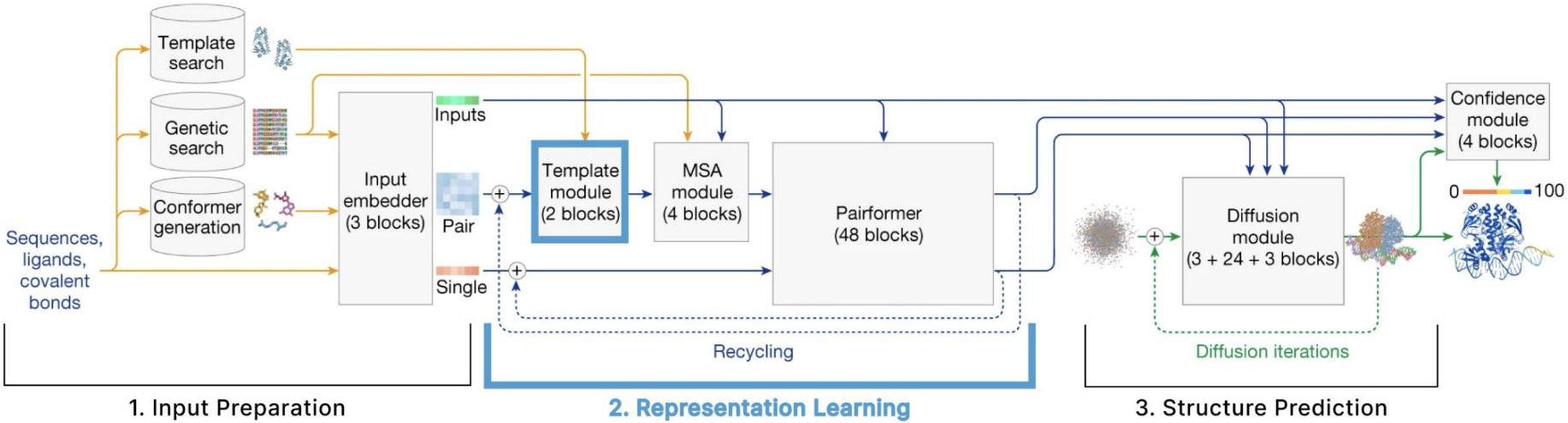
Retrieve similar sequences and structures to create MSA and templates

Create atom-level representation of sequences

Update atom-level representation (Atom Transformer)

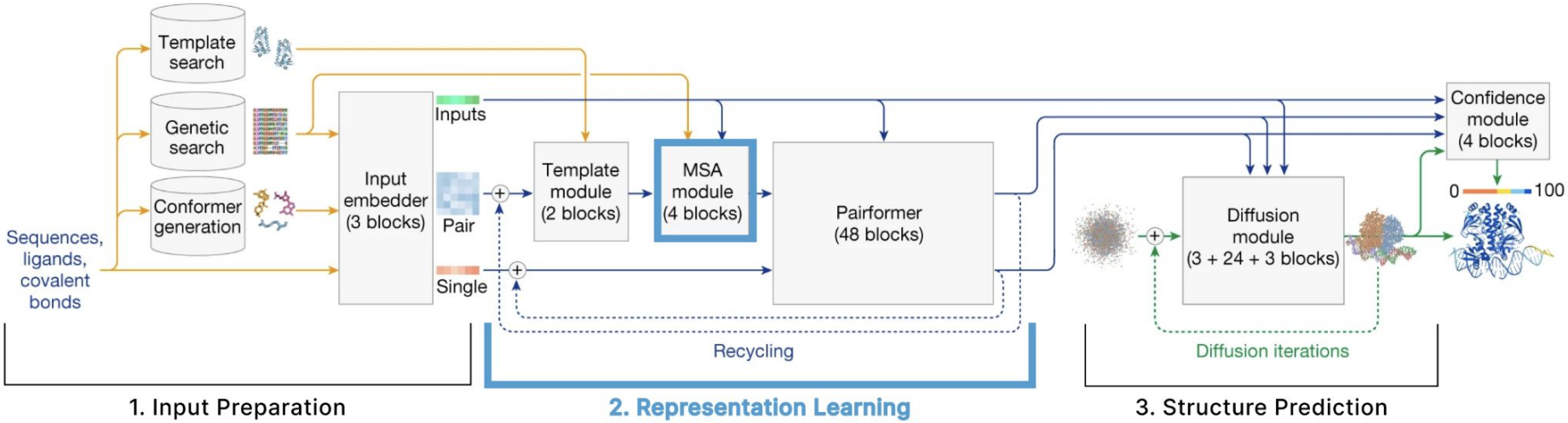
Aggregate atom-level representation to token-level

Template Module



Template Module MSA module Pairformer

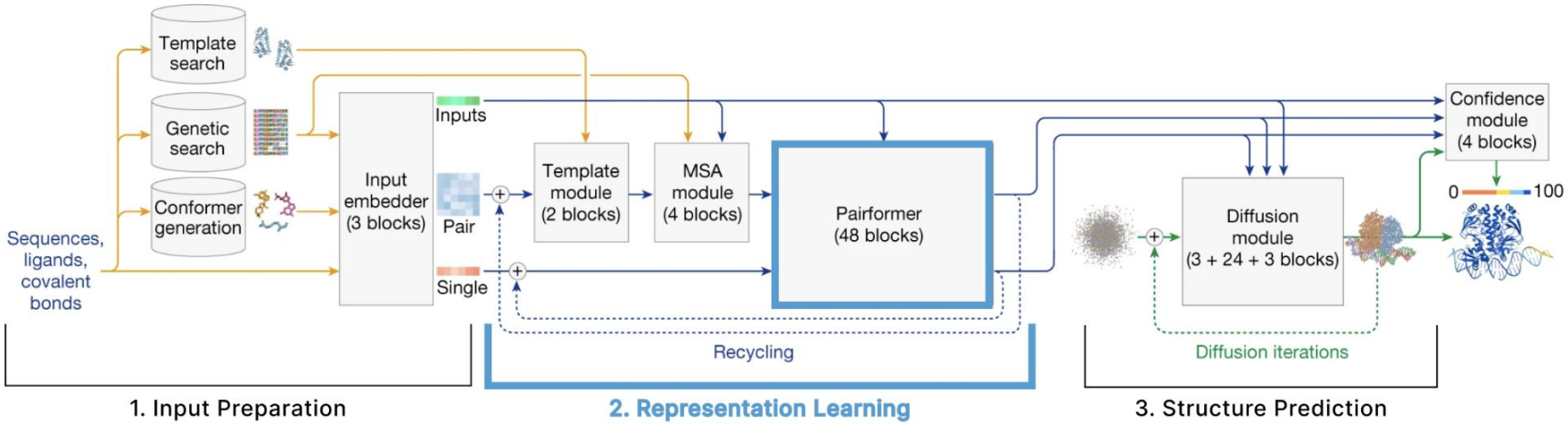
MSA Module



Template Module **MSA module** Pairformer

- **Outer product mean**
- **Row-wise gated attention using only pair bias**

Pairformer Module



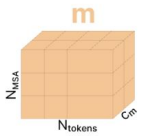
Template Module MSA module **Pairformer**

- **Triangle Updates**
- **Triangle Attention**
- **Attention with Pair Bias (token-level)**

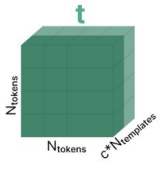
Aggregate Atom-Level → Token-LeveL

Information about related sequences and their structures

Multiple Sequence Alignment

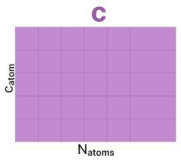


Structure Templates

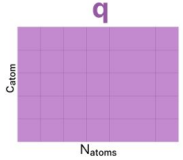


Information about all the atoms (“single”)

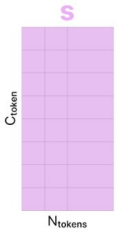
Original Atom-Level Single Representation



Updated Atom-Level Single Representation

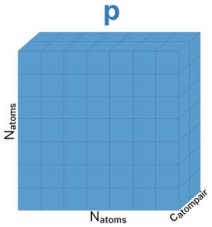


Token-Level Single Representation

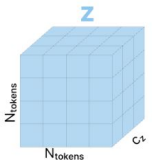


Information about all the pairs of atoms (“pair”)

Atom-Level Pair Representation



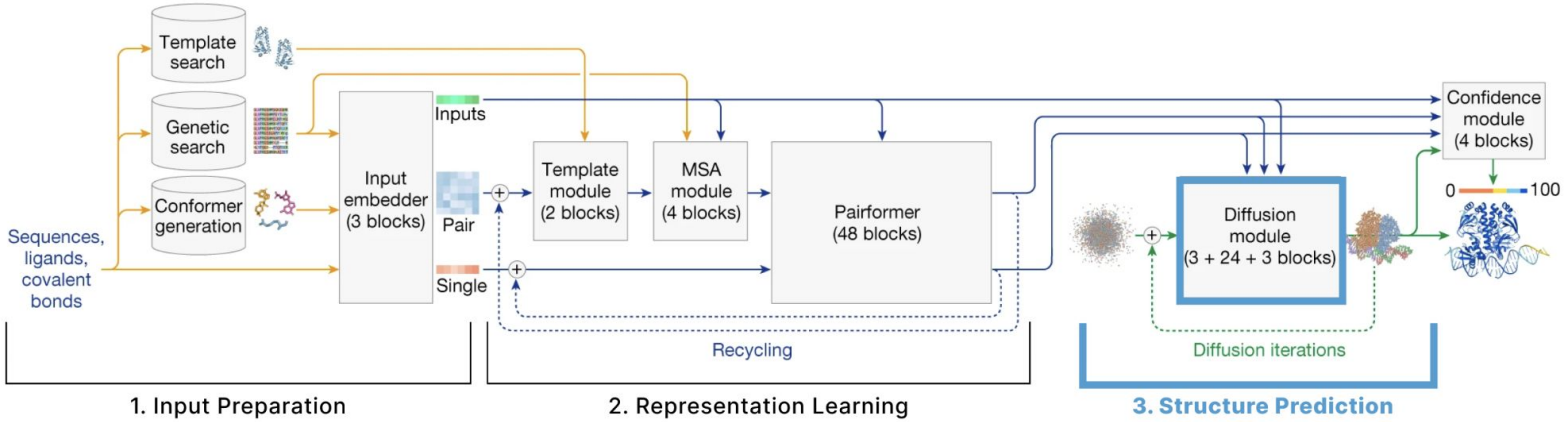
Token-Level Pair Representation



For Step 2, we will 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**).

Let’s Start Part 3

3. Structure Prediction

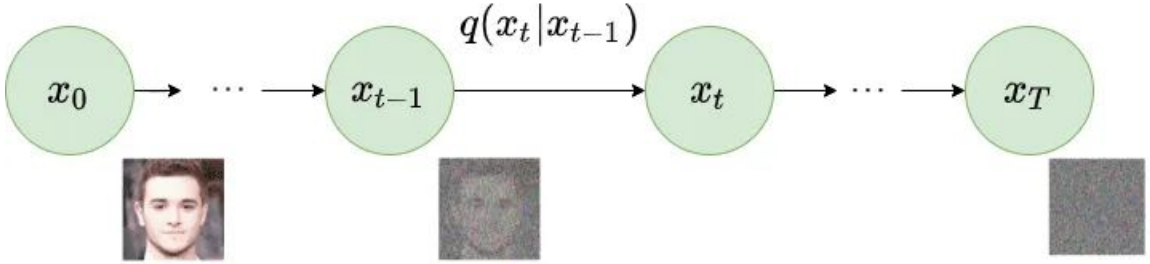


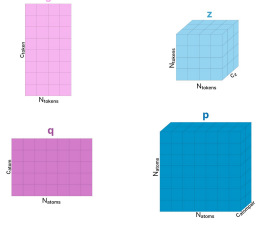
Basics of Diffusion

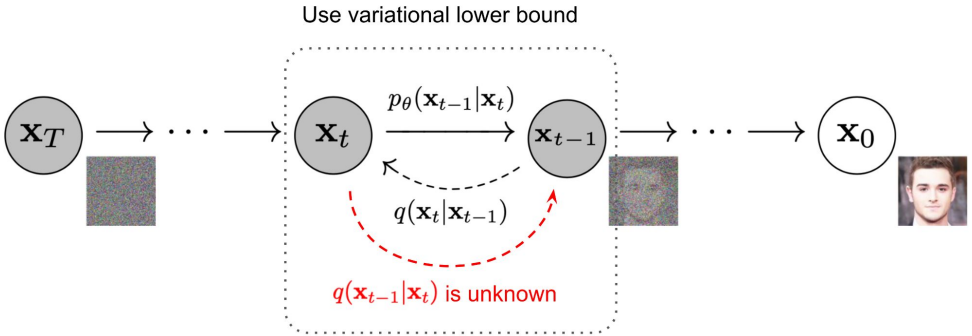
Diffusion Module

1. Prepare token-level conditioning tensors
2. Create atom-level conditioning tensors
3. Token-level attention biased by conditioning tensors
4. Apply attention at atom-level to predict atom-level noise updates

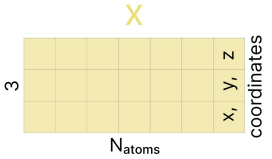
Basics of Diffusion







- **Diffusion model**: Adds noise to real data, then trains a model to reverse it step-by-step.
- **Training**: At each time step, the model predicts the noise added to atom coordinates.
- **Inference**: Starts from pure noise and iteratively removes predicted noise to recover structure.
- **Conditional diffusion**: Model conditions predictions on input info (e.g., protein properties).
- **AF3 input**: x,y,z coordinates of all atoms, with Gaussian noise added during training.
- **Data augmentation**: Random rotations/translations teach model rotational/positional invariance—replacing AF2's complex Invariant Point Attention.



Basics of Diffusion

1️⃣ Forward process (noising process):

- This process is **fixed** and **pre-defined**.
- We take clean data x_0 and gradually add noise to it across T steps according to a schedule.
- At each timestep t, a small amount of Gaussian noise is added:

$$x_t = \sqrt{\bar{\alpha}_t} \cdot x_0 + \sqrt{1 - \bar{\alpha}_t} \cdot \epsilon$$

where:

- $\epsilon \sim \mathcal{N}(0, I)$ is standard Gaussian noise.
- β_t is the noise variance schedule at timestep t.
- $\alpha_t = 1 - \beta_t$, and $\bar{\alpha}_t = \prod_{s=1}^{t} \alpha_s$.
- **This forward process is not learned.** It is just a way of generating noisy training data.

2️⃣ Reverse process (denoising process):

- This is the part that we actually **train the model to learn**.
- The goal is to start from pure noise (x_T) and iteratively remove noise, reconstructing a clean sample x_0.
- The model learns to predict the noise ϵ that was added at each step.

📊 Summary Table

Process	What happens?	Is it learned?
Forward	$x_0 \to x_1 \to \cdots \to x_T$ (add noise)	❌ No
Reverse	$x_T \to x_{T-1} \to \cdots \to x_0$ (remove noise)	✅ Yes

🔧 What does the model learn?

At training time, the model receives a noisy sample x_t and the timestep t, and it tries to predict the noise that was originally added:

$$\hat{\epsilon}_\theta(x_t, t) \approx \epsilon$$

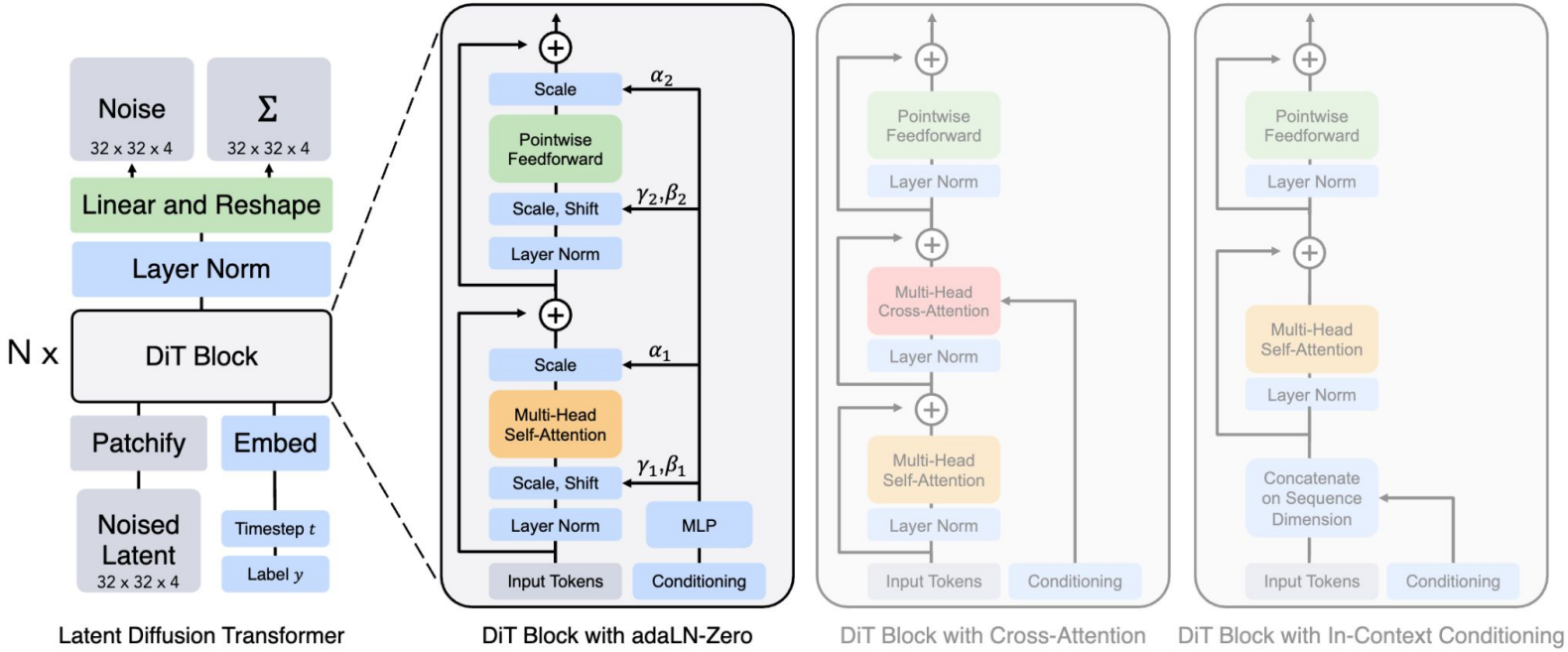
The loss function is usually a simple mean squared error (MSE):

$$\mathcal{L}_{\text{simple}} = \mathbb{E}_{x_0, \epsilon, t} \left[\|\epsilon - \hat{\epsilon}_\theta(x_t, t)\|^2 \right]$$

So, in short:

- The forward process generates training data.
- The reverse process is what we actually train.

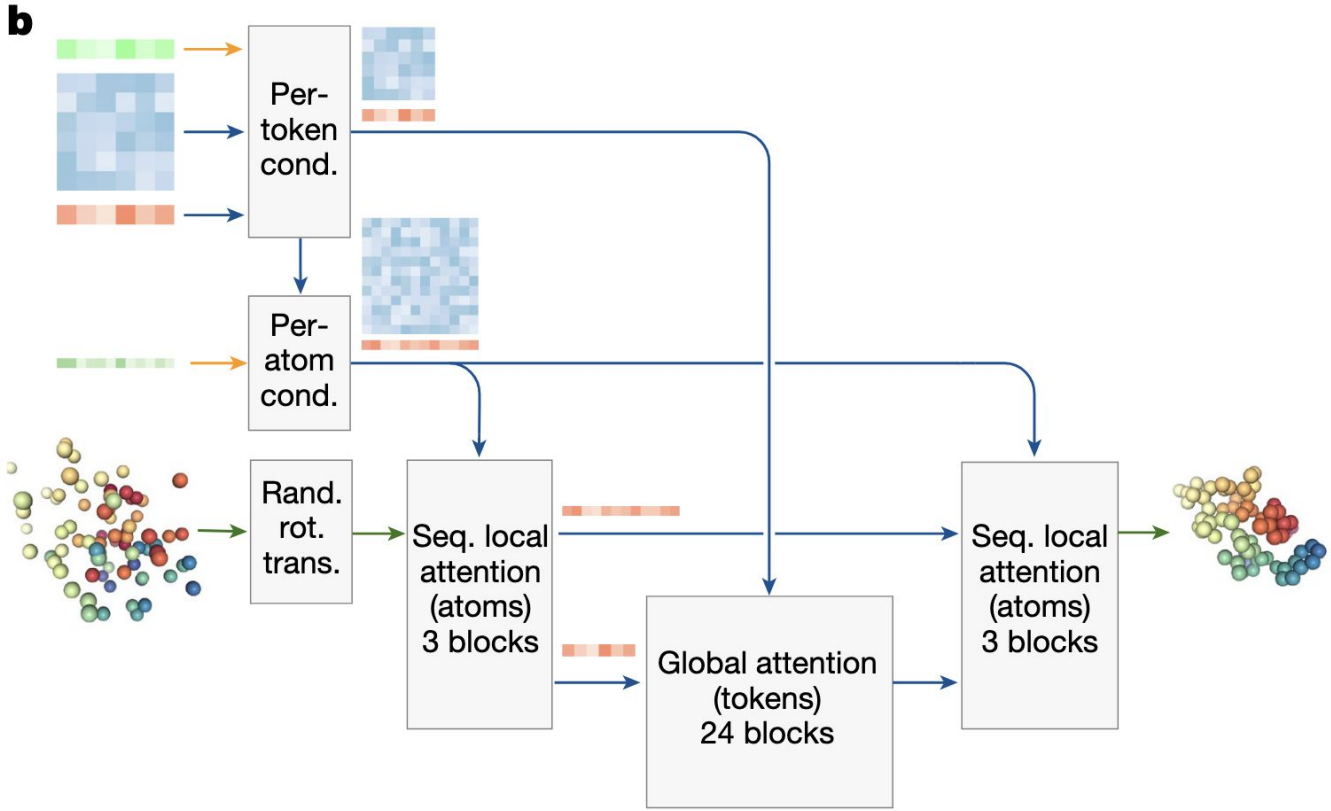
Diffusion Transformer



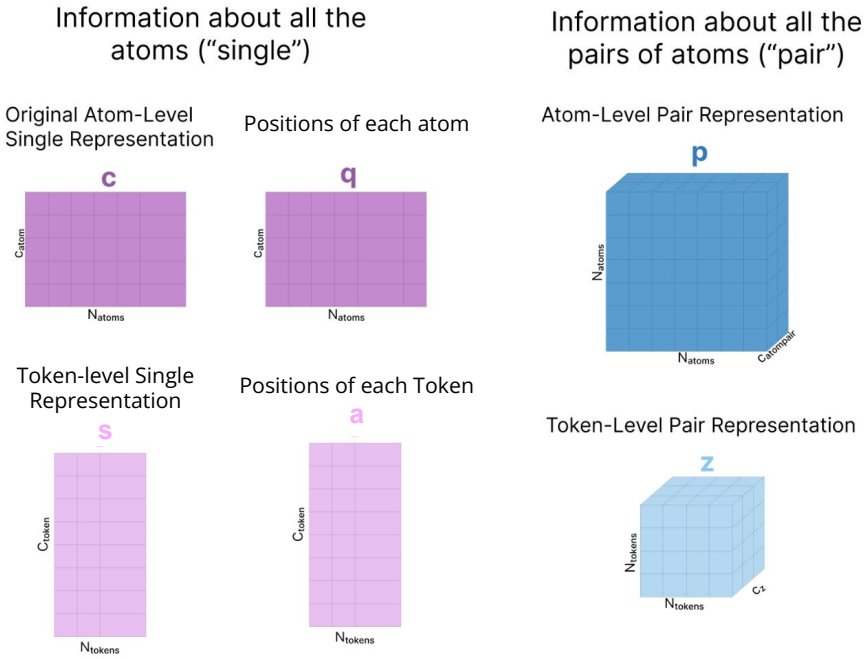
Latent Diffusion Transformer | DiT Block with adaLN-Zero | DiT Block with Cross-Attention | DiT Block with In-Context Conditioning

https://arxiv.org/pdf/2212.09748

Diffusion Module



Atom-Level ⟷ Token-LeveL



atom-level representations (**c**, **q**, **p**)

token-level representations (**s**, **a**, **z**)

Diffusion Module

Algorithm 18 Sample Diffusion

def SampleDiffusion($\{\mathbf{f}^*\}, \{\mathbf{s}_i^{\text{inputs}}\}, \{\mathbf{s}_i^{\text{trunk}}\}, \{\mathbf{z}_{ij}^{\text{trunk}}\}$, Noise Schedule $[c_0, c_1, \ldots, c_T]$, $\gamma_0 = 0.8,\ \gamma_{\min} = 1.0$, noise scale $\lambda = 1.003$, step scale $\eta = 1.5$):

$\vec{\mathbf{x}}_l \sim c_0 \cdot \mathcal{N}(\vec{\mathbf{0}}, \mathbf{I}_3)$ $\quad \vec{\mathbf{x}}_l \in \mathbb{R}^3$

for all $c_\tau \in [c_1, \ldots, c_T]$ **do**

$\{\vec{\mathbf{x}}_l\} \leftarrow \text{CentreRandomAugmentation}(\{\vec{\mathbf{x}}_l\})$

$\gamma = \gamma_0$ if $c_\tau > \gamma_{\min}$ else 0

$\hat{t} = c_{\tau-1}(\gamma + 1)$

$\vec{\xi}_l = \lambda\sqrt{\hat{t}^2 - c_{\tau-1}^2} \cdot \mathcal{N}(\vec{\mathbf{0}}, \mathbf{I}_3)$ $\quad \vec{\xi}_l \in \mathbb{R}^3$

$\vec{\mathbf{x}}_l^{\text{noisy}} = \vec{\mathbf{x}}_l + \vec{\xi}_l$

$\{\vec{\mathbf{x}}_l^{\text{denoised}}\} = \text{DiffusionModule}(\{\vec{\mathbf{x}}_l^{\text{noisy}}\}, \hat{t}, \{\mathbf{f}^*\}, \{\mathbf{s}_i^{\text{inputs}}\}, \{\mathbf{s}_i^{\text{trunk}}\}, \{\mathbf{z}_{ij}^{\text{trunk}}\})$

$\vec{\delta}_l = (\vec{\mathbf{x}}_l - \vec{\mathbf{x}}_l^{\text{denoised}})/\hat{t}$

$dt = c_\tau - \hat{t}$

$\vec{\mathbf{x}}_l \leftarrow \vec{\mathbf{x}}_l^{\text{noisy}} + \eta \cdot dt \cdot \vec{\delta}_l$

end for

return $\{\vec{\mathbf{x}}_l\}$

Algorithm 20 Diffusion Module

def DiffusionModule($\{\vec{\mathbf{x}}_l^{\text{noisy}}\}, \hat{t}, \{\mathbf{f}^*\}, \{\mathbf{s}_i^{\text{inputs}}\}, \{\mathbf{s}_i^{\text{trunk}}\}, \{\mathbf{z}_{ij}^{\text{trunk}}\}$, $\sigma_{\text{data}} = 16, c_{\text{atom}} = 128, c_{\text{atompair}} = 16, c_{\text{token}} = 768)$:

 # Conditioning

$\{\mathbf{s}_i\}, \{\mathbf{z}_{ij}\} = \text{DiffusionConditioning}(\hat{t}, \{\mathbf{f}^*\}, \{\mathbf{s}_i^{\text{inputs}}\}, \{\mathbf{s}_i^{\text{trunk}}\}, \{\mathbf{z}_{ij}^{\text{trunk}}\}, \sigma_{\text{data}})$

 # Scale positions to dimensionless vectors with approximately unit variance.

$\mathbf{r}_l^{\text{noisy}} = \vec{\mathbf{x}}_l^{\text{noisy}} / \sqrt{\hat{t}^2 + \sigma_{\text{data}}^2}$ $\quad \mathbf{r}_l^{\text{noisy}} \in \mathbb{R}^3$

 # Sequence-local Atom Attention and aggregation to coarse-grained tokens

$\{\mathbf{a}_i\}, \{\mathbf{q}_l^{\text{skip}}\}, \{\mathbf{c}_l^{\text{skip}}\}, \{\mathbf{p}_{lm}^{\text{skip}}\} = \text{AtomAttentionEncoder}(\{\mathbf{f}^*\}, \{\mathbf{r}_l^{\text{noisy}}\}, \{\mathbf{s}_i^{\text{trunk}}\}, \{\mathbf{z}_{ij}\}, c_{\text{atom}}, c_{\text{atompair}}, c_{\text{token}})$ $\quad \mathbf{a}_i \in \mathbb{R}^{c_{\text{token}}}$

 # Full self-attention on token level.

$\mathbf{a}_i \mathrel{+}= \text{LinearNoBias}(\text{LayerNorm}(\mathbf{s}_i))$

$\{\mathbf{a}_i\} \leftarrow \text{DiffusionTransformer}(\{\mathbf{a}_i\}, \{\mathbf{s}_i\}, \{\mathbf{z}_{ij}\}, \beta_{ij} = 0, N_{\text{block}} = 24, N_{\text{head}} = 16)$

$\mathbf{a}_i \leftarrow \text{LayerNorm}(\mathbf{a}_i)$

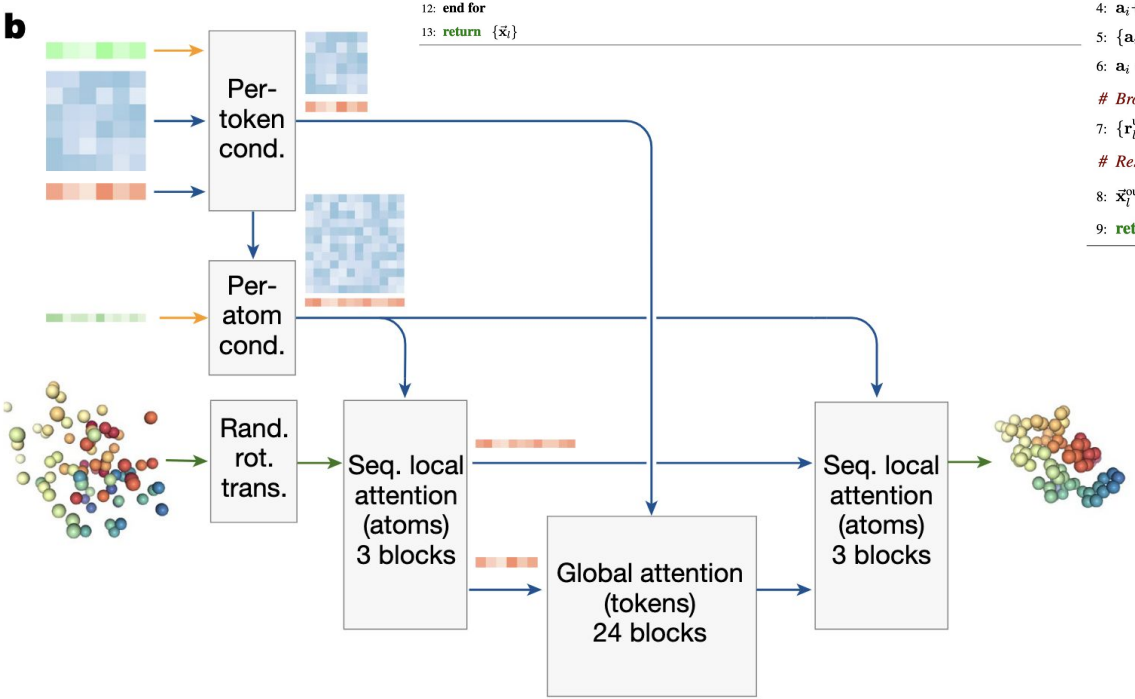
 # Broadcast token activations to atoms and run Sequence-local Atom Attention

$\{\mathbf{r}_l^{\text{update}}\} = \text{AtomAttentionDecoder}(\{\mathbf{a}_i\}, \{\mathbf{q}_l^{\text{skip}}\}, \{\mathbf{c}_l^{\text{skip}}\}, \{\mathbf{p}_{lm}^{\text{skip}}\})$

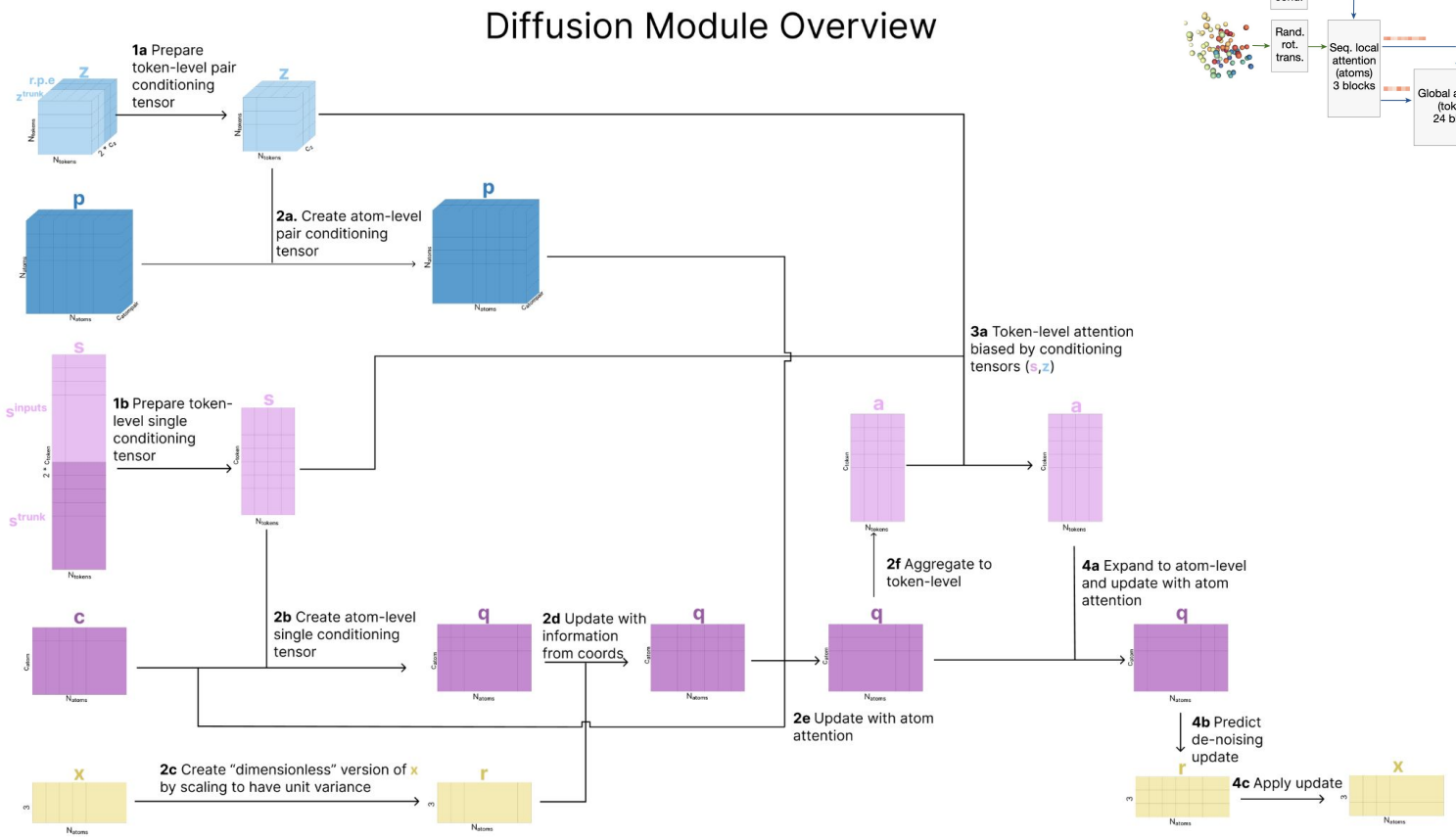
 # Rescale updates to positions and combine with input positions

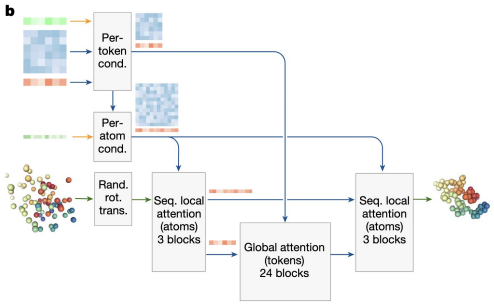
$\vec{\mathbf{x}}_l^{\text{out}} = \sigma_{\text{data}}^2/(\sigma_{\text{data}}^2 + \hat{t}^2) \cdot \vec{\mathbf{x}}_l^{\text{noisy}} + \sigma_{\text{data}} \cdot \hat{t}/\sqrt{\sigma_{\text{data}}^2 + \hat{t}^2} \cdot \mathbf{r}_l^{\text{update}}$

return $\{\vec{\mathbf{x}}_l^{\text{out}}\}$

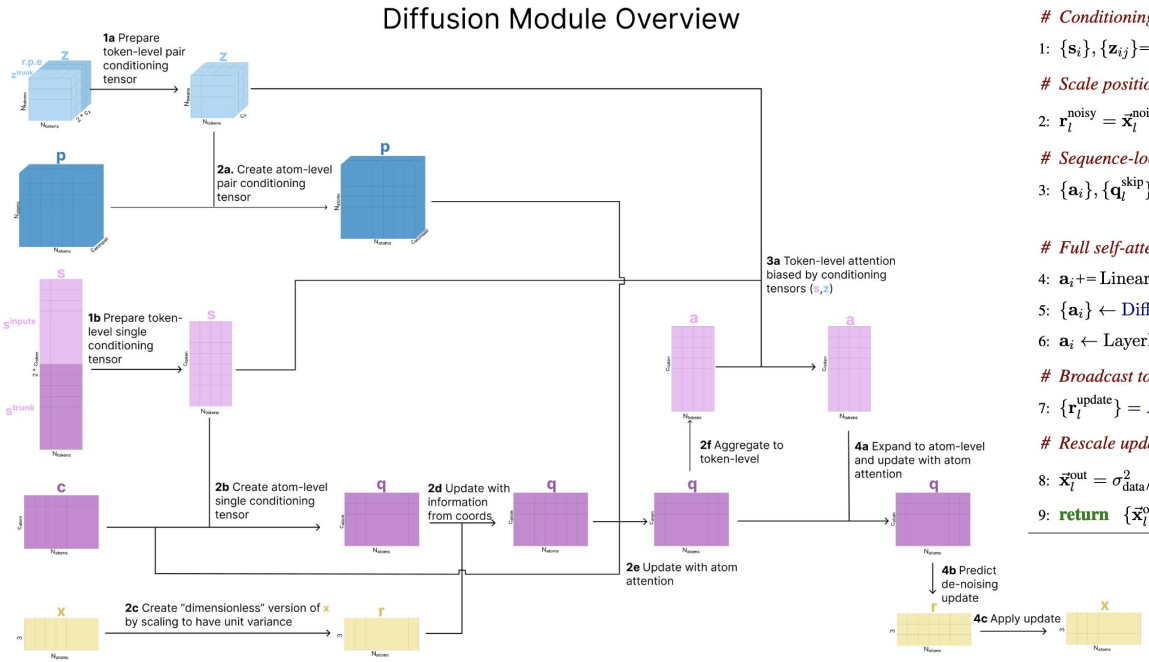


Diffusion Module





Diffusion Module



Algorithm 20 Diffusion Module

$$\textbf{def}\ \text{DiffusionModule}(\{\vec{\mathbf{x}}_l^{\text{noisy}}\}, \hat{t}, \{\mathbf{f}^*\}, \{\mathbf{s}_i^{\text{inputs}}\}, \{\mathbf{s}_i^{\text{trunk}}\}, \{\mathbf{z}_{ij}^{\text{trunk}}\}, \sigma_{\text{data}} = 16, c_{\text{atom}} = 128, c_{\text{atompair}} = 16, c_{\text{token}} = 768) :$$

 # Conditioning

$\{\mathbf{s}_i\}, \{\mathbf{z}_{ij}\} = \text{DiffusionConditioning}(\hat{t}, \{\mathbf{f}^*\}, \{\mathbf{s}_i^{\text{inputs}}\}, \{\mathbf{s}_i^{\text{trunk}}\}, \{\mathbf{z}_{ij}^{\text{trunk}}\}, \sigma_{\text{data}})$

 # Scale positions to dimensionless vectors with approximately unit variance.

$\mathbf{r}_l^{\text{noisy}} = \vec{\mathbf{x}}_l^{\text{noisy}} / \sqrt{\hat{t}^2 + \sigma_{\text{data}}^2}$ $\quad \mathbf{r}_l^{\text{noisy}} \in \mathbb{R}^3$

 # Sequence-local Atom Attention and aggregation to coarse-grained tokens

$\{\mathbf{a}_i\}, \{\mathbf{q}_l^{\text{skip}}\}, \{\mathbf{c}_l^{\text{skip}}\}, \{\mathbf{p}_{lm}^{\text{skip}}\} = \text{AtomAttentionEncoder}(\{\mathbf{f}^*\}, \{\mathbf{r}_l^{\text{noisy}}\}, \{\mathbf{s}_i^{\text{trunk}}\}, \{\mathbf{z}_{ij}\}, c_{\text{atom}}, c_{\text{atompair}}, c_{\text{token}})$ $\quad \mathbf{a}_i \in \mathbb{R}^{c_{\text{token}}}$

 # Full self-attention on token level.

$\mathbf{a}_i \mathrel{+}= \text{LinearNoBias}(\text{LayerNorm}(\mathbf{s}_i))$

$\{\mathbf{a}_i\} \leftarrow \text{DiffusionTransformer}(\{\mathbf{a}_i\}, \{\mathbf{s}_i\}, \{\mathbf{z}_{ij}\}, \beta_{ij} = 0, N_{\text{block}} = 24, N_{\text{head}} = 16)$

$\mathbf{a}_i \leftarrow \text{LayerNorm}(\mathbf{a}_i)$

 # Broadcast token activations to atoms and run Sequence-local Atom Attention

$\{\mathbf{r}_l^{\text{update}}\} = \text{AtomAttentionDecoder}(\{\mathbf{a}_i\}, \{\mathbf{q}_l^{\text{skip}}\}, \{\mathbf{c}_l^{\text{skip}}\}, \{\mathbf{p}_{lm}^{\text{skip}}\})$

 # Rescale updates to positions and combine with input positions

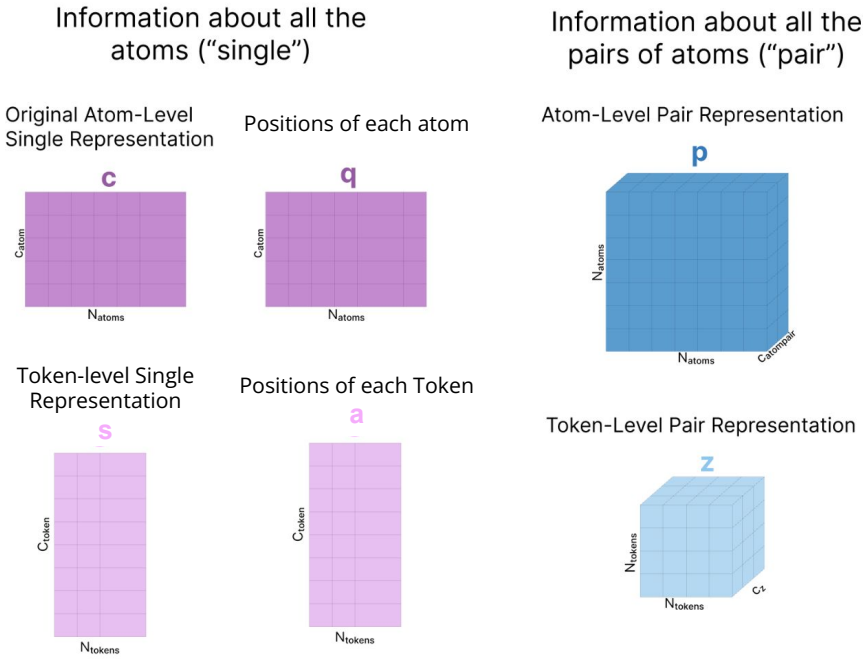
$\vec{\mathbf{x}}_l^{\text{out}} = \sigma_{\text{data}}^2 / (\sigma_{\text{data}}^2 + \hat{t}^2) \cdot \vec{\mathbf{x}}_l^{\text{noisy}} + \sigma_{\text{data}} \cdot \hat{t} / \sqrt{\sigma_{\text{data}}^2 + \hat{t}^2} \cdot \mathbf{r}_l^{\text{update}}$

return $\{\vec{\mathbf{x}}_l^{\text{out}}\}$

The AF3 paper breaks down its diffusion process into 4 steps that involve moving from tokens to atoms, back to tokens, and back to atoms:

1. **Prepare token-level conditioning tensors**
2. **Prepare atom-level conditioning tensors, update them using the Atom Transformer, and aggregate them back to token-level**
3. **Apply attention at the token-level, and project back to atoms**
4. **Apply attention at the atom-level to predict atom-level noise updates**

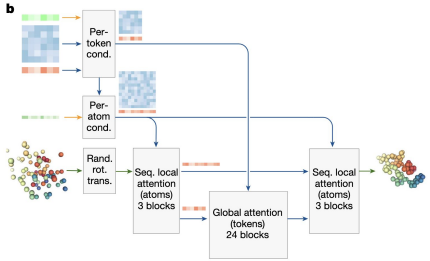
Atom-Level ⟷ Token-LeveL



atom-level representations (**c**, **q**, **p**)

token-level representations (**s**, **a**, **z**)

Diffusion Module - Diffusion Conditioning



Algorithm 20 Diffusion Module

def DiffusionModule($\{\vec{\mathbf{x}}_l^{\text{noisy}}\}, \hat{t}, \{\mathbf{f}^*\}, \{\mathbf{s}_i^{\text{inputs}}\}, \{\mathbf{s}_i^{\text{trunk}}\}, \{\mathbf{z}_{ij}^{\text{trunk}}\}$, $\sigma_{\text{data}} = 16, c_{\text{atom}} = 128, c_{\text{atompair}} = 16, c_{\text{token}} = 768$) :

 # Conditioning

$\{\mathbf{s}_i\}, \{\mathbf{z}_{ij}\} = \text{DiffusionConditioning}(\hat{t}, \{\mathbf{f}^*\}, \{\mathbf{s}_i^{\text{inputs}}\}, \{\mathbf{s}_i^{\text{trunk}}\}, \{\mathbf{z}_{ij}^{\text{trunk}}\}, \sigma_{\text{data}})$

 # Scale positions to dimensionless vectors with approximately unit variance.

$\mathbf{r}_l^{\text{noisy}} = \vec{\mathbf{x}}_l^{\text{noisy}} / \sqrt{\hat{t}^2 + \sigma_{\text{data}}^2}$ $\quad \mathbf{r}_l^{\text{noisy}} \in \mathbb{R}^3$

 # Sequence-local Atom Attention and aggregation to coarse-grained tokens

$\{\mathbf{a}_i\}, \{\mathbf{q}_l^{\text{skip}}\}, \{\mathbf{c}_l^{\text{skip}}\}, \{\mathbf{p}_{lm}^{\text{skip}}\} = \text{AtomAttentionEncoder}(\{\mathbf{f}^*\}, \{\mathbf{r}_l^{\text{noisy}}\}, \{\mathbf{s}_i^{\text{trunk}}\}, \{\mathbf{z}_{ij}\}, c_{\text{atom}}, c_{\text{atompair}}, c_{\text{token}})$ $\quad \mathbf{a}_i \in \mathbb{R}^{c_{\text{token}}}$

 # Full self-attention on token level.

$\mathbf{a}_i \mathrel{+}= \text{LinearNoBias}(\text{LayerNorm}(\mathbf{s}_i))$

$\{\mathbf{a}_i\} \leftarrow \text{DiffusionTransformer}(\{\mathbf{a}_i\}, \{\mathbf{s}_i\}, \{\mathbf{z}_{ij}\}, \beta_{ij} = 0, N_{\text{block}} = 24, N_{\text{head}} = 16)$

$\mathbf{a}_i \leftarrow \text{LayerNorm}(\mathbf{a}_i)$

 # Broadcast token activations to atoms and run Sequence-local Atom Attention

$\{\mathbf{r}_l^{\text{update}}\} = \text{AtomAttentionDecoder}(\{\mathbf{a}_i\}, \{\mathbf{q}_l^{\text{skip}}\}, \{\mathbf{c}_l^{\text{skip}}\}, \{\mathbf{p}_{lm}^{\text{skip}}\})$

 # Rescale updates to positions and combine with input positions

$\vec{\mathbf{x}}_l^{\text{out}} = \sigma_{\text{data}}^2 / (\sigma_{\text{data}}^2 + \hat{t}^2) \cdot \vec{\mathbf{x}}_l^{\text{noisy}} + \sigma_{\text{data}} \cdot \hat{t} / \sqrt{\sigma_{\text{data}}^2 + \hat{t}^2} \cdot \mathbf{r}_l^{\text{update}}$

return $\{\vec{\mathbf{x}}_l^{\text{out}}\}$

Algorithm 21 Diffusion Conditioning

def DiffusionConditioning($\hat{t}, \{\mathbf{f}^*\}, \{\mathbf{s}_i^{\text{inputs}}\}, \{\mathbf{s}_i^{\text{trunk}}\}, \{\mathbf{z}_{ij}^{\text{trunk}}\}, \sigma_{\text{data}}, c_{\text{z}} = 128, c_{\text{s}} = 384$) :

 # Pair conditioning

$\mathbf{z}_{ij} = \text{concat}([\mathbf{z}_{ij}^{\text{trunk}}, \text{RelativePositionEncoding}(\{\mathbf{f}^*\})])$

$\mathbf{z}_{ij} \leftarrow \text{LinearNoBias}(\text{LayerNorm}(\mathbf{z}_{ij}))$ $\quad \mathbf{z}_{ij} \in \mathbb{R}^{c_z}$

for all $b \in [1, 2]$ **do**

  $\mathbf{z}_{ij} \mathrel{+}= \text{Transition}(\mathbf{z}_{ij}, n = 2)$

end for

 # Single conditioning

$\mathbf{s}_i = \text{concat}([\mathbf{s}_i^{\text{trunk}}, \mathbf{s}_i^{\text{inputs}}])$

$\mathbf{s}_i \leftarrow \text{LinearNoBias}(\text{LayerNorm}(\mathbf{s}_i))$ $\quad \mathbf{s}_i \in \mathbb{R}^{c_s}$

$\mathbf{n} = \text{FourierEmbedding}(\frac{1}{4} \log(\hat{t}/\sigma_{\text{data}}), 256)$

$\mathbf{s}_i \mathrel{+}= \text{LinearNoBias}(\text{LayerNorm}(\mathbf{n}))$

for all $b \in [1, 2]$ **do**

  $\mathbf{s}_i \mathrel{+}= \text{Transition}(\mathbf{s}_i, n = 2)$

end for

return $\{\mathbf{s}_i\}, \{\mathbf{z}_{ij}\}$

Diffusion Module

Algorithm 20 Diffusion Module

def DiffusionModule($\{\vec{\mathbf{x}}_l^{\text{noisy}}\}, \hat{t}, \{\mathbf{f}^*\}, \{\mathbf{s}_i^{\text{inputs}}\}, \{\mathbf{s}_i^{\text{trunk}}\}, \{\mathbf{z}_{ij}^{\text{trunk}}\}$,
$\sigma_{\text{data}} = 16, c_{\text{atom}} = 128, c_{\text{atompair}} = 16, c_{\text{token}} = 768$) :

 # Conditioning

$\{\mathbf{s}_i\}, \{\mathbf{z}_{ij}\} = \text{DiffusionConditioning}(\hat{t}, \{\mathbf{f}^*\}, \{\mathbf{s}_i^{\text{inputs}}\}, \{\mathbf{s}_i^{\text{trunk}}\}, \{\mathbf{z}_{ij}^{\text{trunk}}\}, \sigma_{\text{data}})$

 # Scale positions to dimensionless vectors with approximately unit variance.

$\mathbf{r}_l^{\text{noisy}} = \vec{\mathbf{x}}_l^{\text{noisy}} / \sqrt{\hat{t}^2 + \sigma_{\text{data}}^2}$ $\qquad \mathbf{r}_l^{\text{noisy}} \in \mathbb{R}^3$

 # Sequence-local Atom Attention and aggregation to coarse-grained tokens

$\{\mathbf{a}_i\}, \{\mathbf{q}_l^{\text{skip}}\}, \{\mathbf{c}_l^{\text{skip}}\}, \{\mathbf{p}_{lm}^{\text{skip}}\} = \text{AtomAttentionEncoder}(\{\mathbf{f}^*\}, \{\mathbf{r}_l^{\text{noisy}}\}, \{\mathbf{s}_i^{\text{trunk}}\}, \{\mathbf{z}_{ij}\}, c_{\text{atom}}, c_{\text{atompair}}, c_{\text{token}})$
$\mathbf{a}_i \in \mathbb{R}^{c_{\text{token}}}$

 # Full self-attention on token level.

$\mathbf{a}_i \mathrel{+}= \text{LinearNoBias}(\text{LayerNorm}(\mathbf{s}_i))$

$\{\mathbf{a}_i\} \leftarrow \text{DiffusionTransformer}(\{\mathbf{a}_i\}, \{\mathbf{s}_i\}, \{\mathbf{z}_{ij}\}, \beta_{ij} = 0, N_{\text{block}} = 24, N_{\text{head}} = 16)$

$\mathbf{a}_i \leftarrow \text{LayerNorm}(\mathbf{a}_i)$

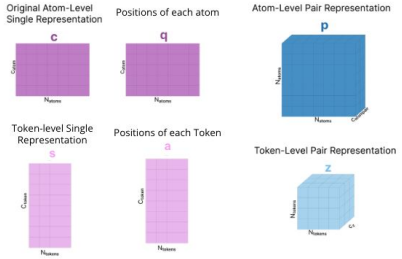
 # Broadcast token activations to atoms and run Sequence-local Atom Attention

$\{\mathbf{r}_l^{\text{update}}\} = \text{AtomAttentionDecoder}(\{\mathbf{a}_i\}, \{\mathbf{q}_l^{\text{skip}}\}, \{\mathbf{c}_l^{\text{skip}}\}, \{\mathbf{p}_{lm}^{\text{skip}}\})$

 # Rescale updates to positions and combine with input positions

$\vec{\mathbf{x}}_l^{\text{out}} = \sigma_{\text{data}}^2 / (\sigma_{\text{data}}^2 + \hat{t}^2) \cdot \vec{\mathbf{x}}_l^{\text{noisy}} + \sigma_{\text{data}} \cdot \hat{t} / \sqrt{\sigma_{\text{data}}^2 + \hat{t}^2} \cdot \mathbf{r}_l^{\text{update}}$

return $\{\vec{\mathbf{x}}_l^{\text{out}}\}$



Algorithm 5 Atom attention encoder

def AtomAttentionEncoder($\{\mathbf{f}^*\}, \{\mathbf{r}_l\}, \{\mathbf{s}_i^{\text{trunk}}\}, \{\mathbf{z}_{ij}\}, c_{\text{atom}}, c_{\text{atompair}}, c_{\text{token}}$) :

 # Create the atom single conditioning: Embed per-atom meta data

$\mathbf{c}_l = \text{LinearNoBias}(\text{concat}(\vec{\mathbf{f}}_l^{\text{ref\_pos}}, \mathbf{f}_l^{\text{ref\_charge}}, \mathbf{f}_l^{\text{ref\_mask}}, \mathbf{f}_l^{\text{ref\_element}}, \mathbf{f}_l^{\text{ref\_atom\_name\_chars}}))$ $\qquad \mathbf{c}_l \in \mathbb{R}^{c_{\text{atom}}}$
$l \in \{1, \ldots, N_{\text{atoms}}\}$

 # Cross attention transformer.

$\{\mathbf{q}_l\} = \text{AtomTransformer}(\{\mathbf{q}_l\}, \{\mathbf{c}_l\}, \{\mathbf{p}_{lm}\}, N_{\text{block}} = 3, N_{\text{head}} = 4)$

 # Aggregate per-atom representation to per-token representation

$\mathbf{a}_i = \underset{\substack{l \in \{1,\ldots,N_{\text{atoms}}\} \\ \text{tok\_idx}(l) = i}}{\text{mean}} (\text{relu}(\text{LinearNoBias}(\mathbf{q}_l)))$ $\qquad \mathbf{a}_i \in \mathbb{R}^{c_{\text{token}}}$

$\mathbf{q}_l^{\text{skip}}, \mathbf{c}_l^{\text{skip}}, \mathbf{p}_{lm}^{\text{skip}} = \mathbf{q}_l, \mathbf{c}_l, \mathbf{p}_{lm}$

return $\{\mathbf{a}_i\}, \{\mathbf{q}_l^{\text{skip}}\}, \{\mathbf{c}_l^{\text{skip}}\}, \{\mathbf{p}_{lm}^{\text{skip}}\}$

Algorithm 6 Atom attention decoder

def AtomAttentionDecoder($\{\mathbf{a}_i\}, \{\mathbf{q}_l^{\text{skip}}\}, \{\mathbf{c}_l^{\text{skip}}\}, \{\mathbf{p}_{lm}^{\text{skip}}\}$) :

 # Broadcast per-token activiations to per-atom activations and add the skip connection

$\mathbf{q}_l = \text{LinearNoBias}(\mathbf{a}_{\text{tok\_idx}(l)}) + \mathbf{q}_l^{\text{skip}}$

 # Cross attention transformer.

$\{\mathbf{q}_l\} = \text{AtomTransformer}(\{\mathbf{q}_l\}, \{\mathbf{c}_l^{\text{skip}}\}, \{\mathbf{p}_{lm}^{\text{skip}}\}, N_{\text{block}} = 3, N_{\text{head}} = 4)$

 # Map to positions update.

$\mathbf{r}_l^{\text{update}} = \text{LinearNoBias}(\text{LayerNorm}(\mathbf{q}_l))$

return $\{\mathbf{r}_l^{\text{update}}\}$

Algorithm 7 Atom Transformer

def AtomTransformer($\{\mathbf{q}_l\}, \{\mathbf{c}_l\}, \{\mathbf{p}_{lm}\}, N_{\text{block}} = 3, N_{\text{head}}$,
$N_{\text{queries}} = 32, N_{\text{keys}} = 128, \mathcal{S}_{\text{subset centres}} = \{15.5, 47.5, 79.5, \ldots\}$) :

 # sequence-local atom attention is equivalent to self attention within rectangular blocks along the diagonal.

$$\beta_{lm} = \begin{cases} 0 & \text{if } |l-c| < N_{\text{queries}}/2 \wedge |m-c| < N_{\text{keys}}/2 \quad \forall\, c \in \mathcal{S}_{\text{subset centres}} \\ -10^{10} & \text{else} \end{cases}$$

$\{\mathbf{q}_l\} = \text{DiffusionTransformer}(\{\mathbf{q}_l\}, \{\mathbf{c}_l\}, \{\mathbf{p}_{lm}\}, \{\beta_{lm}\}, N_{\text{block}}, N_{\text{head}})$

return $\{\mathbf{q}_l\}$

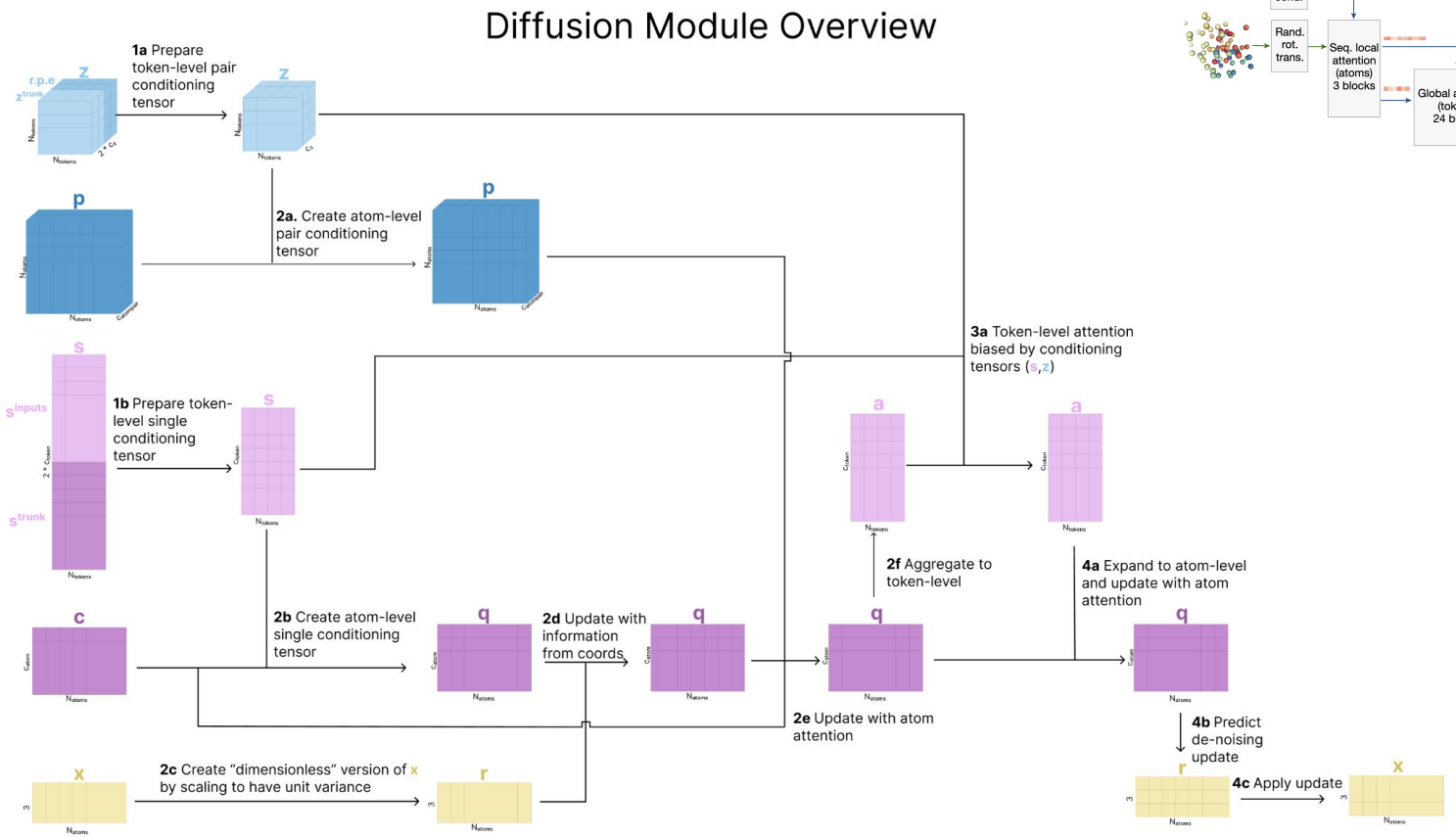
Algorithm 23 Diffusion Transformer

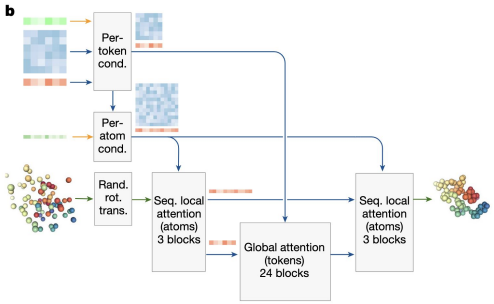
def DiffusionTransformer($\{\mathbf{a}_i\}, \{\mathbf{s}_i\}, \{\mathbf{z}_{ij}\}, \{\beta_{ij}\}, N_{\text{block}}, N_{\text{head}}$):

for all $n \in [1, \ldots, N_{\text{block}}]$ **do**
  $\{\mathbf{b}_i\} = \text{AttentionPairBias}(\{\mathbf{a}_i\}, \{\mathbf{s}_i\}, \{\mathbf{z}_{ij}\}, \{\beta_{ij}\}, N_{\text{head}})$
  $\mathbf{a}_i \leftarrow \mathbf{b}_i + \text{ConditionedTransitionBlock}(\mathbf{a}_i, \mathbf{s}_i)$
end for

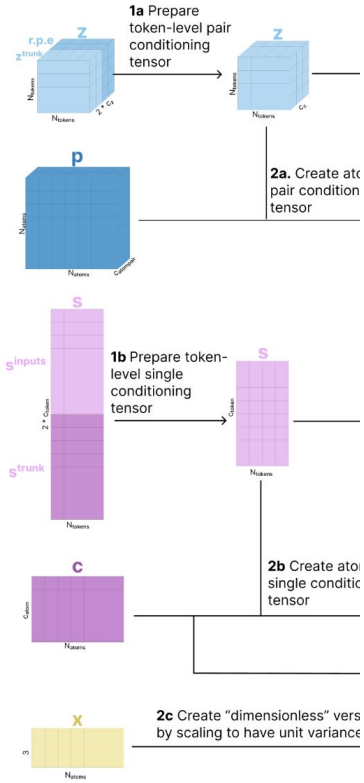
return $\{\mathbf{a}_i\}$

Diffusion Module

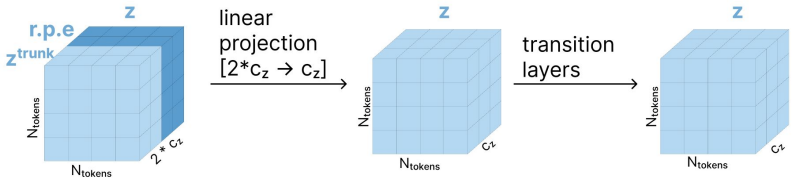




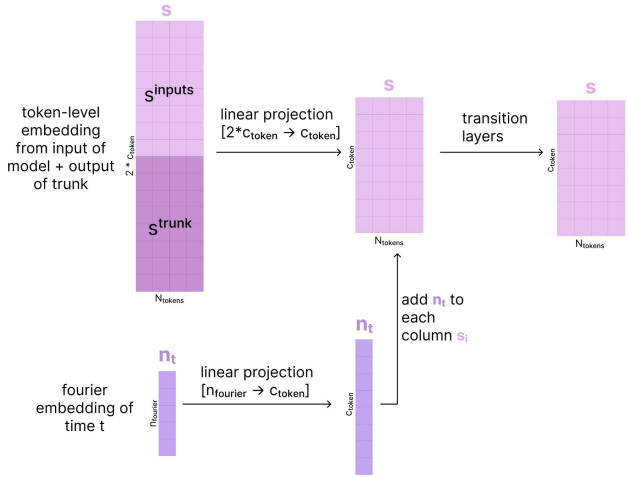
Diffusion Module - Part 1



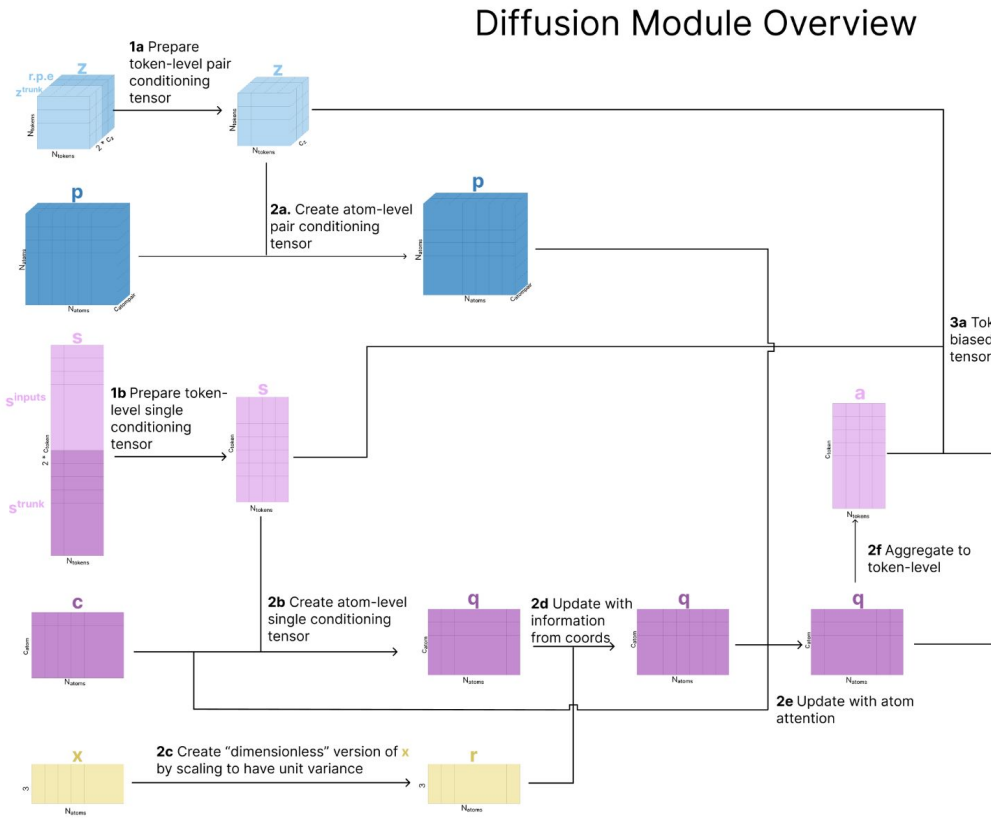
1a Create token-level pair conditioning input (z)



1b Create token-level single conditioning input (s)

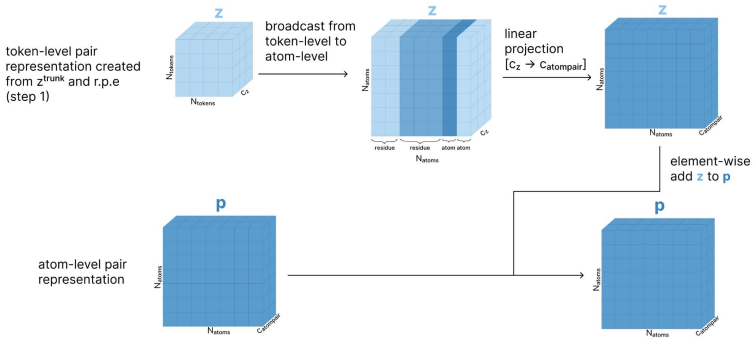


Diffusion Module - Part 2



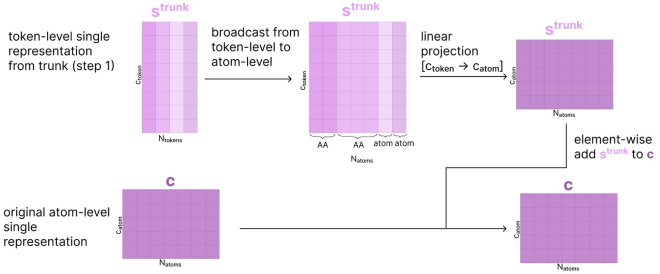
2a

Create atom-level pair conditioning input (p)



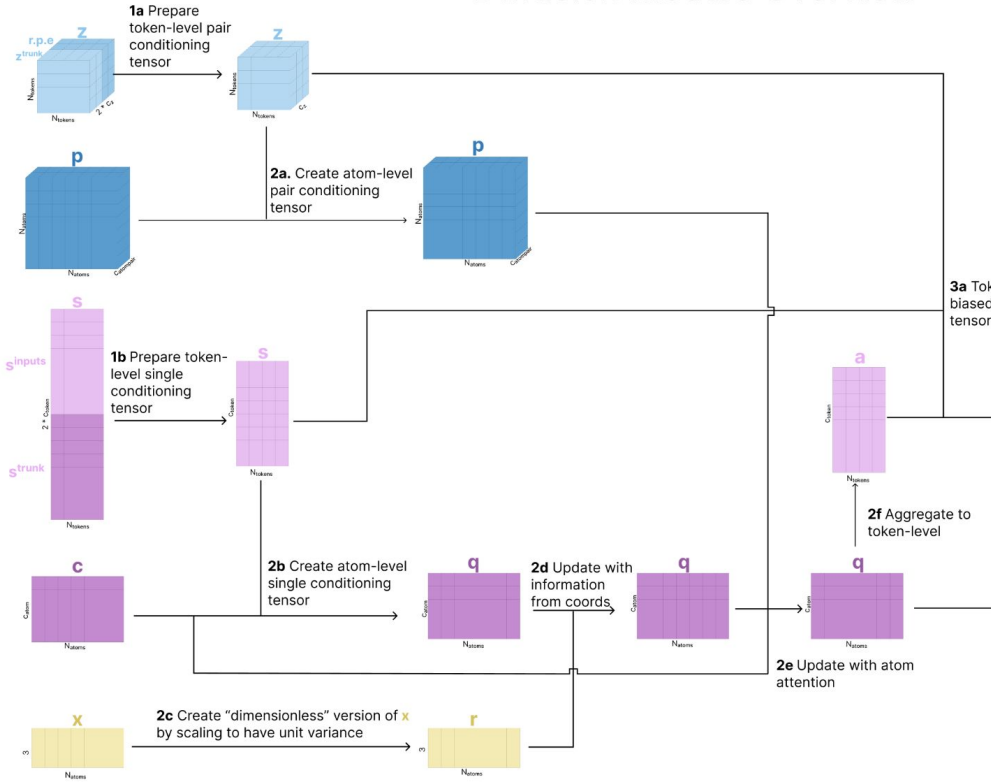
2b

Create atom-level single conditioning input (c)



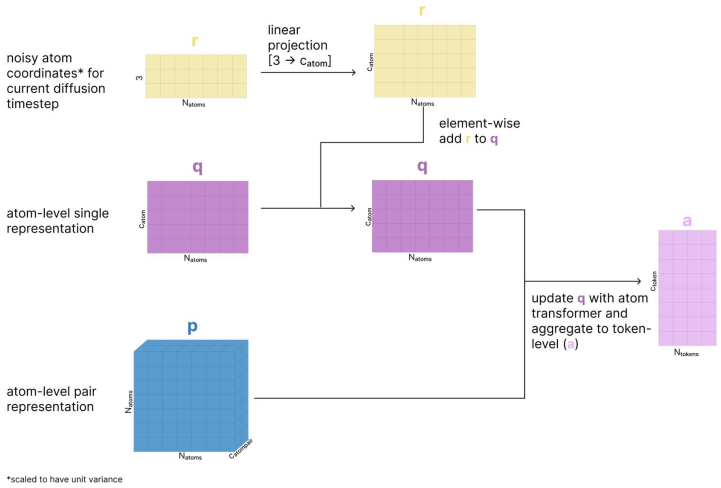
Diffusion Module - Part 2

Diffusion Module Overview

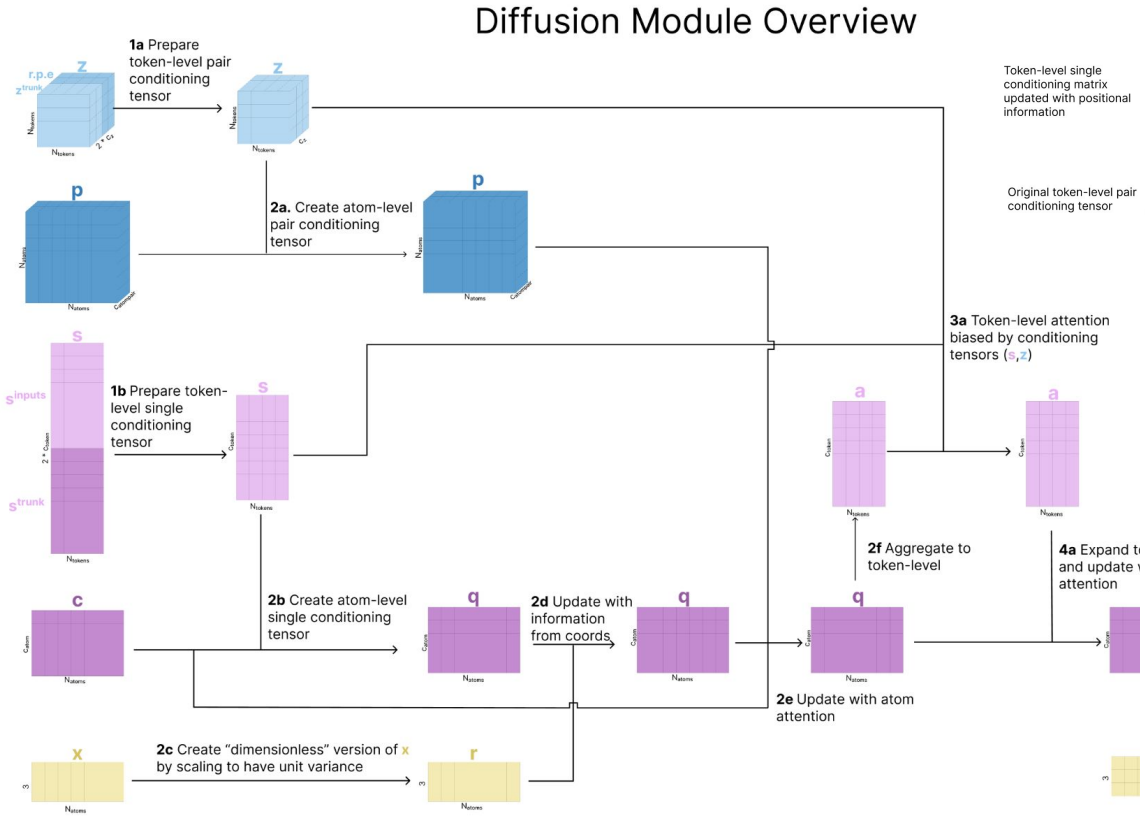


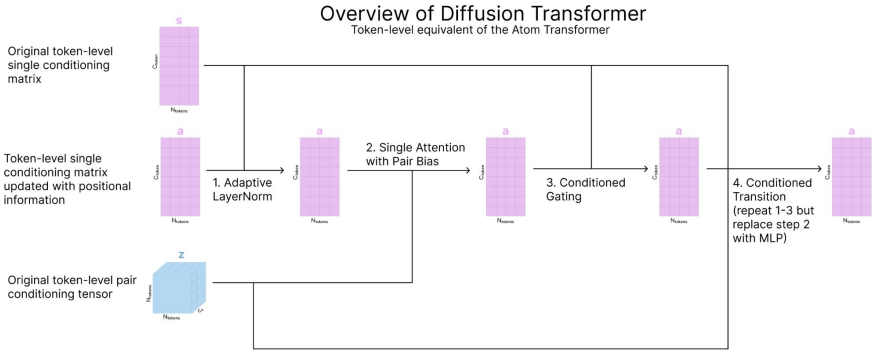
2c-f

Add noisy coordinates to atom-level representation and convert to token-level



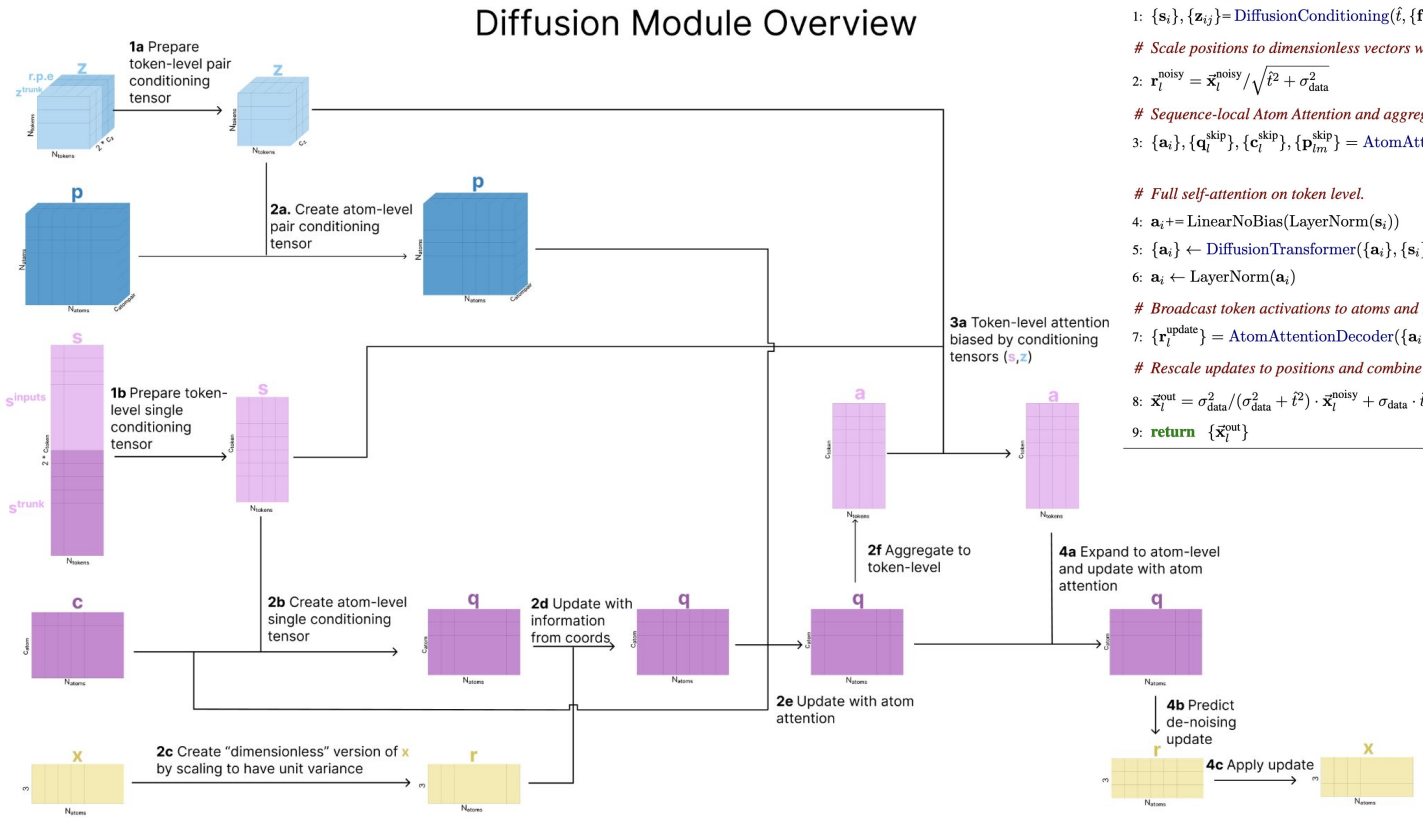
Diffusion Module - Part 3





3a

Diffusion Module - Part 4



Algorithm 20 Diffusion Module

def DiffusionModule($\{\vec{\mathbf{x}}_l^{\text{noisy}}\}, \hat{t}, \{\mathbf{f}^*\}, \{\mathbf{s}_i^{\text{inputs}}\}, \{\mathbf{s}_i^{\text{trunk}}\}, \{\mathbf{z}_{ij}^{\text{trunk}}\}$, $\sigma_{\text{data}} = 16, c_{\text{atom}} = 128, c_{\text{atompair}} = 16, c_{\text{token}} = 768$) :

 # Conditioning

$\{\mathbf{s}_i\}, \{\mathbf{z}_{ij}\} = \text{DiffusionConditioning}(\hat{t}, \{\mathbf{f}^*\}, \{\mathbf{s}_i^{\text{inputs}}\}, \{\mathbf{s}_i^{\text{trunk}}\}, \{\mathbf{z}_{ij}^{\text{trunk}}\}, \sigma_{\text{data}})$

 # Scale positions to dimensionless vectors with approximately unit variance.

$\mathbf{r}_l^{\text{noisy}} = \vec{\mathbf{x}}_l^{\text{noisy}} / \sqrt{\hat{t}^2 + \sigma_{\text{data}}^2}$ $\quad \mathbf{r}_l^{\text{noisy}} \in \mathbb{R}^3$

 # Sequence-local Atom Attention and aggregation to coarse-grained tokens

$\{\mathbf{a}_i\}, \{\mathbf{q}_l^{\text{skip}}\}, \{\mathbf{c}_l^{\text{skip}}\}, \{\mathbf{p}_{lm}^{\text{skip}}\} = \text{AtomAttentionEncoder}(\{\mathbf{f}^*\}, \{\mathbf{r}_l^{\text{noisy}}\}, \{\mathbf{s}_i^{\text{trunk}}\}, \{\mathbf{z}_{ij}\}, c_{\text{atom}}, c_{\text{atompair}}, c_{\text{token}})$ $\quad \mathbf{a}_i \in \mathbb{R}^{c_{\text{token}}}$

 # Full self-attention on token level.

$\mathbf{a}_i \mathrel{+}= \text{LinearNoBias}(\text{LayerNorm}(\mathbf{s}_i))$

$\{\mathbf{a}_i\} \leftarrow \text{DiffusionTransformer}(\{\mathbf{a}_i\}, \{\mathbf{s}_i\}, \{\mathbf{z}_{ij}\}, \beta_{ij} = 0, N_{\text{block}} = 24, N_{\text{head}} = 16)$

$\mathbf{a}_i \leftarrow \text{LayerNorm}(\mathbf{a}_i)$

 # Broadcast token activations to atoms and run Sequence-local Atom Attention

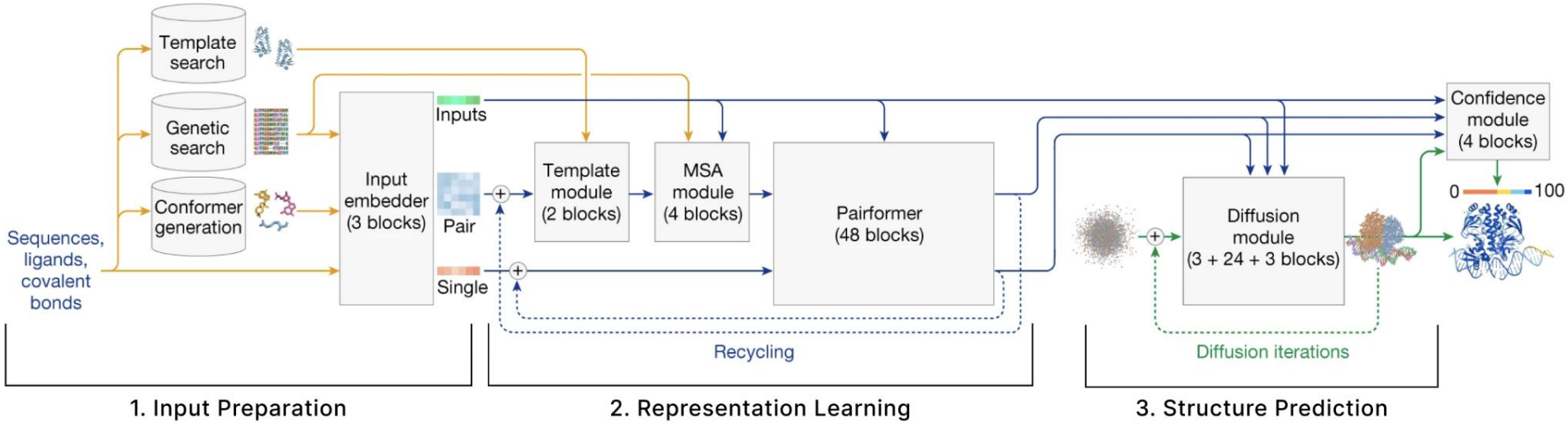
$\{\mathbf{r}_l^{\text{update}}\} = \text{AtomAttentionDecoder}(\{\mathbf{a}_i\}, \{\mathbf{q}_l^{\text{skip}}\}, \{\mathbf{c}_l^{\text{skip}}\}, \{\mathbf{p}_{lm}^{\text{skip}}\})$

 # Rescale updates to positions and combine with input positions

$\vec{\mathbf{x}}_l^{\text{out}} = \sigma_{\text{data}}^2 / (\sigma_{\text{data}}^2 + \hat{t}^2) \cdot \vec{\mathbf{x}}_l^{\text{noisy}} + \sigma_{\text{data}} \cdot \hat{t} / \sqrt{\sigma_{\text{data}}^2 + \hat{t}^2} \cdot \mathbf{r}_l^{\text{update}}$

return $\{\vec{\mathbf{x}}_l^{\text{out}}\}$

Architecture Overview



Loss function and confidence heads

$$L_{\text{loss}} = L_{\text{distogram}} * \alpha_{\text{distogram}} + L_{\text{diffusion}} * \alpha_{\text{diffusion}} + L_{\text{confidence}} * \alpha_{\text{confidence}}$$

The loss is a weighted sum of 3 terms:

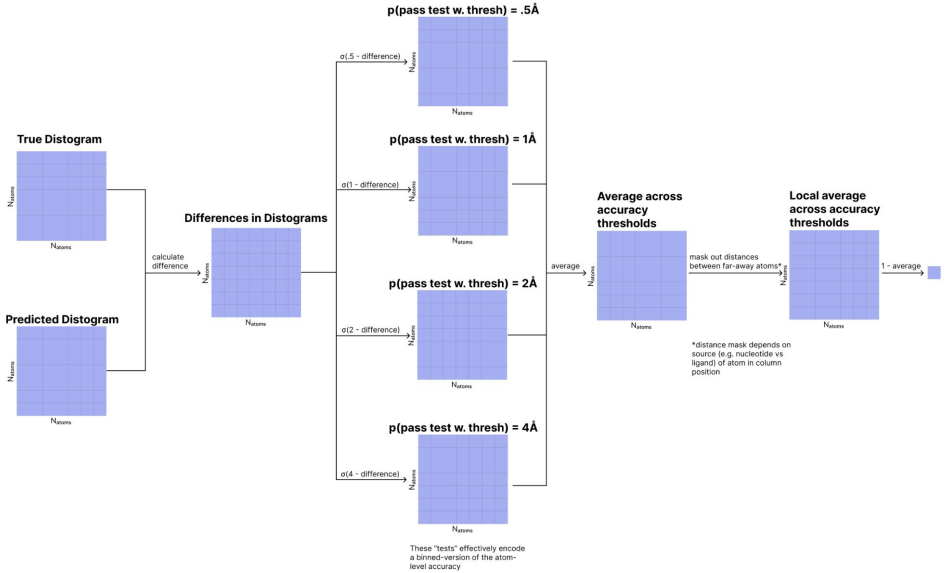
- $L_{\text{distogram}}$ which evaluates the accuracy of the predicted distogram at a token-level
- $L_{\text{diffusion}}$ which evaluates the accuracy of the predicted distogram at an atom-level. It looks at all pairwise distances then includes additional terms to prioritize distances between nearby atoms and atoms involved in protein-ligand bonds.
- $L_{\text{confidence}}$ which evaluates the model's self-awareness about which structures are likely to be inaccurate

Diffusion Loss

$$L_{\text{diffusion}} = (L_{\text{MSE}} + L_{\text{bond}} \cdot \alpha_{\text{bond}}) \cdot \frac{\hat{t}^2 + \sigma_{\text{data}}^2}{(\hat{t} + \sigma_{\text{data}})^2} + L_{\text{smooth\_lddt}}$$

- **LMSE**
  - Mean squared error over *all* atom positions.
  - Similar to distogram loss but not binned.
  - DNA, RNA, ligand atoms upweighted.
- **Lbond**
  - Focuses on protein-ligand bond lengths.
  - Extra MSE applied to protein-ligand atom pairs.
- **Lsmooth_LDDT**
  - Soft version of LDDT (local distance difference test).
  - Smooth probability of passing distance thresholds.
  - Uses 4 thresholds: 4Å, 2Å, 1Å, 0.5Å.
  - Ignores far atom pairs to focus on local accuracy.

Smooth Local Distance Difference Test (LDDT)

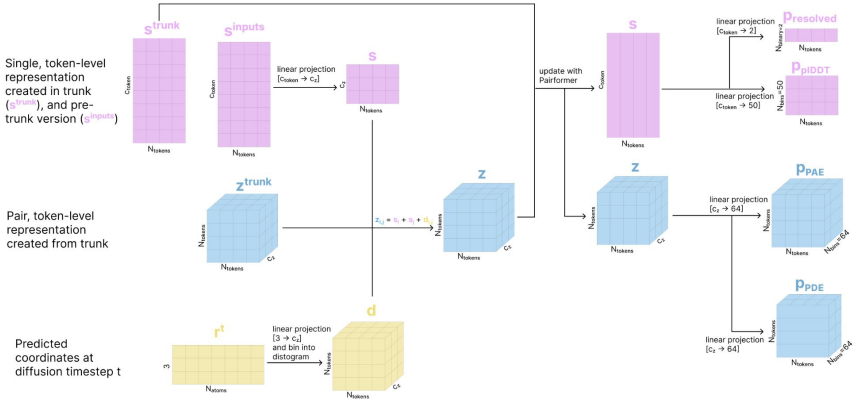


Confidence Loss

$$L_{\text{confidence}} = L_{\text{plDDT}} + L_{\text{PDE}} + L_{\text{resolved}} + L_{\text{PAE}} \cdot \alpha_{\text{PAE}}$$

- **plDDT (Predicted LDDT):**
  Predicts atom-level local distance accuracy.
- **PAE (Predicted Alignment Error):**
  Predicts how far a token is from its true position after aligning with another token.
- **PDE (Predicted Distance Error):**
  Predicts errors in pairwise distances between tokens.
- **Experimentally Resolved Prediction:**
  Predicts which atoms are experimentally resolved in the structure.

Confidence Head Architecture



Confidence Loss

$$L_{\text{confidence}} = L_{\text{plDDT}} + L_{\text{PDE}} + L_{\text{resolved}} + L_{\text{PAE}} \cdot \alpha_{\text{PAE}}$$

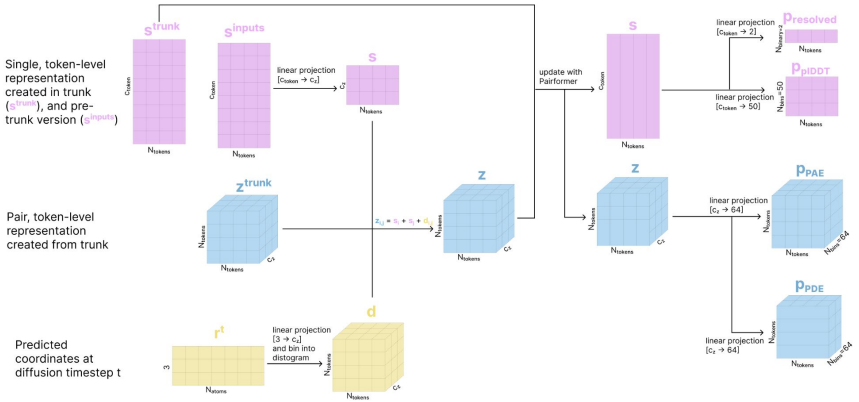
How Confidence Loss Works

- The model predicts error metrics during diffusion.
- True error metrics are calculated on predicted structures.
- Loss measures the difference between predicted vs actual errors.
- Even if structure is wrong, the loss is low if the model correctly predicts its own uncertainty.

Gradients Only Affect Confidence Heads

- Confidence loss updates only the confidence prediction heads.
- Main structure prediction model is unaffected.

Confidence Head Architecture



Other Training Details

Beyond architecture, several training strategies improve AF3 performance:

- Recycling
- Cross-distillation
- Cropping & staged training
- Clashing handling
- Batch size optimization

Recycling

- **Same weights reused multiple times.**
- Inputs pass through modules iteratively → progressively refined representations.
- Diffusion model inherently uses recycling during inference, using timestep information.
- Inspired by AlphaFold 2 (AF2).

Cross-Distillation

- AF3 uses:
  - **Self-distillation**: learning from its own predictions.
  - **Cross-distillation**: learning from AF2 and AF-Multimer predictions.
- Motivation:
  - Diffusion eliminates AF2's characteristic "spaghetti" low-confidence regions.
  - Including AF2 generations helps AF3 learn to output unfolded, low-confidence regions when appropriate.

Cropping & Training Stages

- Long sequences are computationally expensive.
- Cropping strategies:
  - **Contiguous cropping**: random continuous segments.
  - **Spatial cropping**: residues near a reference atom.
  - **Spatial interface cropping**: residues near binding interfaces.
- Progressive fine-tuning on longer sequences to generalize better.

Table 6 | Training stages

	Initial training	Fine tuning 1	Fine tuning 2	Fine tuning 3
Sequence crop size N_{token}	384	640	768	768
Parameters initialized from	Random	Initial training	Fine tuning 1	Fine tuning 2
Sampling weight for disorder PDB distillation	0.02	0.01	0.02	0.02
Train on transcription factor distillation sets	False	False	True	True
Masked diffusion loss for non-protein in disorder PDB distillation	True	False	False	False
Train structure and distogram	True	True	True	False
Train PAE head	False	False	False	True
Diffusion batch size	48	32	32	32
Training samples ($\cdot 10^6$)	≈20	≈1.5	≈1.5	≈1.8
Training times (days on 256 A100s)	≈10	≈3	≈5	≈2
Polymer-ligand bond loss weight	0	1	1	1
Max number of chains	20	20	20	50

Clashing

- No explicit clash penalty in diffusion loss.
- Overlapping atoms rarely occur after training.
- **Clash penalty applied only during ranking of generated structures.**

Batch Size Optimization

- Diffusion step is lighter than representation trunk.
- Strategy:
  - Pass input once through trunk.
  - Apply **48 parallel data augmentations** on trunk outputs.
  - Efficiently trains many augmented versions simultaneously.

ML Design Insights

Pair-Bias Attention

- Self-attention + additional pair-wise bias.

Self-Supervised Training

- AF3 does not use large-scale self-supervised pretraining on MSA.
- Possible reasons: compute cost, supervised hybrid model works better, mixed token types (AA, DNA, RNA, ligands).

Classification vs Regression

- Uses both MSE and binned classification losses.
- Binning may provide more stable gradients.

Similarities to Recurrent Networks

- Gating mechanisms (like LSTMs/GRUs).
- Iterative weight reuse.
- Adaptive computation reminiscent of recurrent models and ACT.

End of AF3

Thanks for listening!

Questions - Comments?

Algorithm 1 Main Inference Loop

def MainInferenceLoop($\{\mathbf{f}^*\}, N_{\text{cycle}} = 4, c_s = 384, c_z = 128$) :

$\{\mathbf{s}_i^{\text{inputs}}\} = \text{InputFeatureEmbedder}(\{\mathbf{f}^*\})$

$\mathbf{s}_i^{\text{init}} = \text{LinearNoBias}(\mathbf{s}_i^{\text{inputs}})$ $\mathbf{s}_i^{\text{init}} \in \mathbb{R}^{c_s}$

$\mathbf{z}_{ij}^{\text{init}} = \text{LinearNoBias}(\mathbf{s}_i^{\text{inputs}}) + \text{LinearNoBias}(\mathbf{s}_j^{\text{inputs}})$ $\mathbf{z}_{ij}^{\text{init}} \in \mathbb{R}^{c_z}$

$\mathbf{z}_{ij}^{\text{init}} \mathrel{+}= \text{RelativePositionEncoding}(\{\mathbf{f}^*\})$

$\mathbf{z}_{ij}^{\text{init}} \mathrel{+}= \text{LinearNoBias}(\mathbf{f}_{ij}^{\text{token\_bonds}})$

$\{\hat{\mathbf{z}}_{ij}\}, \{\hat{\mathbf{s}}_i\} = \mathbf{0}, \mathbf{0}$

for all $c \in [1, \ldots, N_{\text{cycle}}]$ **do**

  $\mathbf{z}_{ij} = \mathbf{z}_{ij}^{\text{init}} + \text{LinearNoBias}(\text{LayerNorm}(\hat{\mathbf{z}}_{ij}))$ $\mathbf{z}_{ij} \in \mathbb{R}^{c_z}$

  $\{\mathbf{z}_{ij}\} \mathrel{+}= \text{TemplateEmbedder}(\{\mathbf{f}^*\}, \{\mathbf{z}_{ij}\})$

  $\{\mathbf{z}_{ij}\} \mathrel{+}= \text{MsaModule}(\{\mathbf{f}_{Si}^{\text{msa}}\}, \{\mathbf{z}_{ij}\}, \{\mathbf{s}_i^{\text{inputs}}\})$

  $\mathbf{s}_i = \mathbf{s}_i^{\text{init}} + \text{LinearNoBias}(\text{LayerNorm}(\hat{\mathbf{s}}_i))$ $\mathbf{s}_i \in \mathbb{R}^{c_s}$

  $\{\mathbf{s}_i\}, \{\mathbf{z}_{ij}\} = \text{PairformerStack}(\{\mathbf{s}_i\}, \{\mathbf{z}_{ij}\})$

  $\{\hat{\mathbf{s}}_i\}, \{\hat{\mathbf{z}}_{ij}\} \leftarrow \{\mathbf{s}_i\}, \{\mathbf{z}_{ij}\}$

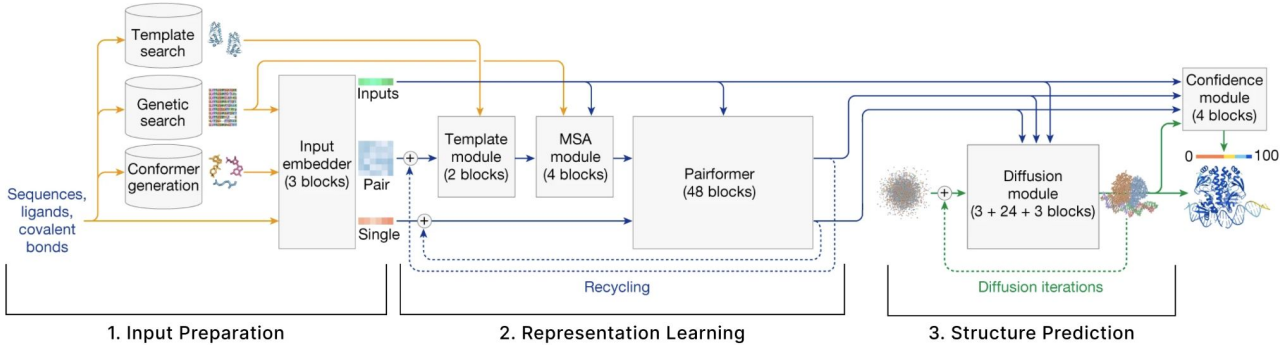
end for

$\{\vec{\mathbf{x}}_l^{\text{pred}}\} = \text{SampleDiffusion}(\{\mathbf{f}^*\}, \{\mathbf{s}_i^{\text{inputs}}\}, \{\mathbf{s}_i\}, \{\mathbf{z}_{ij}\})$

$\{\mathbf{p}_l^{\text{plddt}}\}, \{\mathbf{p}_{ij}^{\text{pae}}\}, \{\mathbf{p}_{ij}^{\text{pde}}\}, \{\mathbf{p}_l^{\text{resolved}}\} = \text{ConfidenceHead}(\{\mathbf{s}_i^{\text{inputs}}\}, \{\mathbf{s}_i\}, \{\mathbf{z}_{ij}\}, \{\vec{\mathbf{x}}_l^{\text{pred}}\})$

$\mathbf{p}_{ij}^{\text{distogram}} = \text{DistogramHead}(\mathbf{z}_{ij})$

return $\{\vec{\mathbf{x}}_l^{\text{pred}}\}, \{\mathbf{p}_l^{\text{plddt}}\}, \{\mathbf{p}_{ij}^{\text{pae}}\}, \{\mathbf{p}_{ij}^{\text{pde}}\}, \{\mathbf{p}_l^{\text{resolved}}\}, \{\mathbf{p}_{ij}^{\text{distogram}}\}$



Resource

🔗 The Illustrated AlphaFold

🔗 AlphaFold3 and its improvements in comparison to AlphaFold2

🔗 AlphaFold2, AlphaFold-Multimer, AlphaFold3

🔗 "Sparks of Chemical Intuition"-and Gross Limitations!-in AlphaFold 3

🔗 AlphaFold3 Explained

▶️ AlphaFold 3 deep dive | Looking Glass Universe

🔗 Introduction to diffusion models for machine learning

🔗 What are Diffusion Models?

▶️ Diffusion Models for AI Image Generation | IBM Technology