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

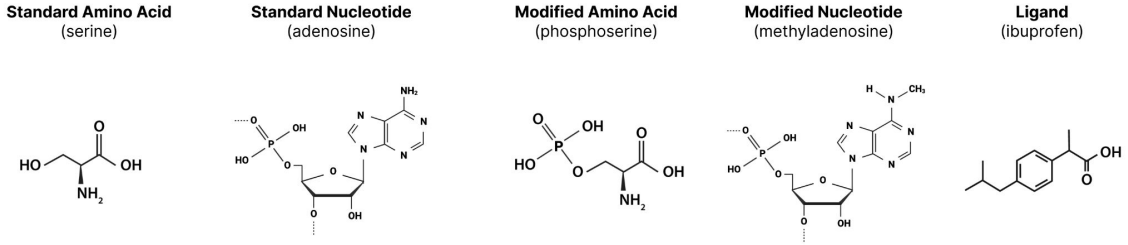
From Sequences to Multi-molecular Structures

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**

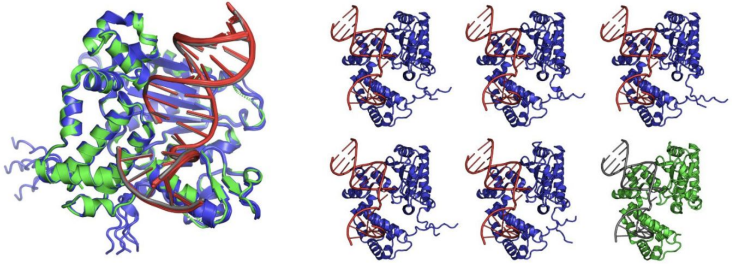


AlphaFold 3 is not just an extension; it's a **new paradigm**.

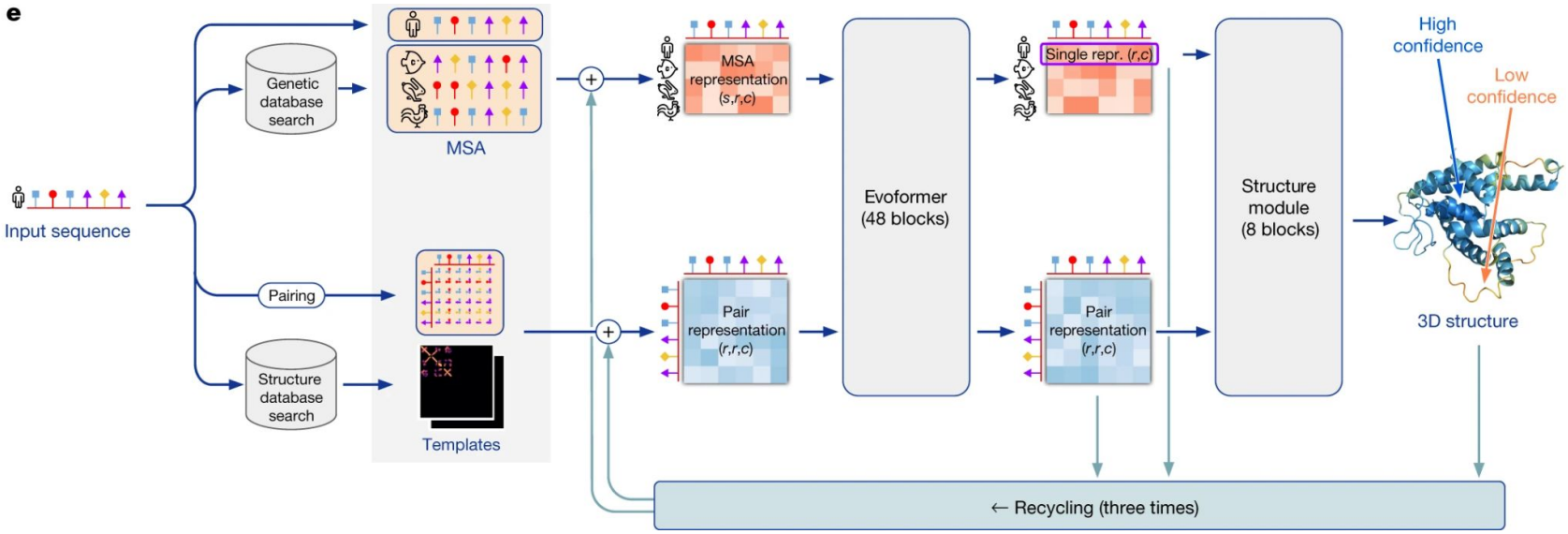
Why Predict Interactions?

- 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**.

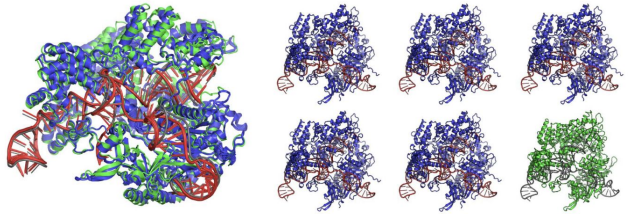
2. Structural Representation

- **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

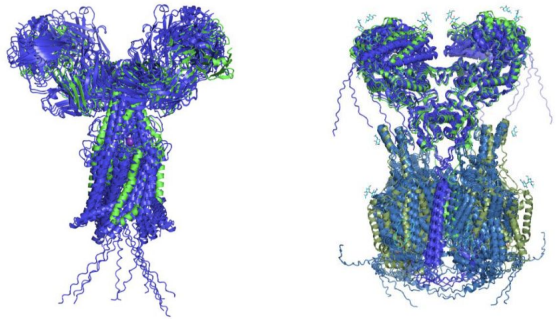
- **AF2:** Uses equivariant spatial transformations and post-relaxation for clash resolution.
- **AF3:** Employs a **generative diffusion model** (adds/removes Gaussian noise) with **minimal spatial inductive bias**. Post-processing/relaxation is rarely needed.

5. Architecture

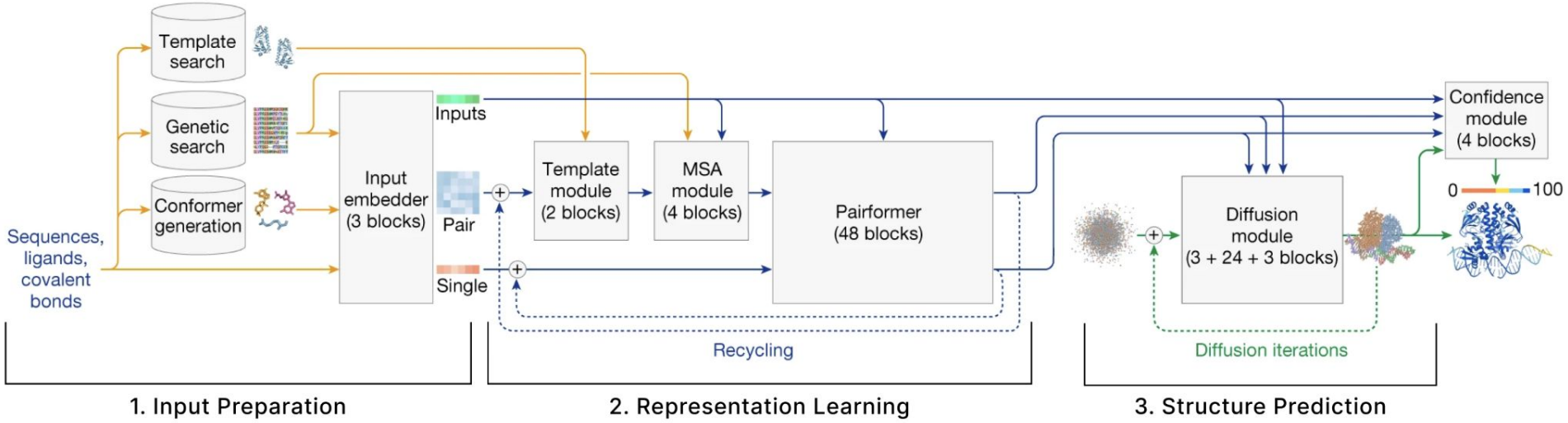
- **AF2:** Uses the **Evoformer** trunk, based on MSA and residue pair representations.
- **AF3:** Replaces Evoformer with **Pairformer**:
  - **No MSA stack** in the core, operates **only on token pairs**.
  - No outer product mean (no single-to-pair flow).

6. Activation and Data

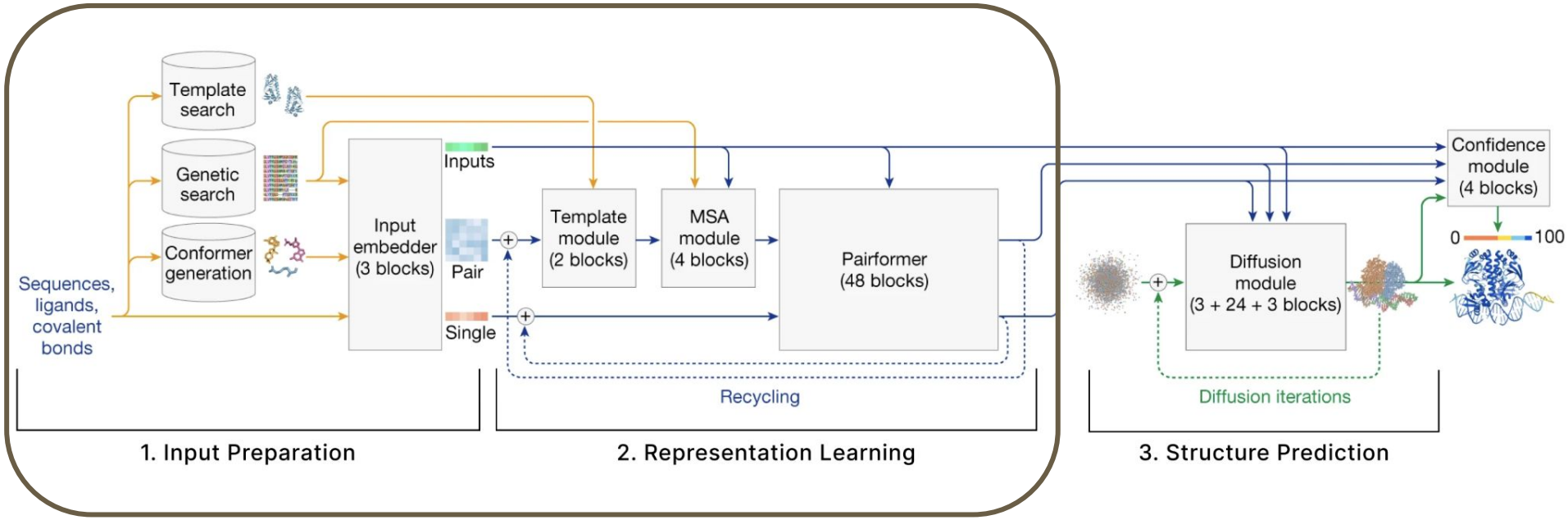
- **AF2:** Uses **ReLU** activations.
- **AF3:** Switches to **SwiGLU** in transition blocks (ReLU still used in atom attention).



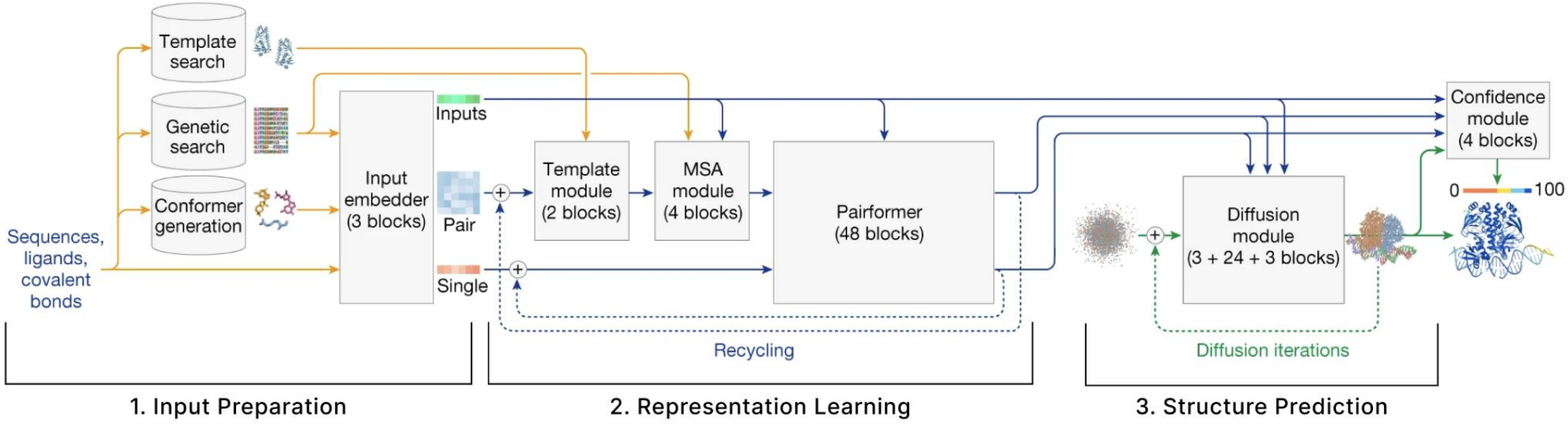
What We'll Cover in Part 1



What We'll Cover in Part 1



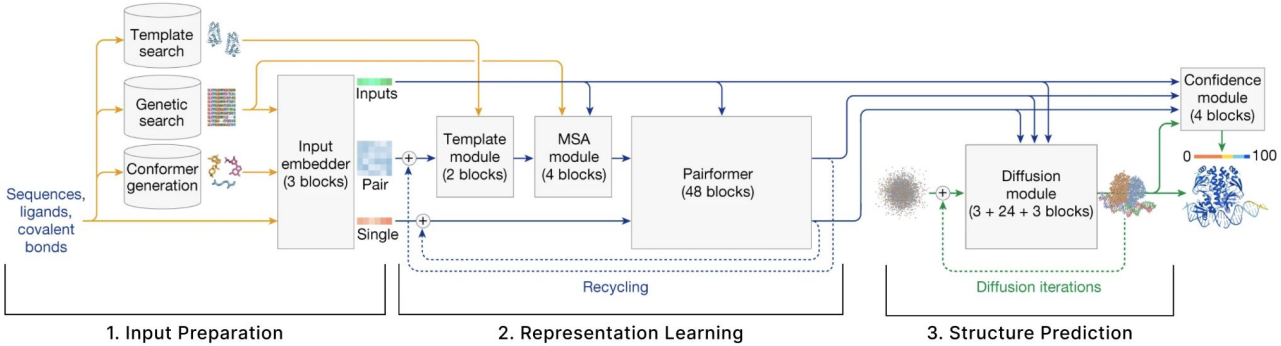
Architecture Overview



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

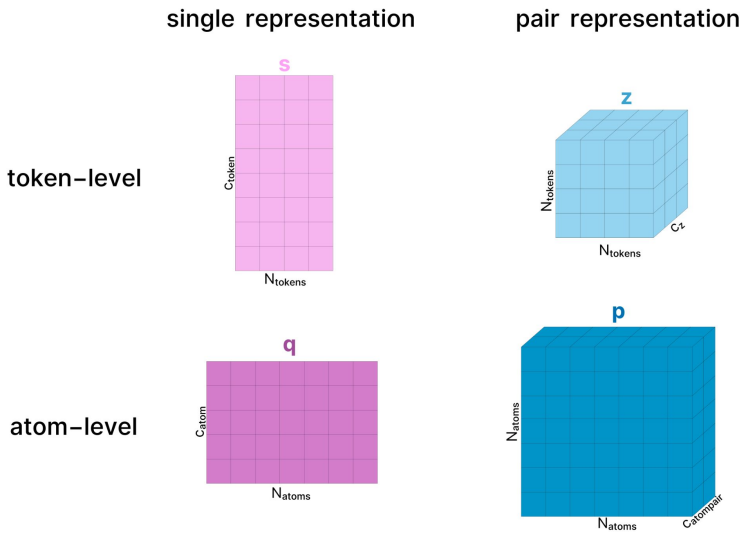
$\{\vec{\mathbf{x}}_l^{\text{pred}}\} = \text{SampleDiffusion}(\{\mathbf{f}^*\}, \{\mathbf{s}_i^{\text{inputs}}\}, \{\mathbf{s}_i\}, \{\mathbf{z}_{ij}\})$

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

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

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

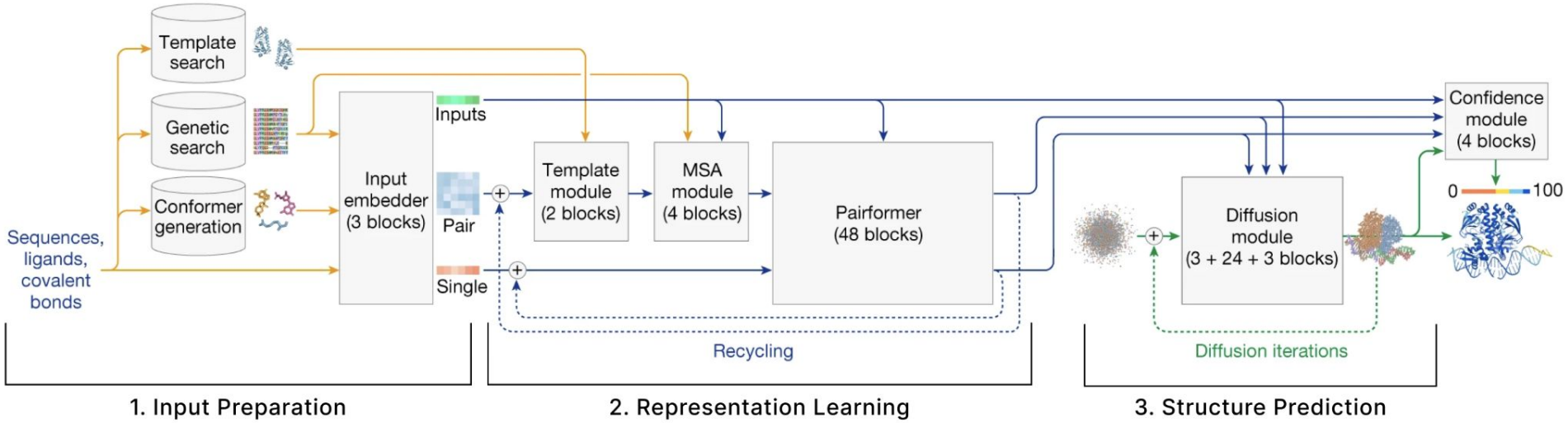
Notes on the variables and diagrams



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

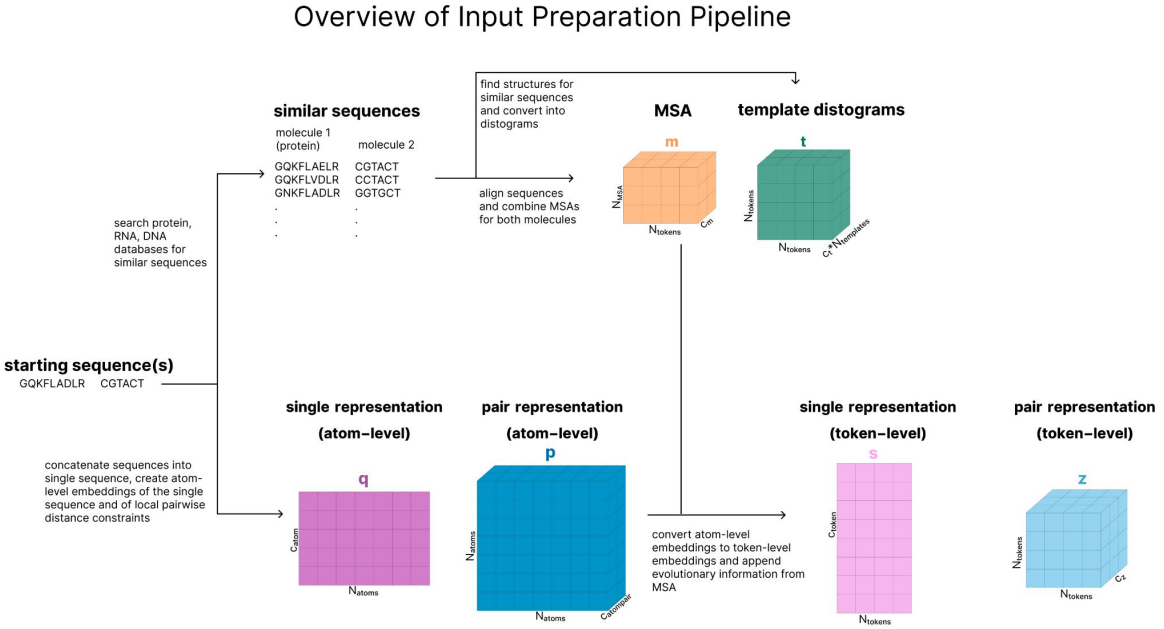


Input Features

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.

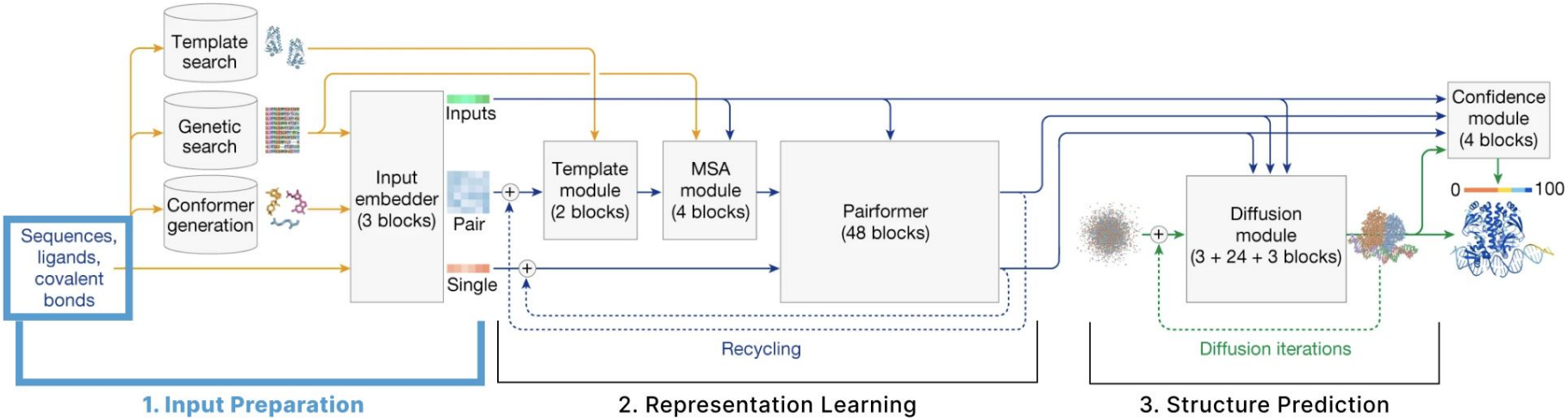
Input Preparation Pipeline



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

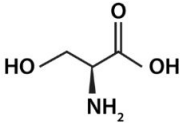
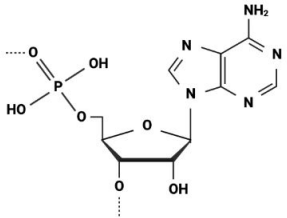
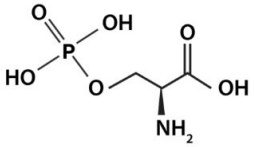
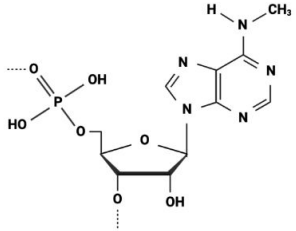
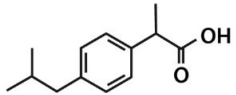
Retrieve similar sequences and structures to create MSA and templates

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

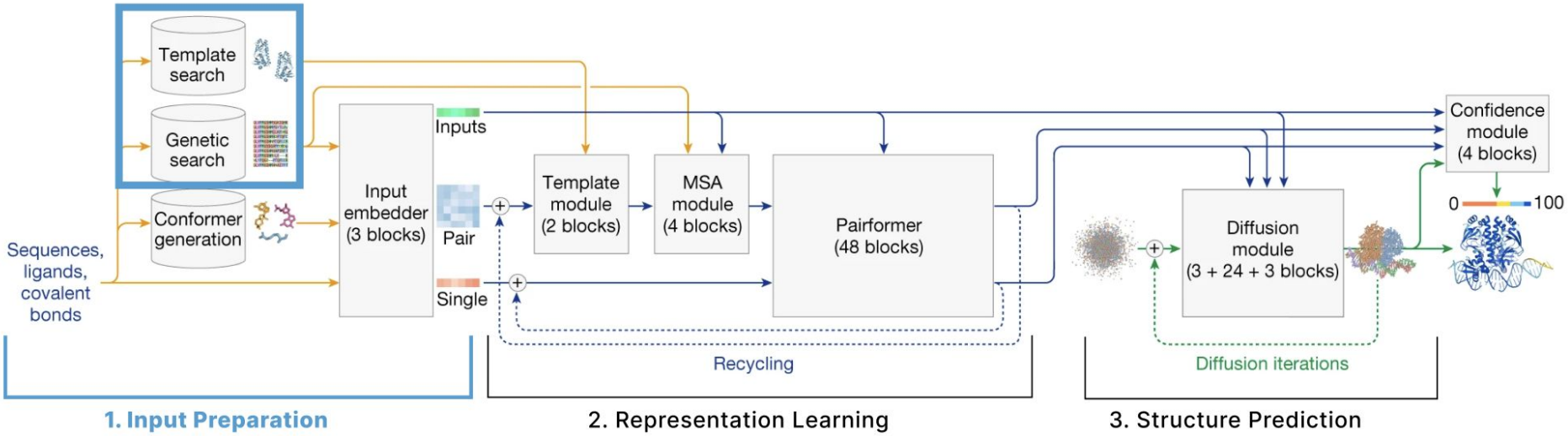
AF2 used one token per amino acid. AF3 extends this:

- **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)

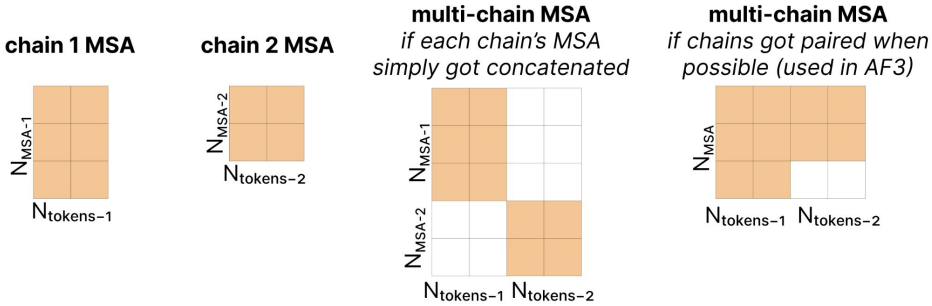
Aggregate atom-level representation to token-level

Retrieval (Create MSA and Templates)

Retrieval as Input (RAG-like Process)
AF3 begins by retrieving similar protein and RNA sequences (MSA) and related structures (templates). Unlike AF-Multimer, RNA sequences are also retrieved. While not traditionally called “retrieval,” this mirrors RAG in language models.

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.

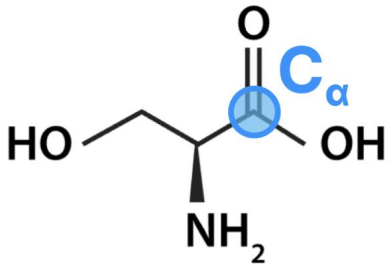


Retrieval (Create MSA and Templates)

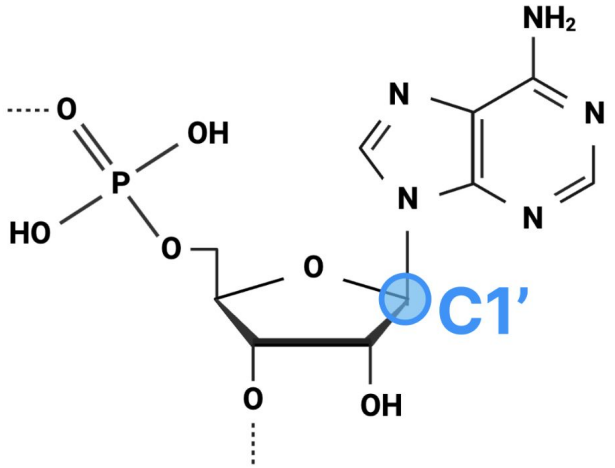
Template Representation

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.

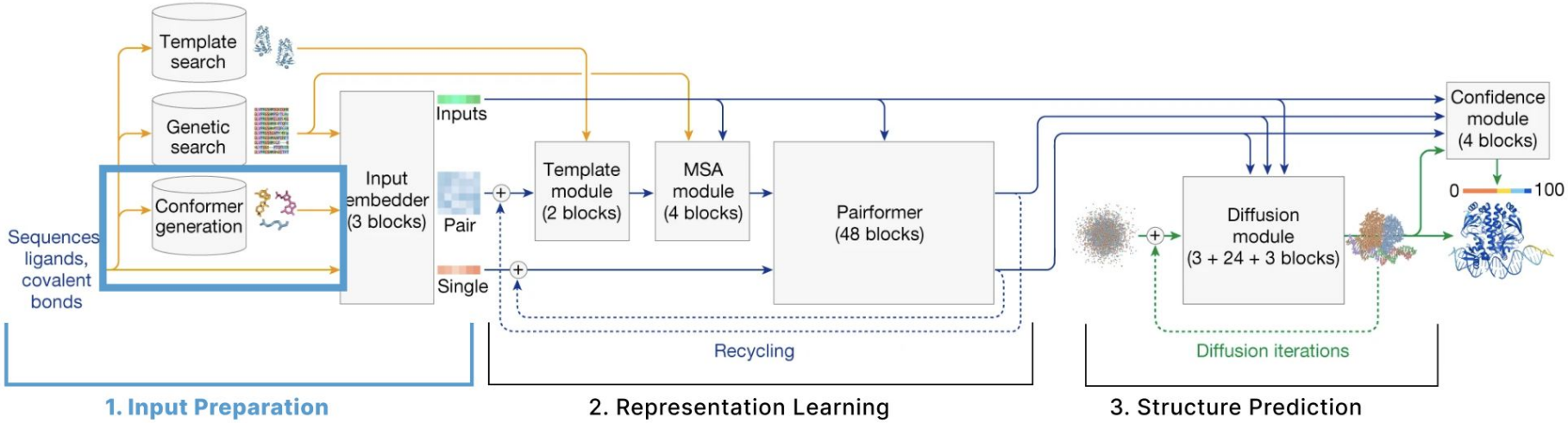
Standard Amino Acid



Standard Nucleotide



Create Atom-Level Representations



Tokenize input sequences

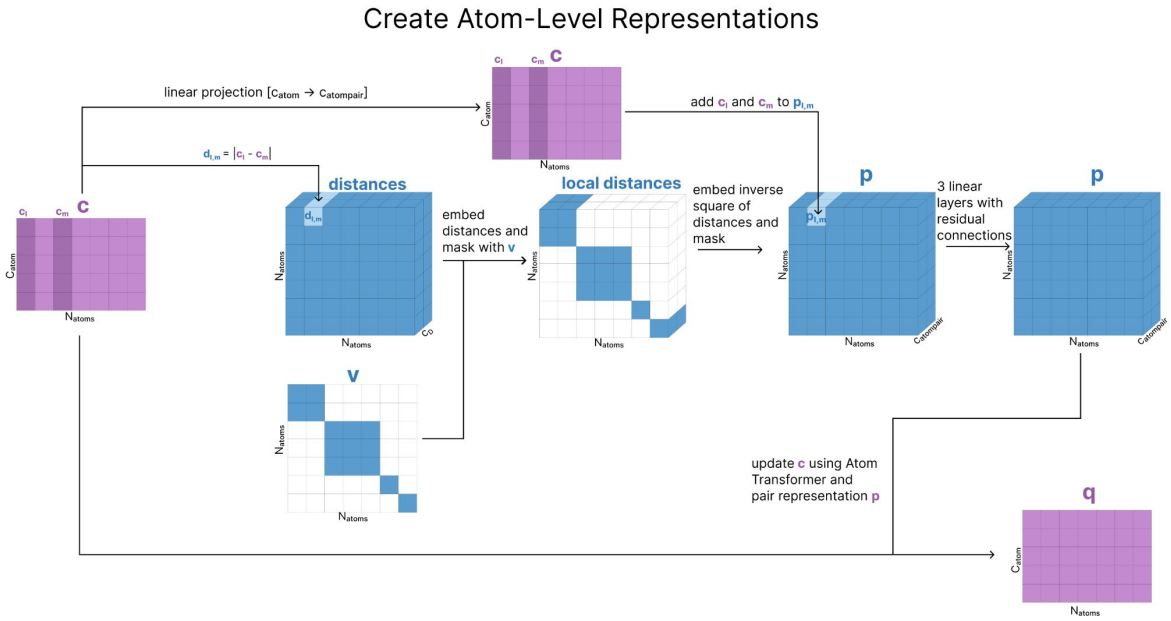
Retrieve similar sequences and structures to create MSA and templates

Create atom-level representation of sequences

Update atom-level representation (Atom Transformer)

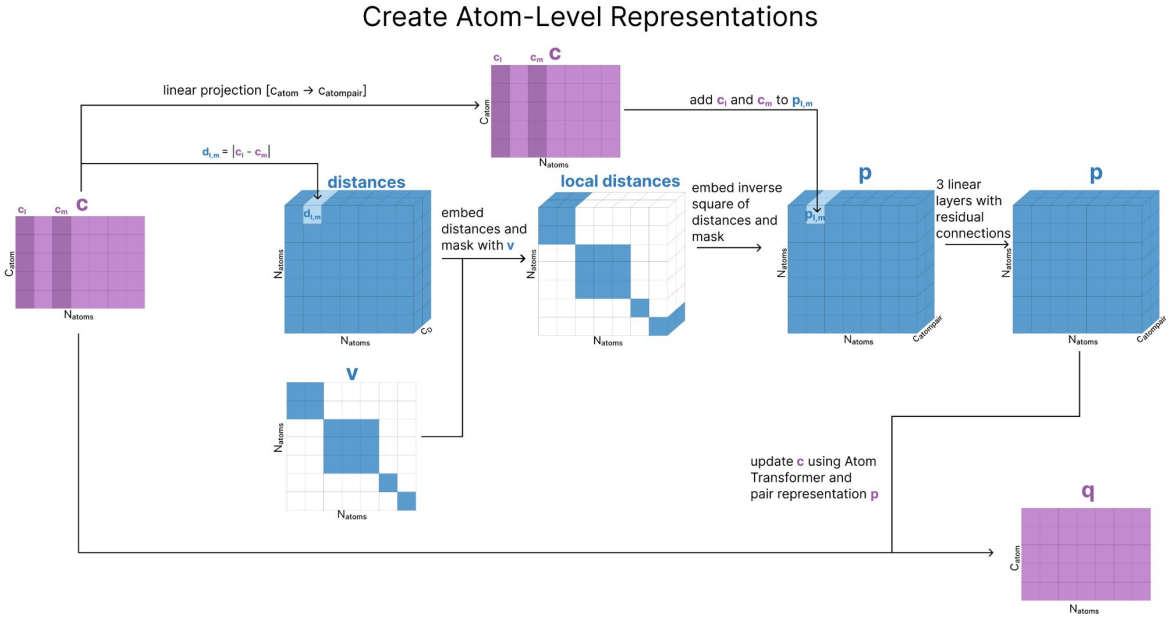
Aggregate atom-level representation to token-level

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

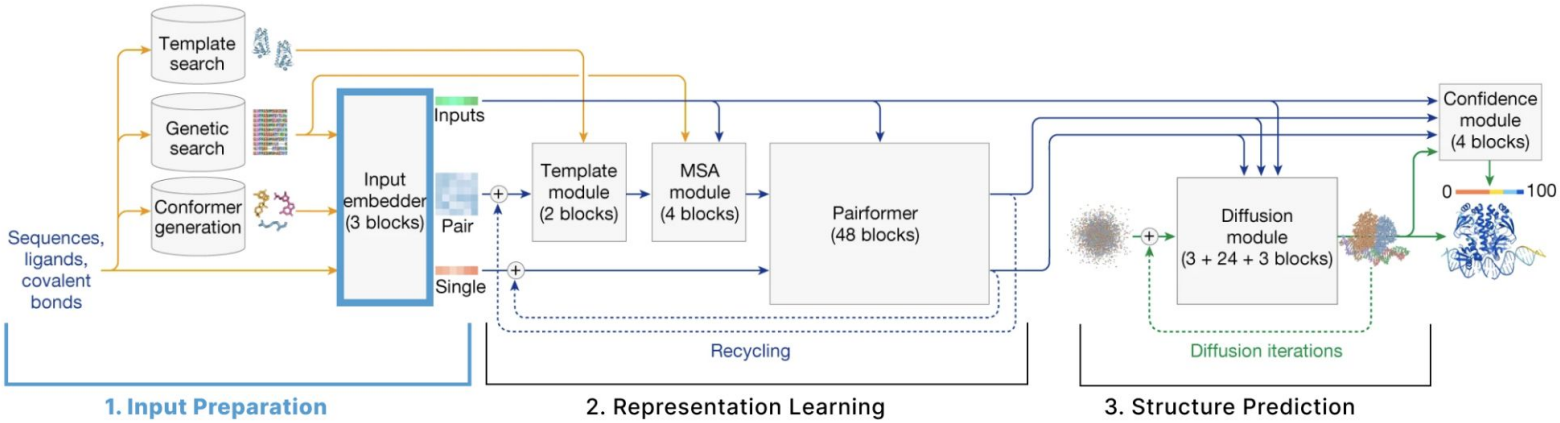
$\mathbf{a}_i = \underset{\substack{l \in \{1,\ldots,N_{\text{atoms}}\} \\ \text{tok\_idx}(l) = i}}{\text{mean}} (\text{relu}(\text{LinearNoBias}(\mathbf{q}_l)))$ $\quad \mathbf{a}_i \in \mathbb{R}^{c_{\text{token}}}$

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

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

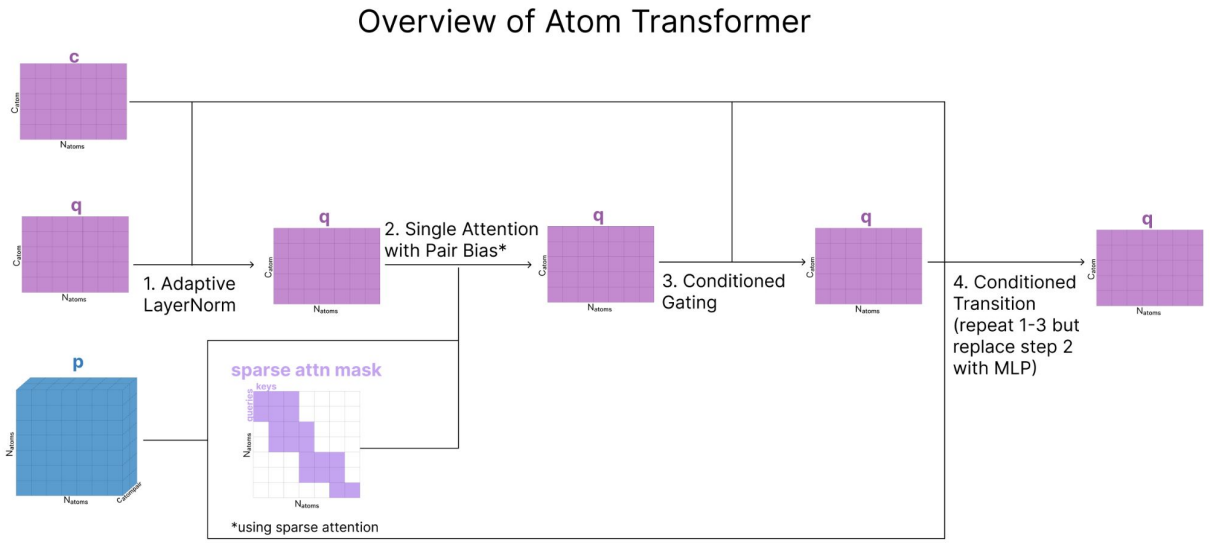
Update atom-level representation (Atom Transformer)

Aggregate atom-level representation to token-level

AtomTransformer introduces building blocks used elsewhere

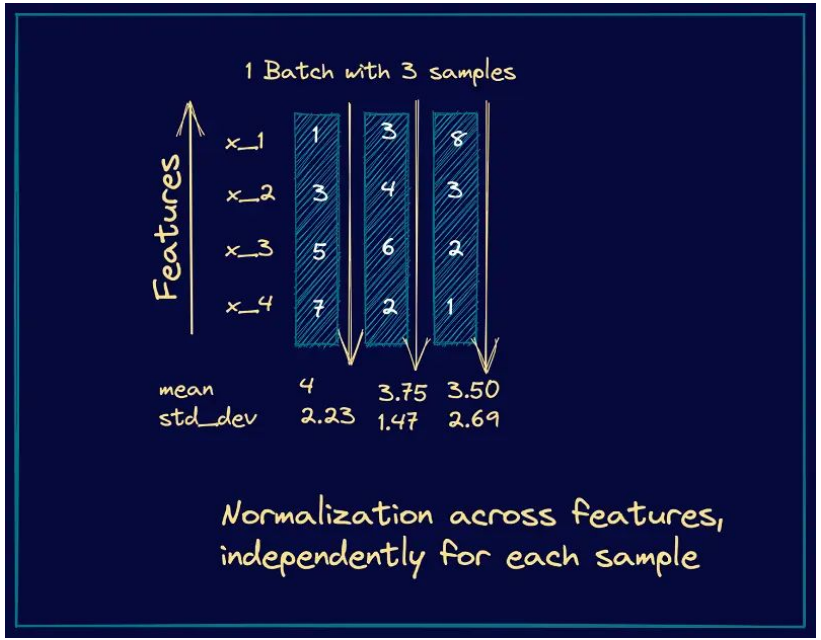
* **Adaptive LayerNorm**
* **Attention with Pair Bias**
* **Sequence-local atom attention**
* **Conditioned Gating**
* **Conditioned Transition blocks**

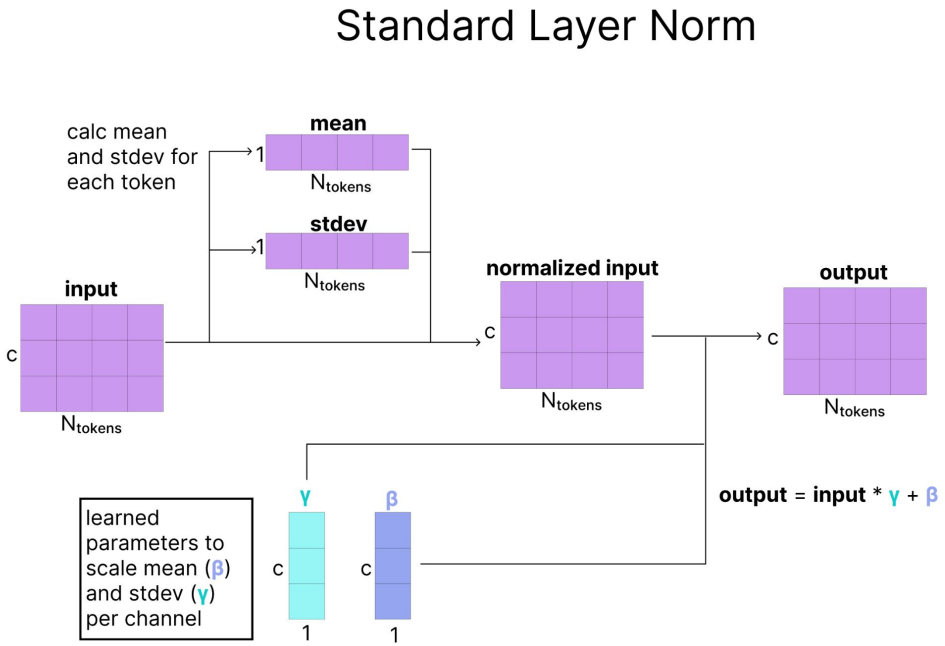
Create Atom-Level Representations



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

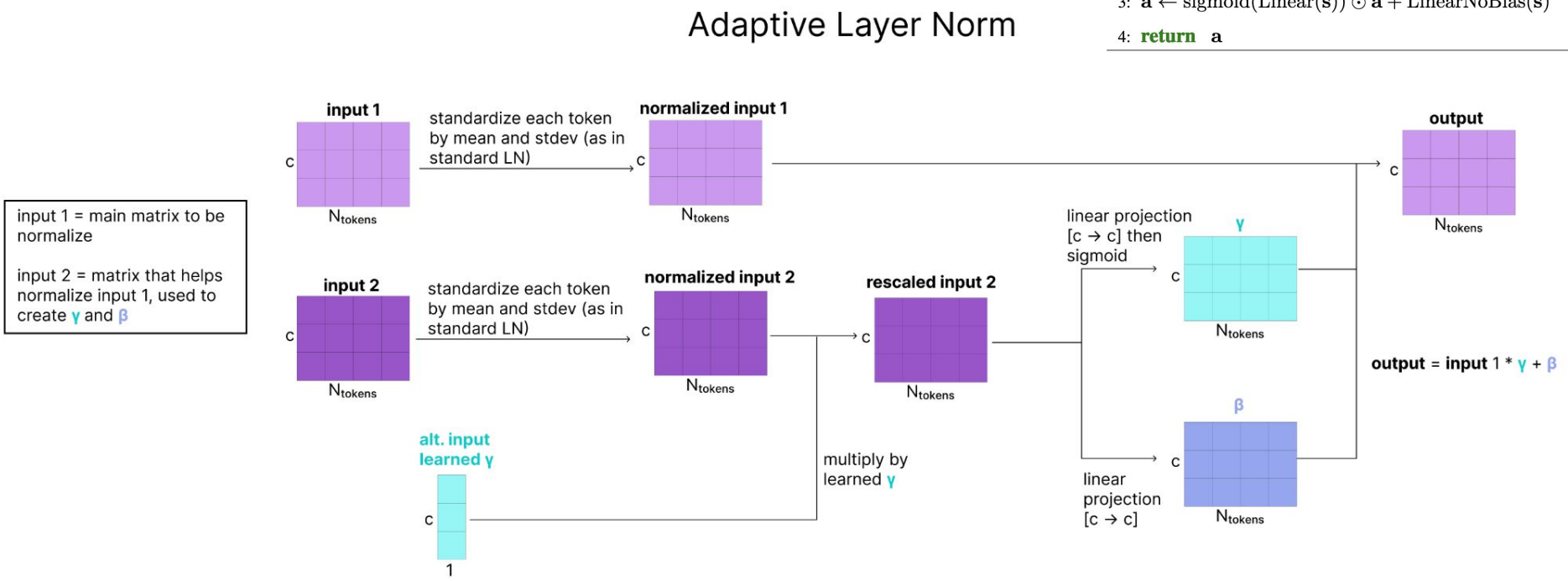
def $\text{AdaLN}(\mathbf{a}, \mathbf{s})$:

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** $\mathbf{a}$

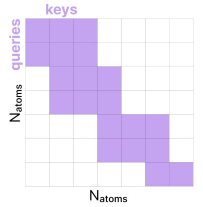


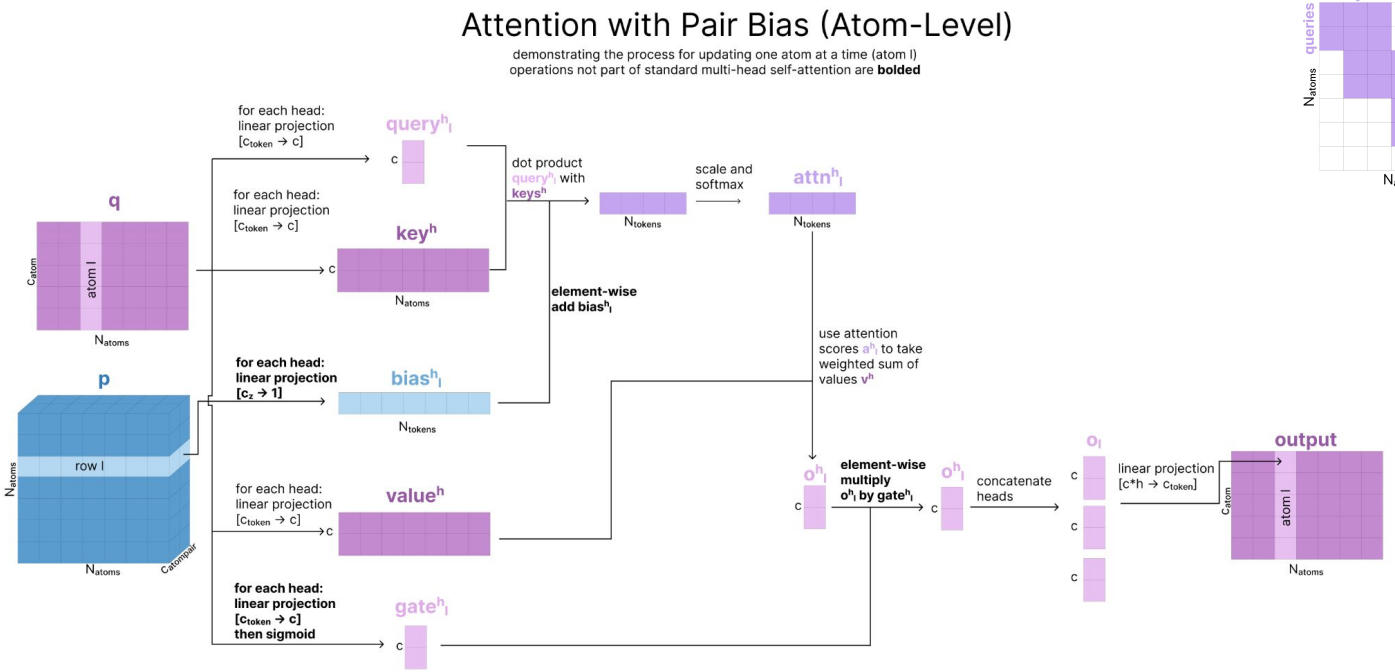
Instead of fixed scaling (gamma) and bias (beta), AdaNorm adaptively predicts these parameters from **c** to normalize **q**.

Atom Transformer - 2. Attention with Pair Bias & Gating

Sequence-local atom attention

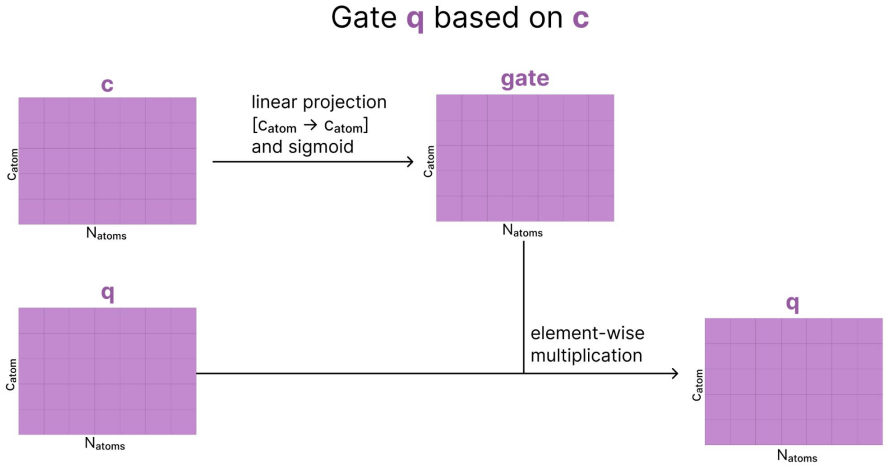
Self-attention run within rectangular blocks where 32 query atoms can attend to 128 nearby key atoms

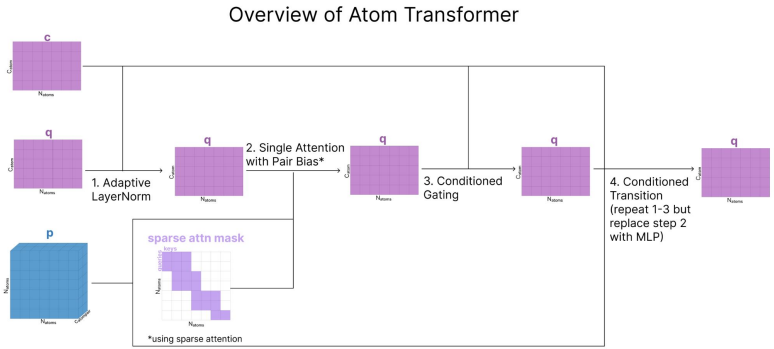




- Attention uses queries, keys, and values from **q** with added pairwise biases from **p**.
- A gating sigmoid filters attention output, controlling information flow like LSTM gates.
- Sparse local attention attends over groups of 32 atoms at a time can all attend to 128 other atoms for efficiency.

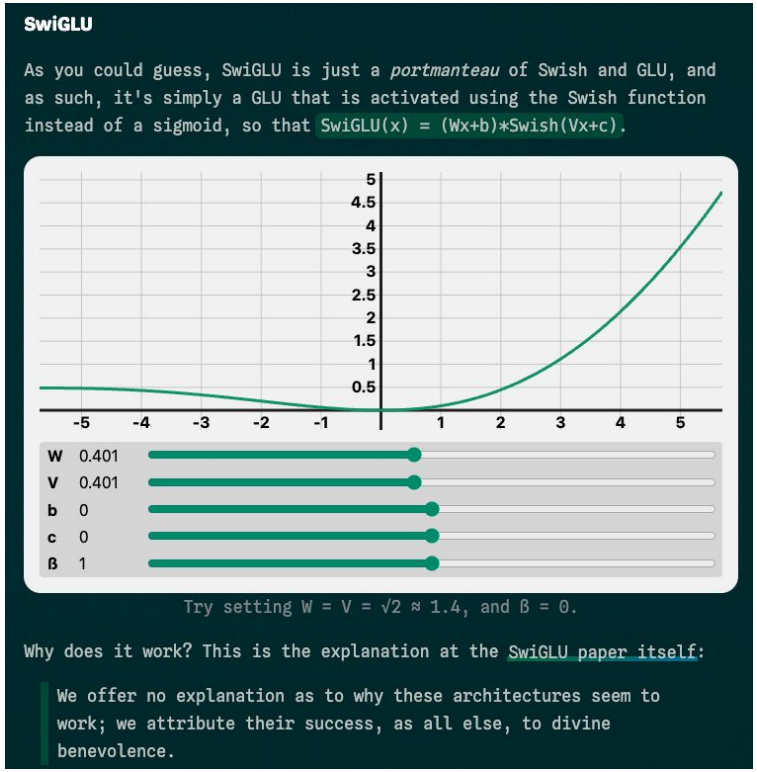
Atom Transformer - 3. Conditioned Gating

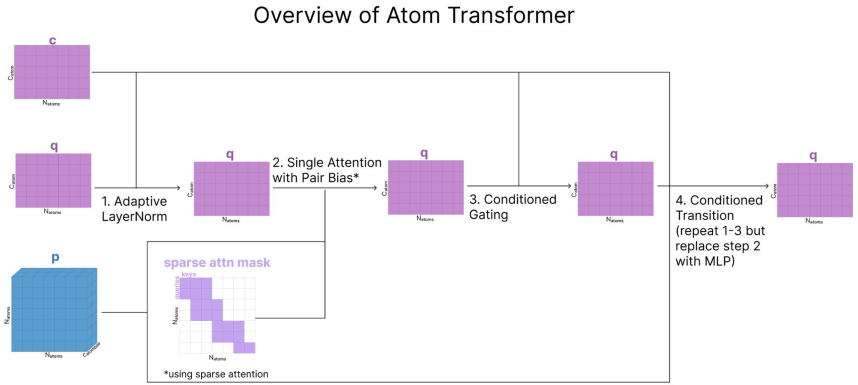




A second gating step is applied, conditioned on **c**, to selectively modulate the representation.

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

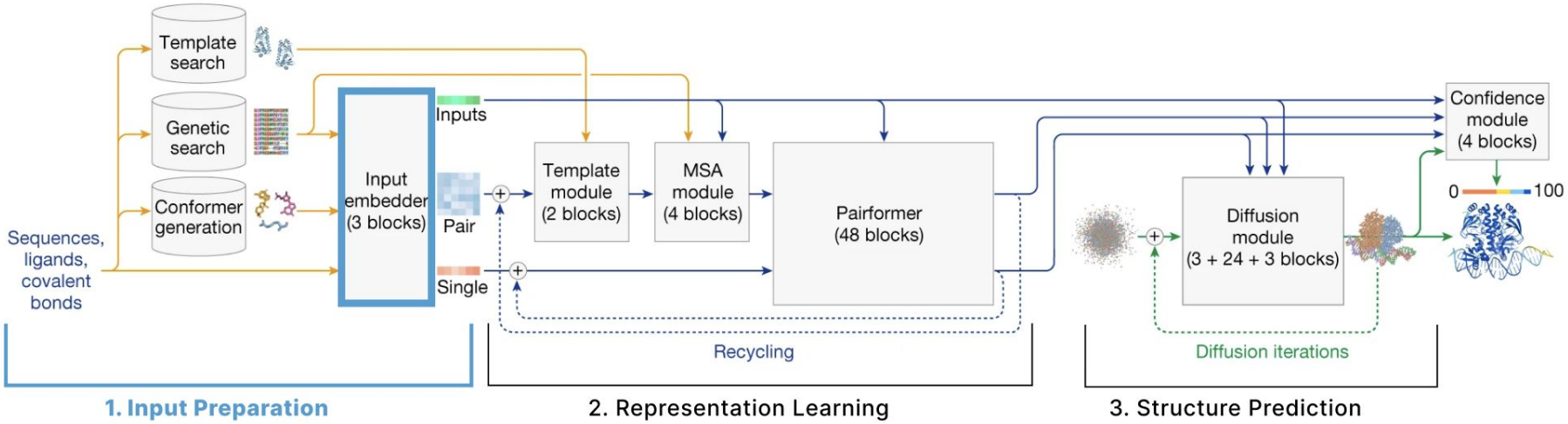




MLP layers are “conditioned” by AdaNorm and gating based on **c**. AF3 replaces ReLU with SwiGLU for improved non-linearity.

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

Aggregate Atom-Level → Token-Level



Tokenize input sequences

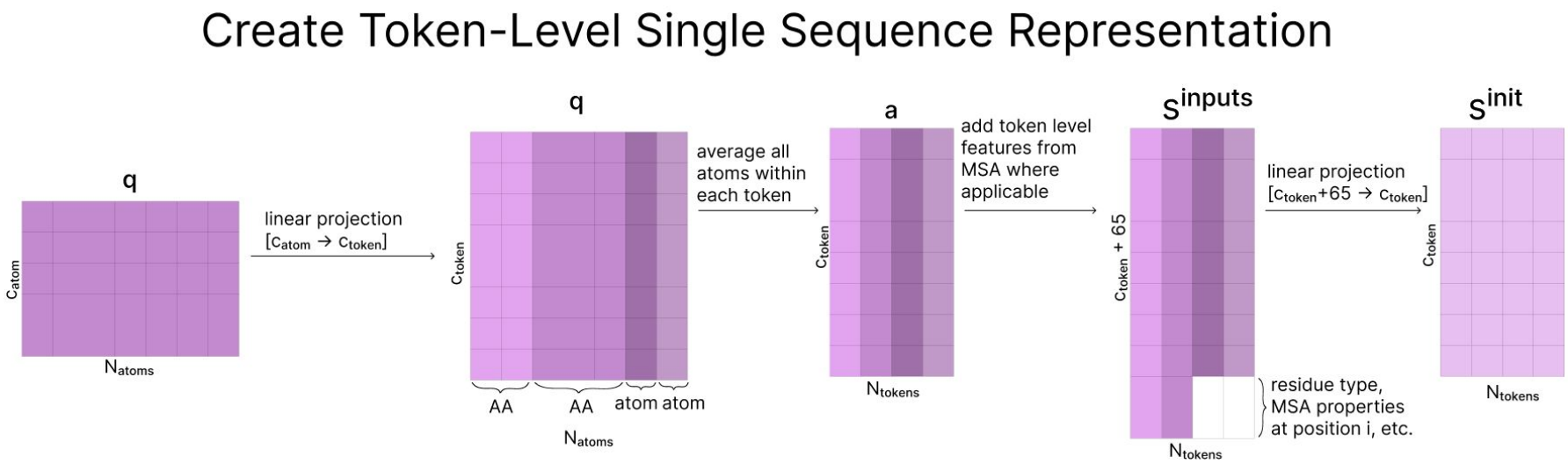
Retrieve similar sequences and structures to create MSA and templates

Create atom-level representation of sequences

Update atom-level representation (Atom Transformer)

Aggregate atom-level representation to token-level

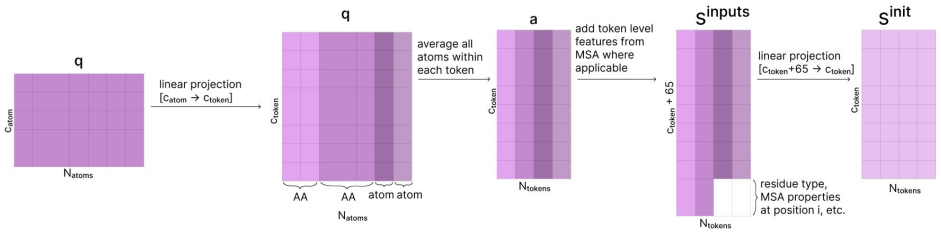
Aggregate Atom-Level → Token-Level



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

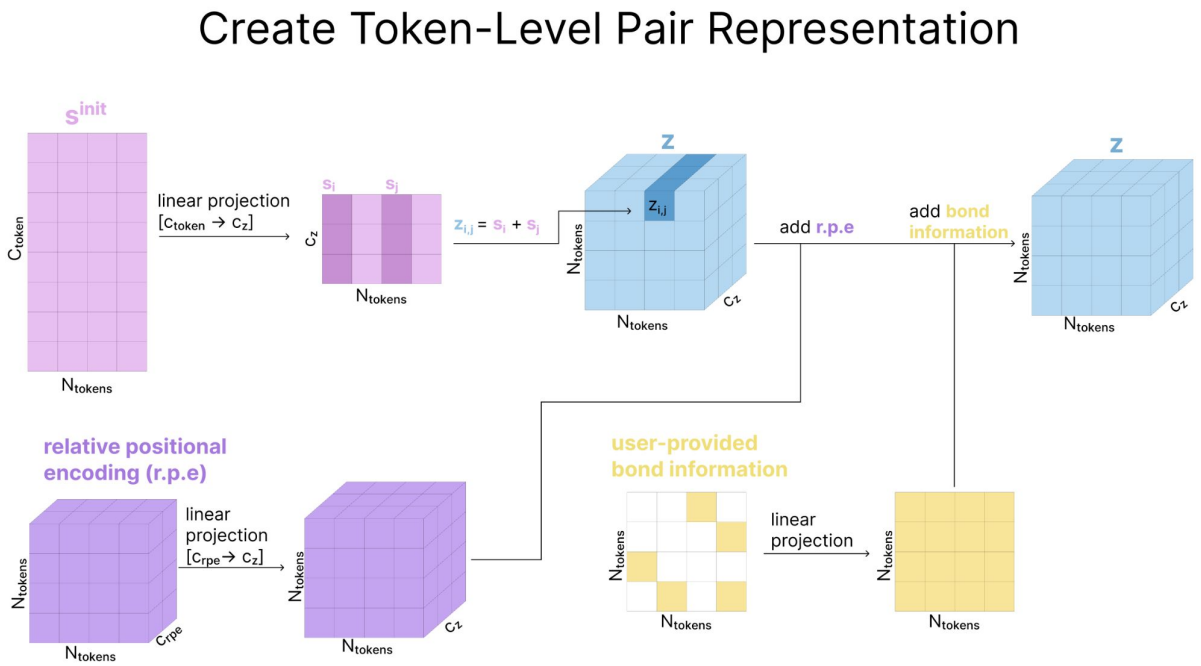
1: $\{\mathbf{s}_i^{\text{inputs}}\} = \text{InputFeatureEmbedder}(\{\mathbf{f}^*\})$

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.

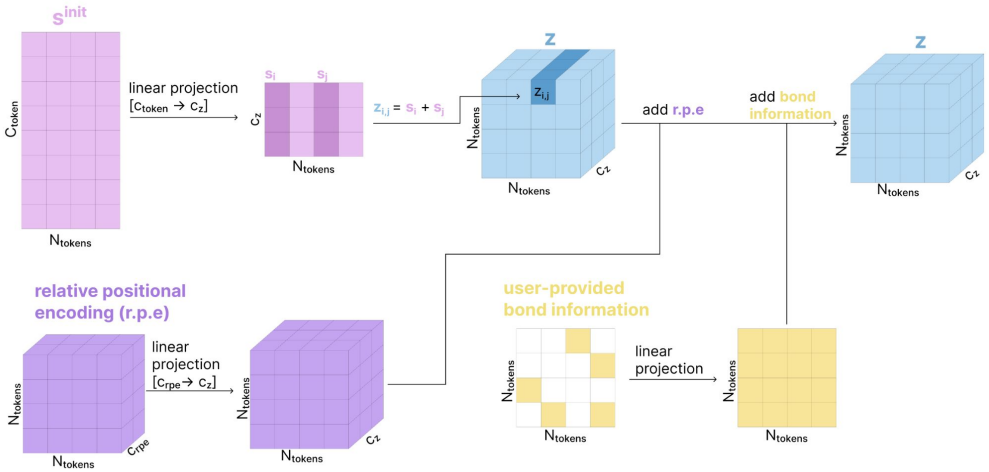
Aggregate Atom-Level → Token-Level



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

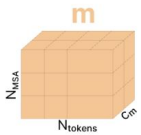
5: $\mathbf{z}_{ij}^{\text{init}} \mathrel{+}= \text{LinearNoBias}(\mathbf{f}_{ij}^{\text{token\_bonds}})$

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.

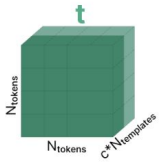
Aggregate Atom-Level → Token-LeveL

Information about related sequences and their structures

Multiple Sequence Alignment

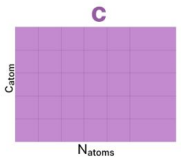


Structure Templates

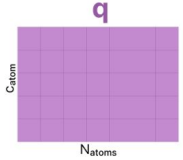


Information about all the atoms (“single”)

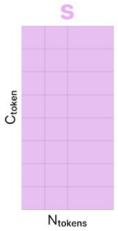
Original Atom-Level Single Representation



Updated Atom-Level Single Representation

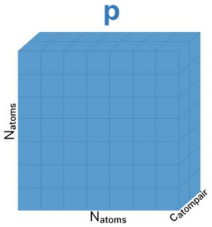


Token-Level Single Representation

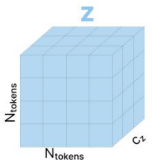


Information about all the pairs of atoms (“pair”)

Atom-Level Pair Representation

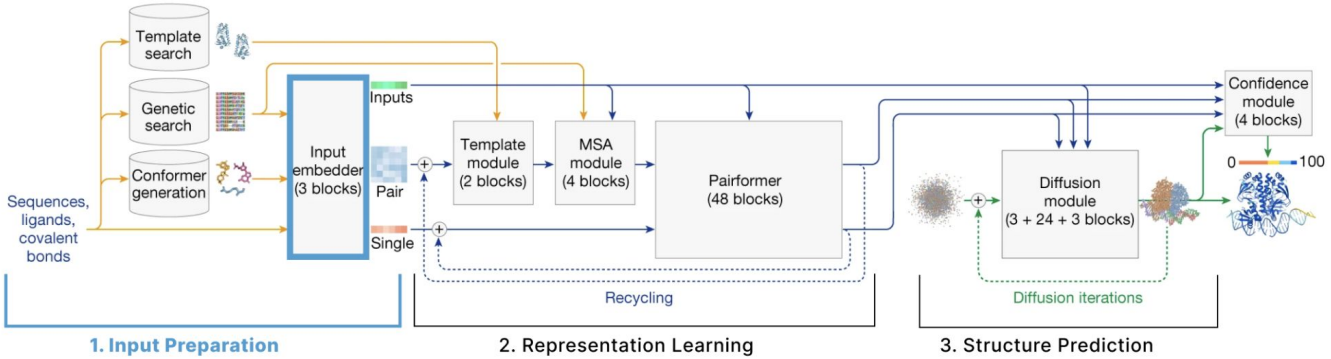


Token-Level Pair Representation



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**

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

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

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

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

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

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

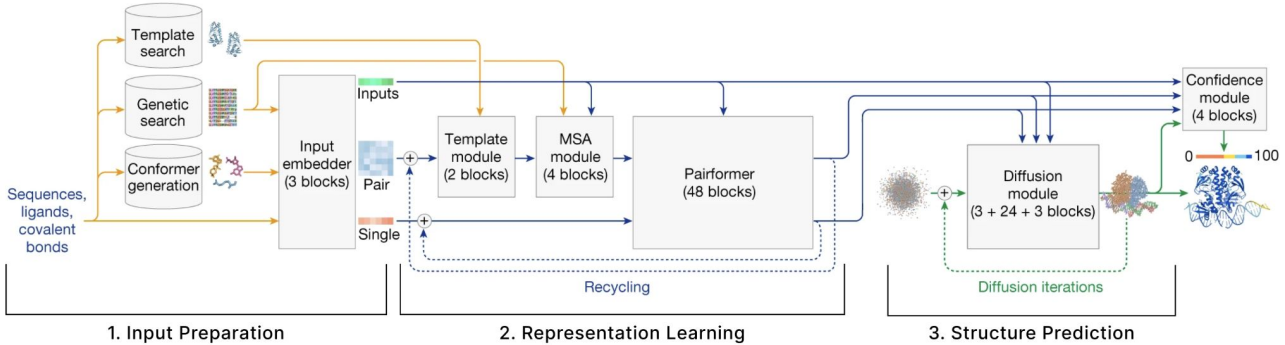
end for

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

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

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

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



Resource

- https://elanapearl.github.io/blog/2024/the-illustrated-alphafold/
- https://medium.com/@falk_hoffmann/alphafold3-and-its-improvements-in-comparison-to-alphafold2-96815ffbb044
- https://310.ai/blog/alphafold2-alphafold-multimer-alphafold3
- https://towardsdatascience.com/sparks-of-chemical-intuition-and-gross-limitations-in-alphafold-3-8487ba4dfb53/
- https://www.ai4pharm.info/alphafold3