Nuclear Modelling using AI from laptops to eXascale (MATRIX):

1) **Programming CFD/Nuclear Models using AI software** which enables interoperability between GPUs, CPUs and AI computers (energy efficient exascale Cerebras CS2 ~1M node chip) & exploitation of community AI software;

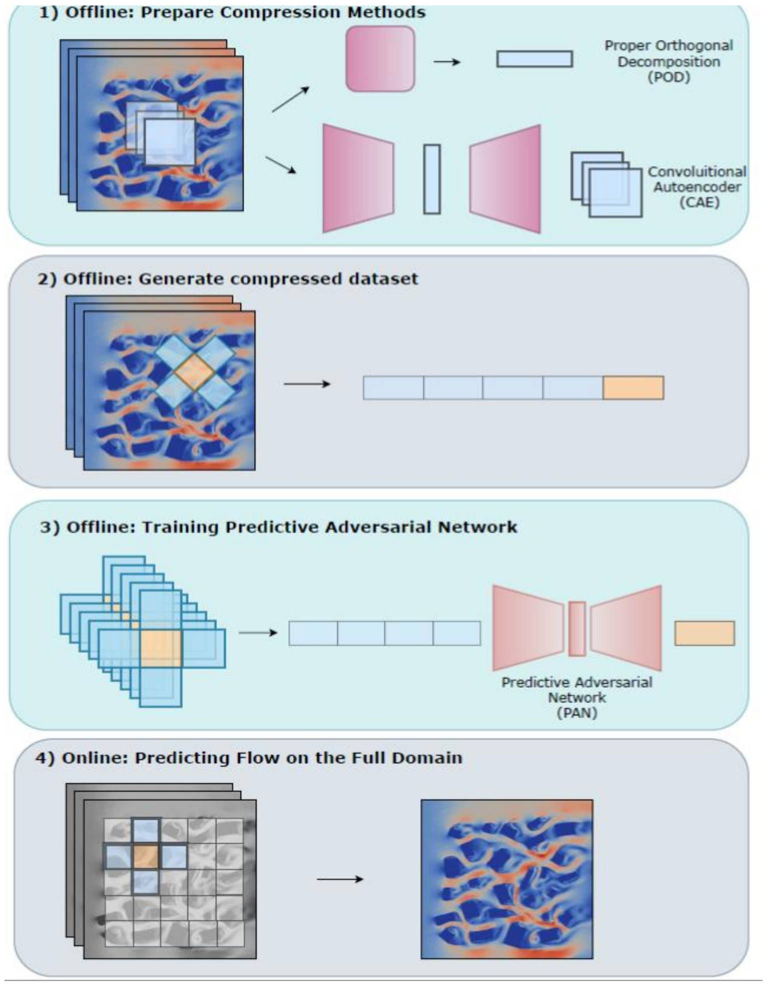
2) **AI-Physics modelling** to reduce the number of unknowns solved for which can be important for some computer architectures as well as speeding up models;

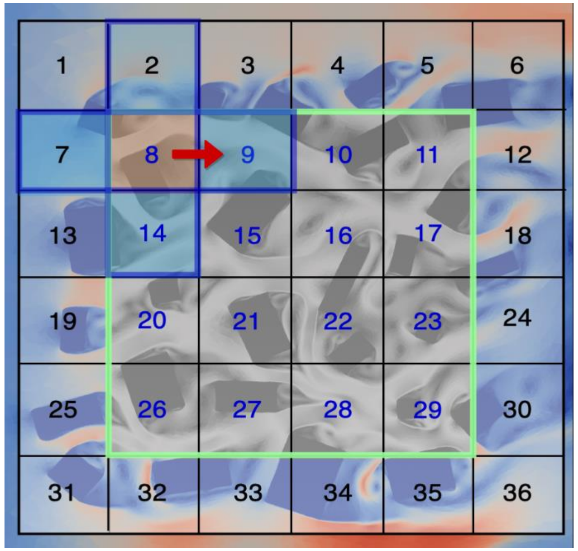
3) **Hybrisation of AI Physics modelling and discretization** of the differential equations in order to construct; new sub-grid-scale (SGS) models, methods that force the equation residuals to zero (RDEIM) or physics informed methods - it's now accepted that future SGS methods will increasing be based on AI and they may need implicit coupling to CFD using AI programming (1 above);

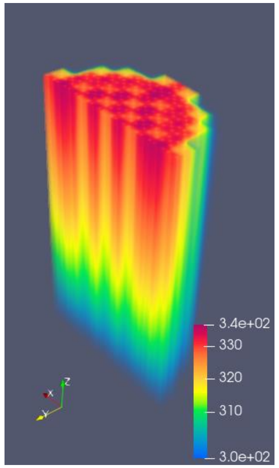
4) **Digital twins** that are able to assimilate data, perform uncertainty quantification and optimization using the optimization engine embedded in all AI software.

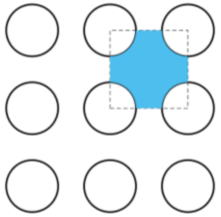
Building a detailed flow model of using AI modelling

– bottom middle grid architecture of an AI computer e.g. Cerebras CS-2 with 800,000 nodes on a chip

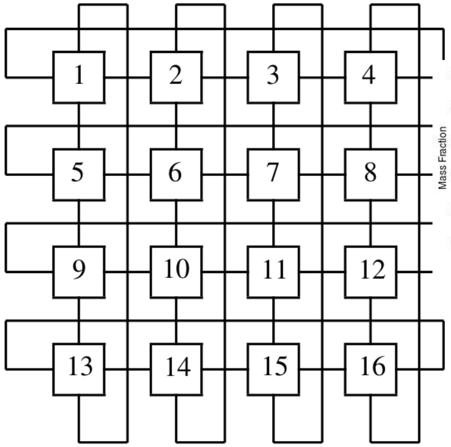

1) Offline: Prepare Compression Methods
Proper Orthogonal Decomposition (POD)
Convoluitional Autoencoder (CAE)
2) Offline: Generate compressed dataset
3) Offline: Training Predictive Adversarial Network
Predictive Adversarial Network (PAN)
4) Online: Predicting Flow on the Full Domain

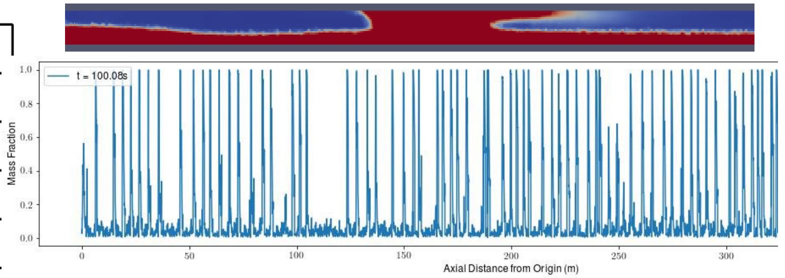






Left above: Automatic code generation DEVITO to form eqns – BWP temperature. Right subdomain.

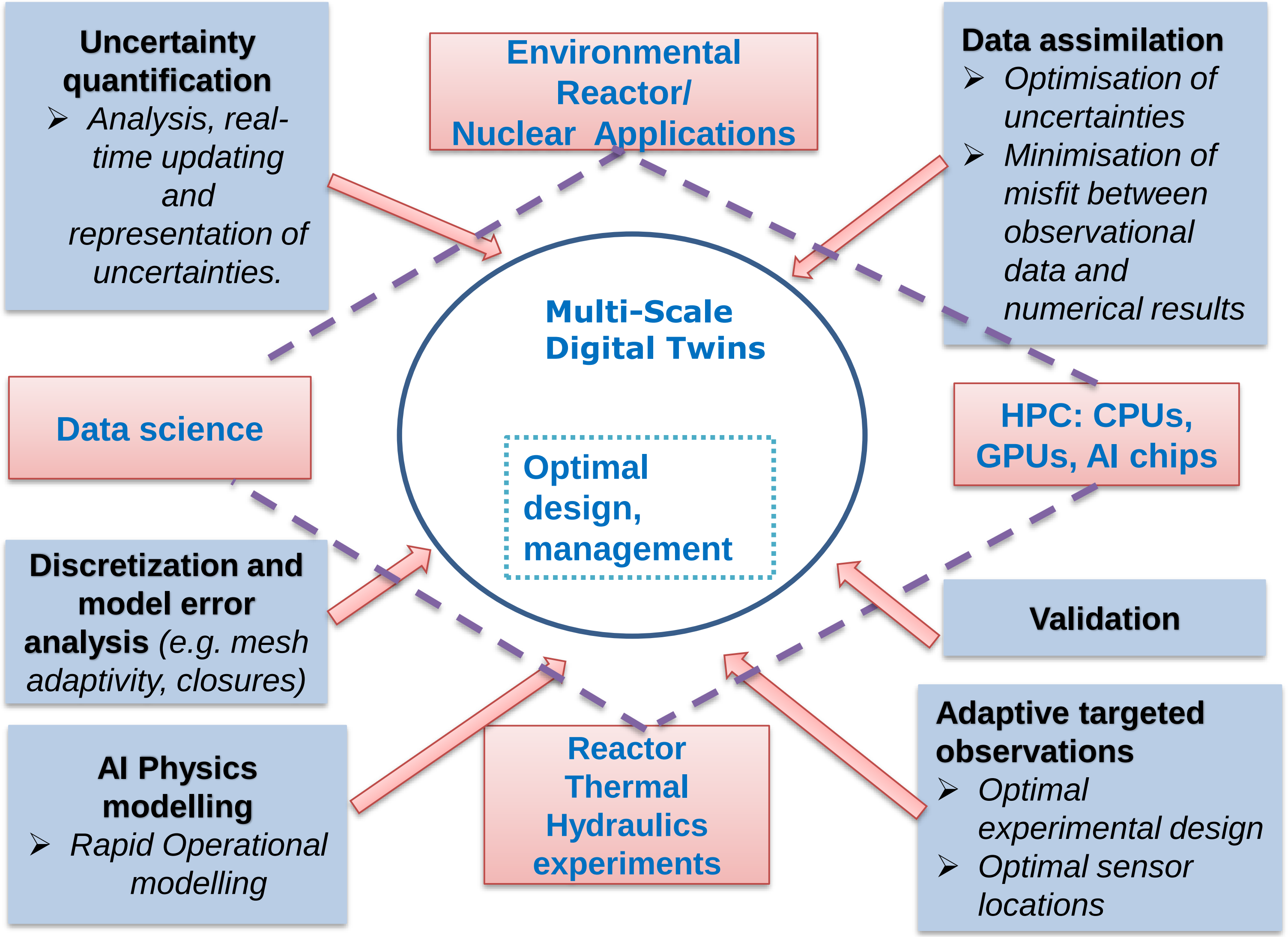




AI model of multiphase slug flow in 1000m pipe from subdomains. Top: cross section of liquid in subdomain. Bottom: liquid along pipe fraction

Nuclear Modelling using AI at eXascale (MATRIX):

- AI-Physics modelling
- AI SGS CFD/TH/RT modelling
- Hierarchical modelling – from CFD/RT to AI Models - multiscale
- AI computers e.g. ~1M node Cerebras chip
- Digital Twins (Right) taking advantage of AI optimization
- Code in AI e.g. PyTorch, SymPy
- Summary speed: AI models 10^3-10^6 faster, AI computer 10^3 faster, Total 10^6-10^9 faster



Uncertainty quantification
- *Analysis, real-time updating and representation of uncertainties.*

Environmental Reactor/ Nuclear Applications

Data assimilation
- *Optimisation of uncertainties*
- *Minimisation of misfit between observational data and numerical results*

Multi-Scale Digital Twins

Optimal design, management

Data science

HPC: CPUs, GPUs, AI chips

Discretization and model error analysis *(e.g. mesh adaptivity, closures)*

Validation

AI Physics modelling
- *Rapid Operational modelling*

Reactor Thermal Hydraulics experiments

Adaptive targeted observations
- *Optimal experimental design*
- *Optimal sensor locations*

Advantages of programming CFD/Neutronics/Solids in AI:

1) **Solvers:** Linear, non-linear multigrid methods using convolutional auto-encoders, Newton Raphson iteration. AI models can also be used as preconditions in CFD.
2) **Hierarchical solvers:** From CFD to fine and coarse AI-Physics models and one catch switch between these locally or globally.
3) **Embedded CFD and AI Physics modelling** in a single neural network – SGS modelling, PINN, RDEIM.
4) **Automatic Adjoint Generation and Digital twins** that are able to assimilate data, perform uncertainty quantification and optimization using the optimization engine embedded in all AI software.
5) **Realize new features in modelling:** reduced arithmetic speed; coupling different physics, finance, economic, language and/or social models; leverage AI code.
6) **More accessible AI programming for modelers** - similar to AI applications.
7) **Optimized code interoperability between computer architectures:** CPU, GPU and AI chips - exascale.
8) **Long term model/code sustainability.**

Strategy Issues for CFD implementation using AI software:

1) **Multi-grid solvers – through CNN autoencoder (done)**

2) **Adjoints – through weights and activation functions (done)**

3) **Unstructured meshes e.g. Space Filling Curves CNN, MeshCNN (done)**

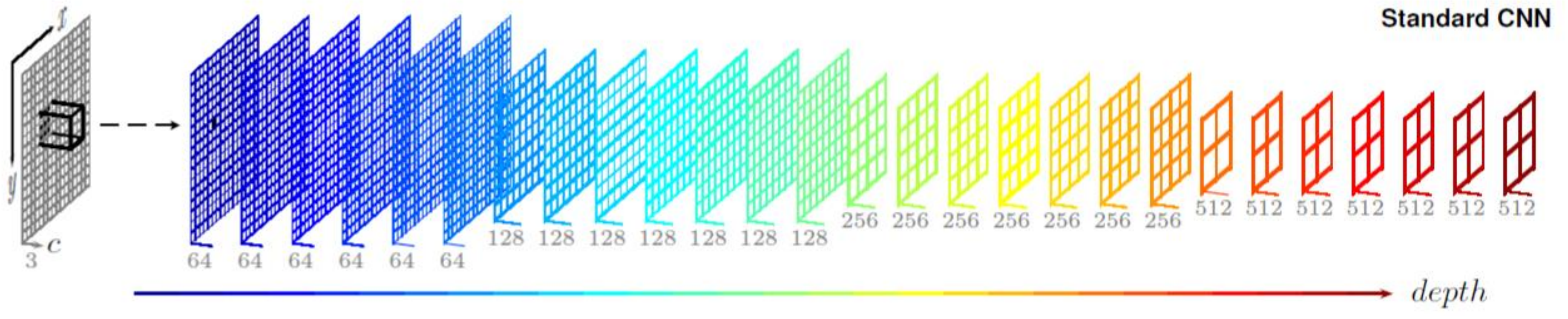
4) **Defining discretization e.g. DEVITO, OpenSBLI, SymPy, Firedrack**

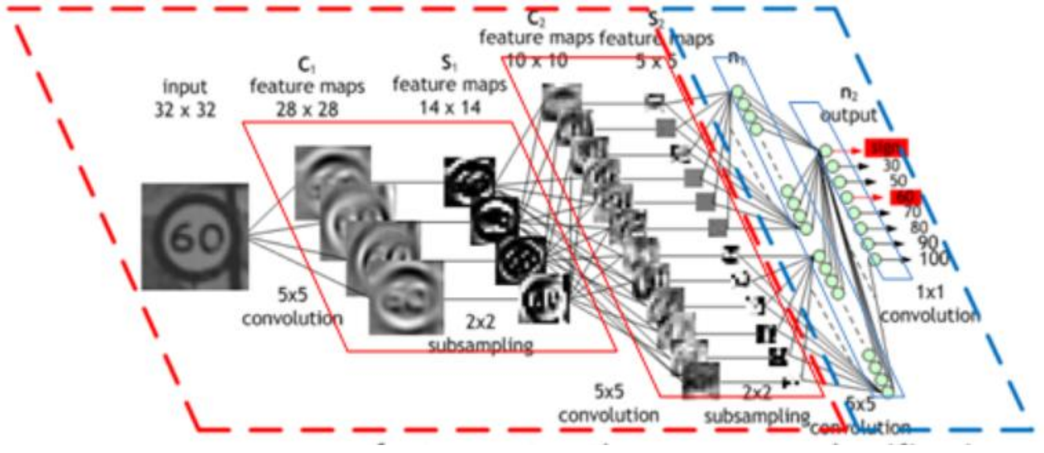
5) **Parallel updating of halos e.g. SciML, Lightning-Horovod, DeepSpeed, transformer methods (dense), MPI may be used for block unstructured**

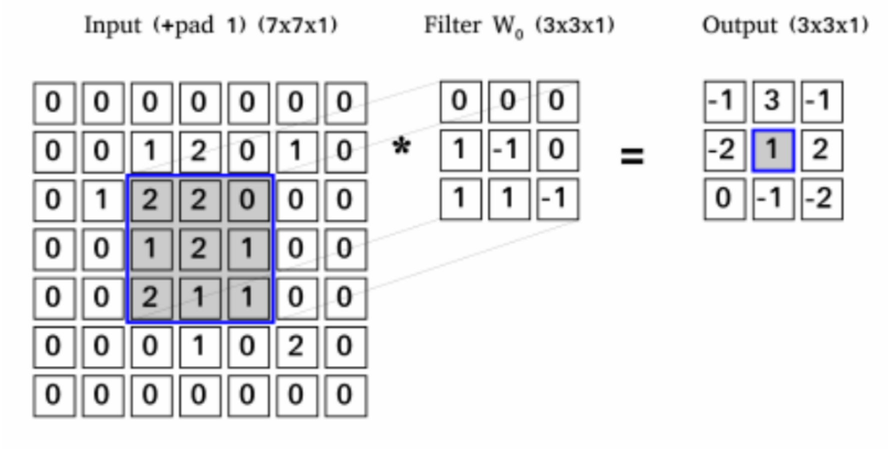
6) **Large matrix/solution free methods e.g. radiation transport – use single CNN filter for semi-structured discretization in each block (done)**

7) **GUIs linkage – keep flexible**

Convolutional ANNs with linear activation functions and weights from discretization stencil can be Multi-Grid Solvers:

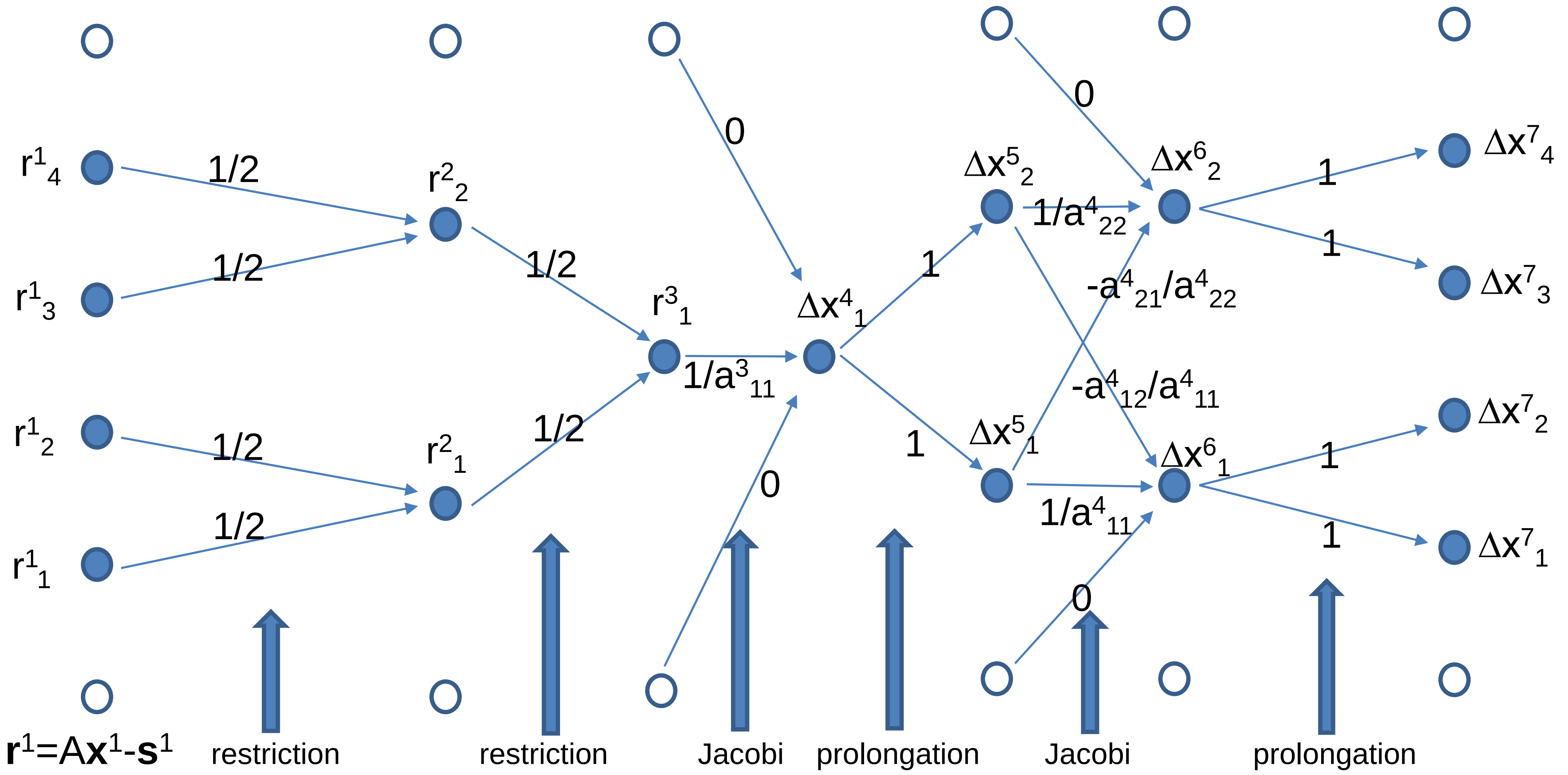






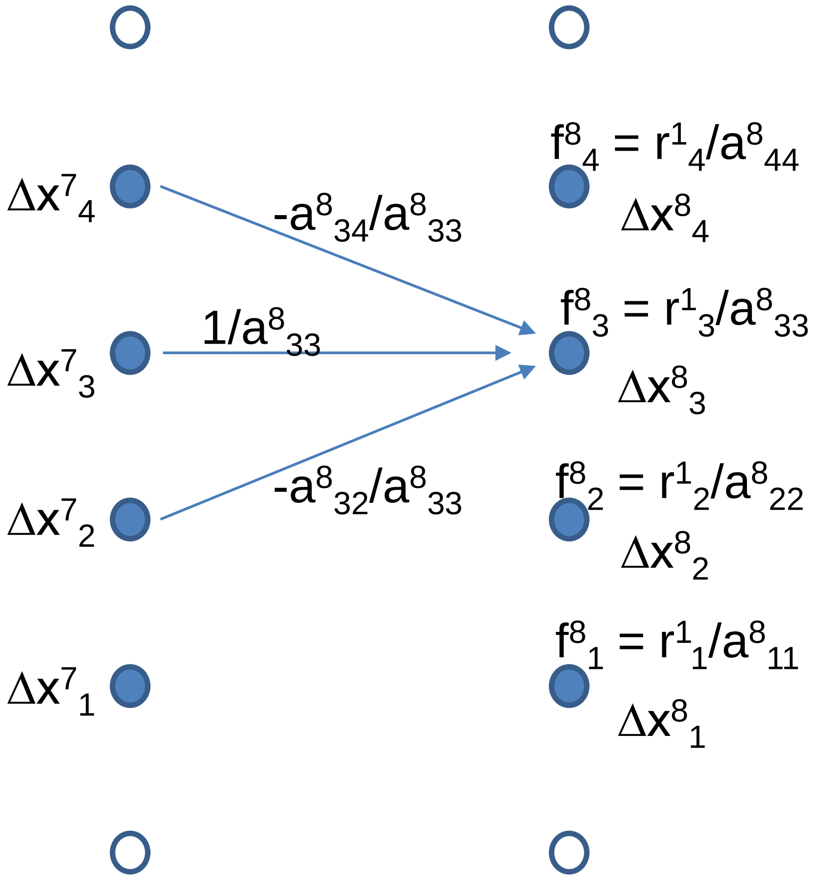
Solve Ax=s using Simplest F-cycle Multi-Grid Method and Convolutional ANN in 1D – neural network shown:

Superscript ANN level, subscript ANN neuron; Jacobi smoother



Final Layer – Jacobi iteration with bias

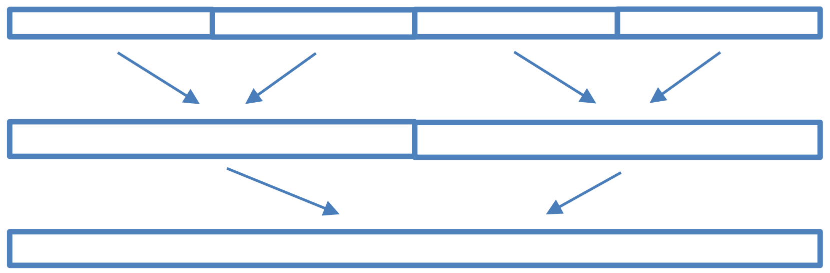
f is the bias



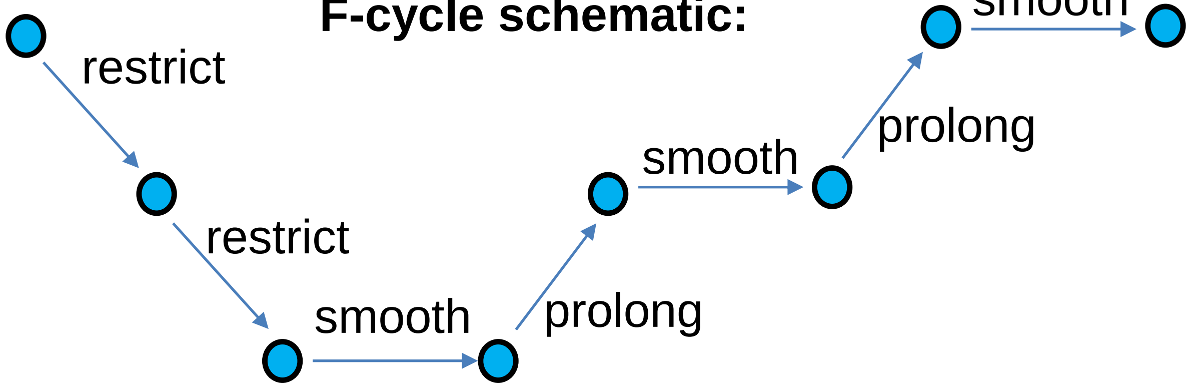
$\mathbf{x}^1 \rightarrow \mathbf{x}^1 + \Delta\mathbf{x}^8$ then repeat F cycle

For 1D linear DG: Have 2 channels. One for each DG node and use the inverse of the 2x2 matrix rather than $1/a^8_{33}$ etc. **3D linear DG** has 8 channels for each DG node.

Cells/nodes/neurons:



F-cycle schematic:



Jacobi relaxation when solving $Ax=s$:

Jacobi iteration: $x^{new}_i = (1/a_{ii}) (-\sum_j a_{ij}x^{old}_j + s_i)$

Suppose $\alpha \epsilon (0,1]$ is the relaxation coefficient (e.g. $\alpha=0.5$) then

$x_i = (1- \alpha)\, x^{old}_i + \alpha\, (1/a_{ii}) (-\sum_j a_{ij}x^{old}_i + s_i)$.

Thus $x_i = \sum_j b_{ij}x^{old}_j + c_i$

in which $b_{ij} = -\alpha\, (1/a_{ii})\, a_{ij}$ if $i \neq j$

and $b_{ii} = 1$

and $c_i = \alpha\, (1/a_{ii})\, s_i$ (is the bias)

Thus $w_{ij} = b_{ij}$ are the weights of the convolutional filter

Time stepping of $\partial T/\partial t + u\partial T/\partial x + v\partial T/\partial y + \sigma T - \mu \nabla . \nabla\, T = s$ (advection velocity (u,v)):

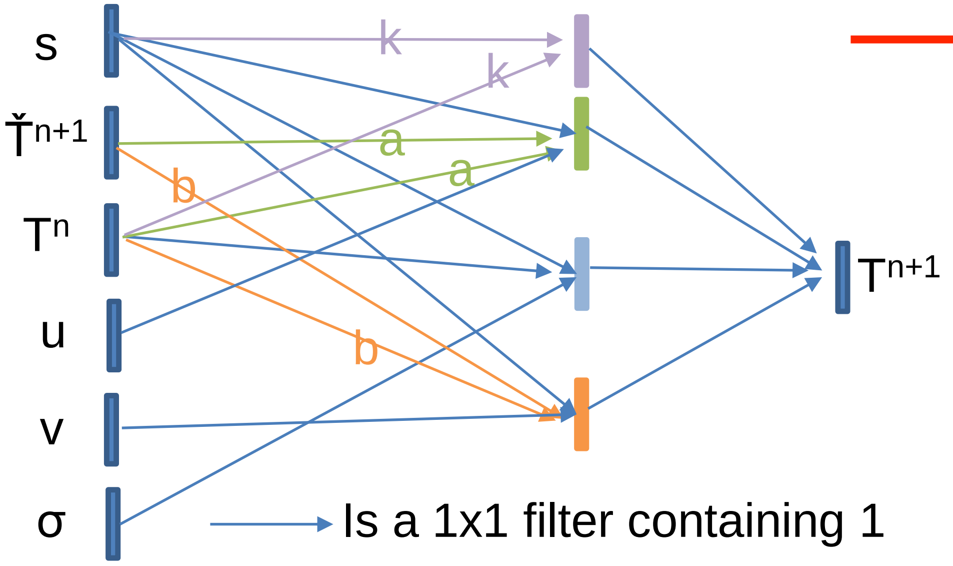
Two filters are needed a_{ij} and b_{ij}. The half in the bellow is used to obtain 2nd order accuracy in time, $\check{T}^{n+1}$ is the best guess for T^{n+1} which might be T^n. Δt = time step size. ả, β are advection/diffusion stencils/discretizations. Discretization becomes:

$$(T^{n+1}_i - T^n_i)/\Delta t + \sigma_i\, T^{n+1}_i + u_i \textstyle\sum_j \text{ả}_{ij}\,(1/2)\,(\check{T}^{n+1}_j + T^n_j) + v_i \sum_j \beta_{ij}\,(1/2)\,(\check{T}^{n+1}_j + T^n_j) + \sum_j k\mu_{ij}\,(1/2)\,(\check{T}^{n+1}_j + T^n_j) = s_i$$

in which $a_{ij} = (1/2)\,_j\,\text{ả}_{ij}\,\Delta t$, $b_{ij} = (1/2)\,\beta_{ij}\,\Delta t$ and $k_{ij} = (1/2)\,\mu_{ij}\,\Delta t$.

$$T^{n+1}_i = (1/(1+\Delta t\,\sigma_i))\,T^n_i + s_i + (u_i/(1+\Delta t\,\sigma_i))\textstyle\sum_j a_{ij}\,(\check{T}^{n+1}_j + T^n_j) + (v_i/(1+\Delta t\,\sigma_i))\sum_j b_{ij}\,(\check{T}^{n+1}_j + T^n_j) + \sum_j k_{ij}\,(\check{T}^{n+1}_j + T^n_j) \qquad (1)$$

This can be turned into a CNN as follows:

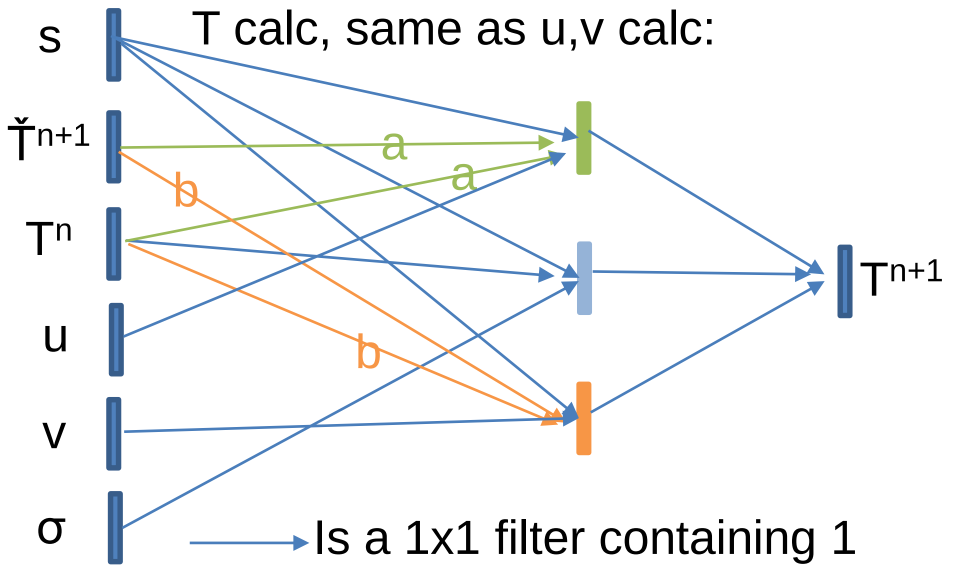


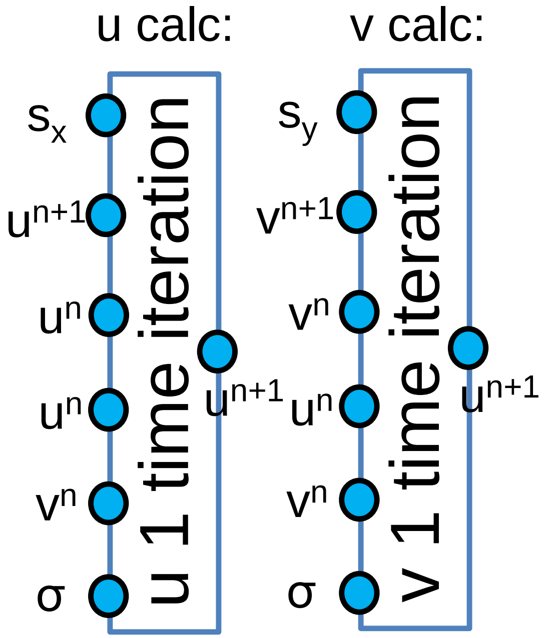
Is a 1x1 filter containing 1

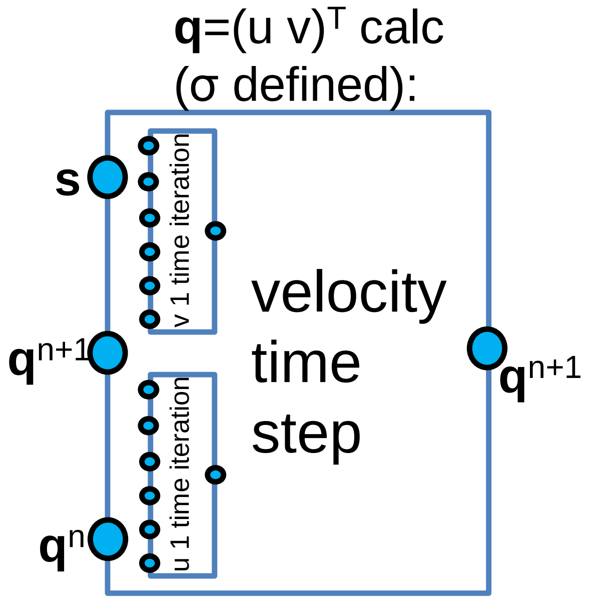
Change the activation functions to implement equation (1) above. The green and orange CNN layers have these tailored activation functions and the purple filter is for constant diffusion.

Use two steps of this CNN to obtain 2nd Order accuracy in time.

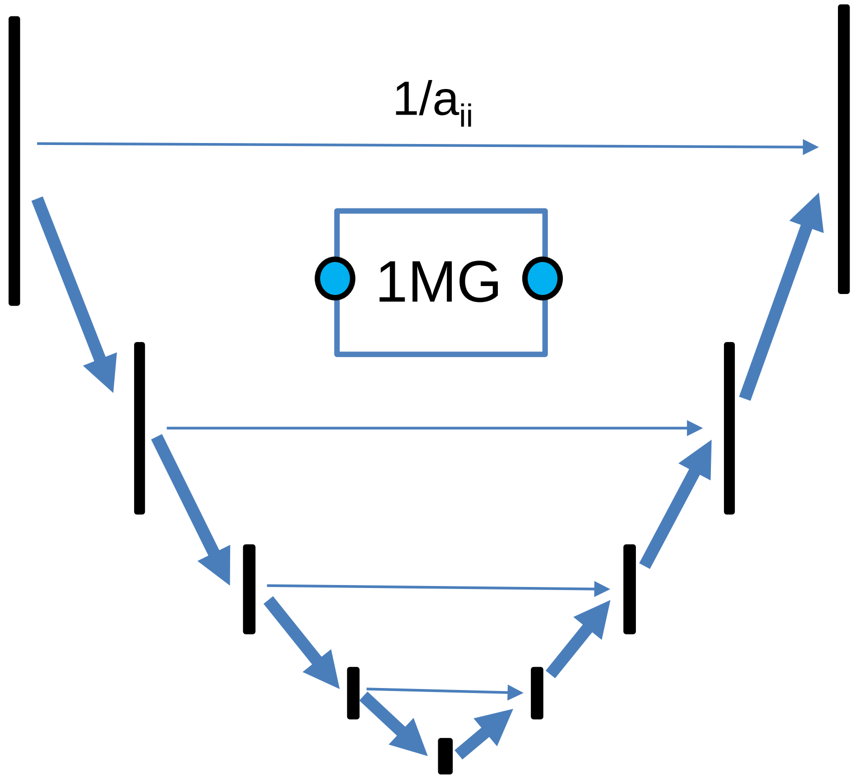
Time stepping of $\partial \mathbf{q}/\partial t + u\partial \mathbf{q}/\partial x + v\partial \mathbf{q}/\partial y + \sigma \mathbf{q} - \mu \nabla . \nabla \mathbf{q} = \mathbf{s}$ (advection velocity $\mathbf{q}=(u\ v)^T$ & source $\mathbf{s}=(s_x\ s_y)^T$):

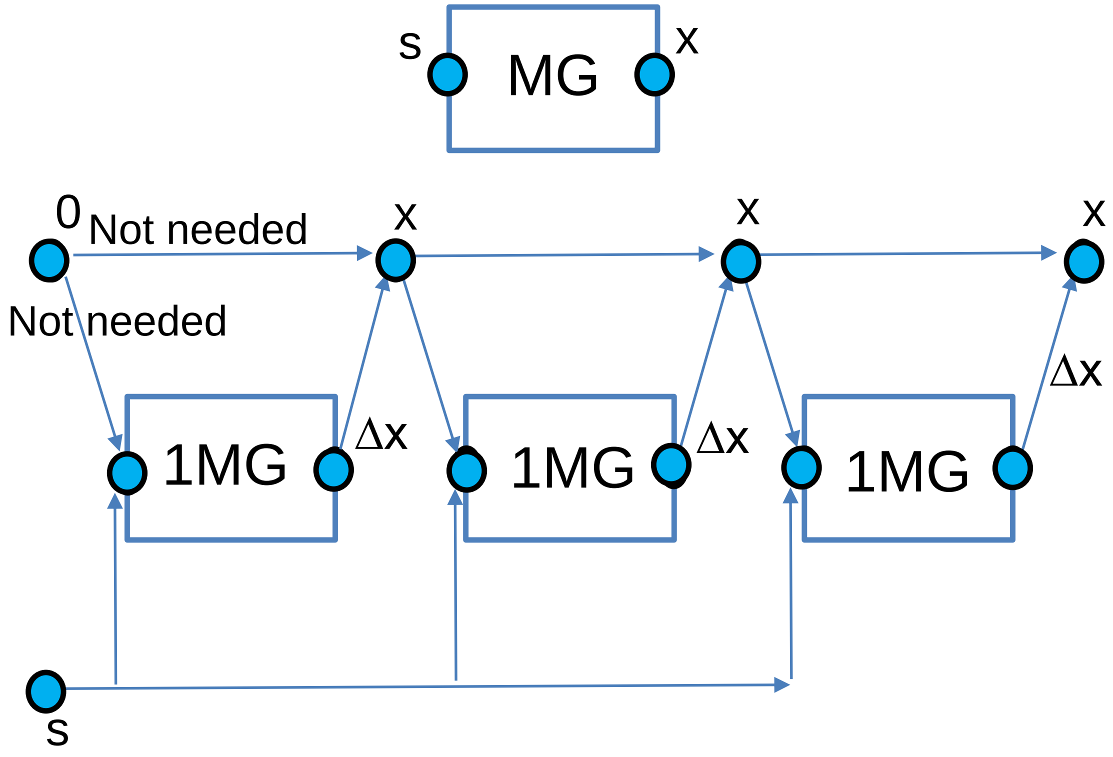




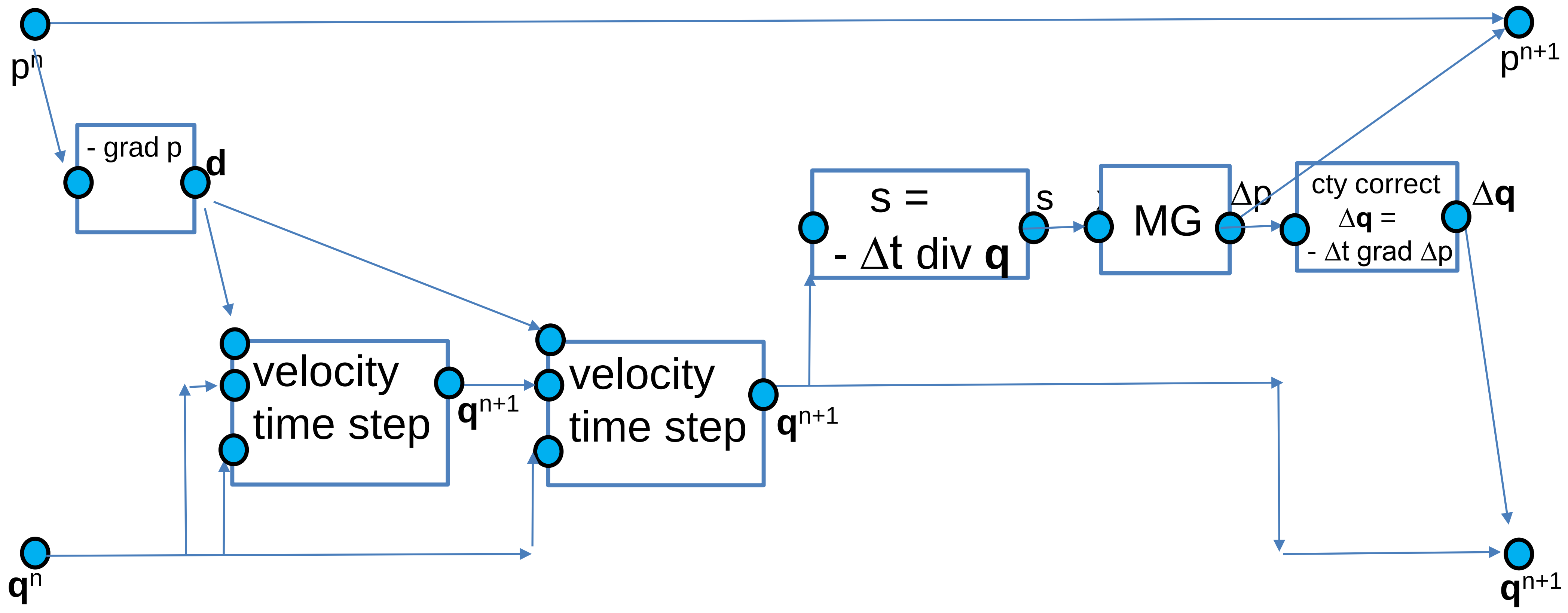


Left: Multi-Grid (MG) CNN skip layers rather than use biases (1MG cycle – similar to U-net); Right: Multi-Grid (3 iterations – cycles) solving Ax= s, starting from 0:





CFD time step:



Governing equation for projection base CFD:

Governing eqns ($\mathbf{q}=(u\ \ v)^T$)

$$D\mathbf{q}/Dt + \sigma\,\mathbf{q} - \mu\nabla.\nabla\mathbf{q} = -\,\mathrm{grad}\ p,$$

$$\mathrm{div}\ \mathbf{q} = 0.$$

Derived eqns – solution steps:

1) $D\mathbf{q}/Dt + \sigma\,\mathbf{q} - \mu\nabla.\nabla\mathbf{q} = -\,\mathrm{grad}\ p,$

2) $\nabla\,(1/(1+\Delta t\,\sigma))\,\nabla\Delta p = -\,(1/\Delta t)\ \mathrm{div}\ \mathbf{q},$

or for simplicity $\nabla\,.\,\nabla\Delta p = -\,(1/\Delta t)\ \mathrm{div}\ \mathbf{q}.$

3) $\Delta\mathbf{q} = -\,(1/(1+\Delta t\,\sigma))\ \mathrm{grad}\ \Delta p,$

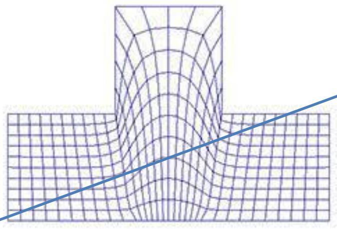
or for simplicity $\Delta\mathbf{q} = -\,\mathrm{grad}\ \Delta p.$

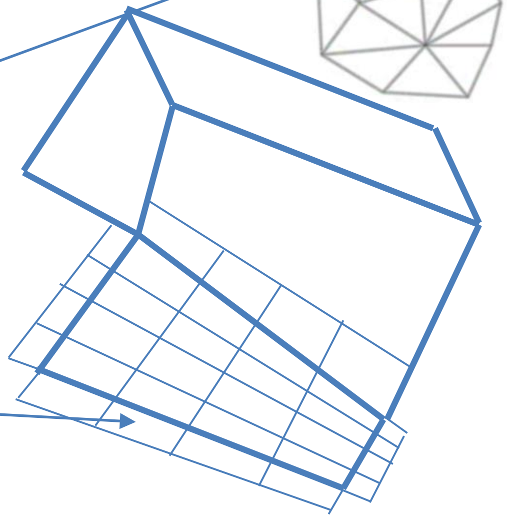
4) $p \rightarrow p + \Delta p.$

5) $\mathbf{q} \rightarrow \mathbf{q} + \Delta\mathbf{q}.$

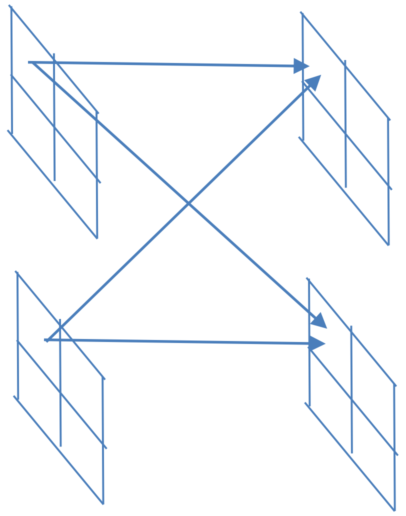
Matrix Free Boltzmann Transport Model for Neutronics/Radiation:

1) **Have a single filter for advection layer** for each SN direction to make method matrix free.
2) **Each SN direction has a channel.**
3) **For distorted blocks** have filters that interpolate between different orientations & element shapes.
4) **Use Multi-Grid** in DG Cartesian Space and SN angle. Collapse each **structured block** to single cell/node. Halo cells/DG nodes still structured.
5) **Use block unstructured** e.g. structured DG code.
6) **For linear DG** in 1D have 2 channels and replace a_{ij} with the inverse of the element block diagonal matrix. Similarly 3D have 8 channels.
7) **Start simple** with 1 block and 1 direction.
8) **For scattering removal** have a 2nd layer after advection layer, say. Could have a filter for each material in layer or have different weights of the connections – see pics bottom right.
9) **If massive number of angles** can efficiently have a single element in a structured block.

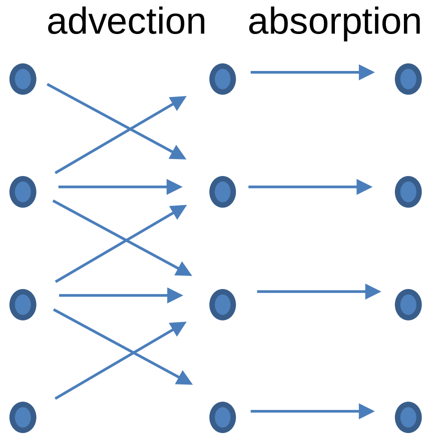




2D with 2 angles:



CNN Layers in 1D:



Boltzmann Details: Formation of SN DG Filters:

1) For structured block that is not distorted then a single filter is enough to represent $\Omega \cdot \nabla \Phi$ for each SN direction as each sub-element is the same.
2) For distorted blocks have form filters from spatial tables for each element and face of elements.
3) The filters for 3D DG 8 node bi-linear elements have 3 inner element contributions for each component of ∇. These are multiplied by Ω to achieve a discretization of $\Omega \cdot \nabla \Phi$ internal to an element. Each of these 3 filters has 8x8 values operating on the nodes of the element.
4) For the 3D DG 8 node element there for each face there is an incoming and an outgoing contribution filter set. Each of these has 3 components (filters) and the filters are each of size 4x4. There are thus 8 of these two surface filters (one for each element face) and thus there are 8 times 2 times 3 surface (=48) filters to represent the surface integrals.
5) For SN in angle multi-grid formation and using CNN then choose SN directions so as to map to a square grid.

Boltzmann Details: Solution Free Boltzmann transport – the solution vector too large to fit on computer e.g. for some GPU or AI computers say

The coarse grid in angle equation is:

$A\,\Phi \;+\; B\,\Psi \;= s_{\Phi}$

The fine grid equation is:

$C\,\Phi \;+\; D\,\Psi \;= s_{\Psi}$

In which Φ is the coarse grid in angle solution vector (e.g. scalar flux or low SN order angular flux) and Ψ is the fine SN angular vector. D is the discretized transport operator. The coarse grid in angle equation could be from diffusion theory (DSA) or low order SN.

Combining these the coarse grid equation becomes:

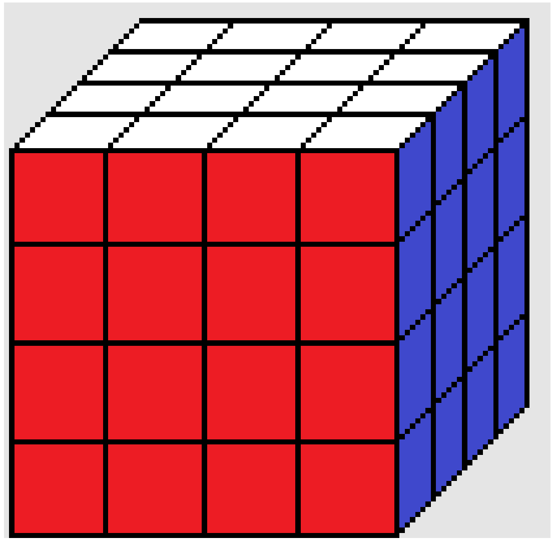
$(A \;- B\,D^{-1}\,C)\,\Phi \;= s_{\Phi} - \;BD^{-1}\,s_{\Psi}$

The basic idea is only to use the coarse (in angle solution) vector Φ and never actually form the whole solution vector Ψ or in the above $D^{-1}\,C\,\Phi$. Instead each direction is solved for with D^{-1} and is discarded after operating on B.

A multi-grid in angle and space can be applied to the SN in angle discretized system.

For time dependent one can only resolve the low order SN solution in time. For SN only include transport and total cross section terms in matrix D put other terms into the sources.

Boltzmann Details: Discretization in SN Angle and Cartesian Space on Structured Grids

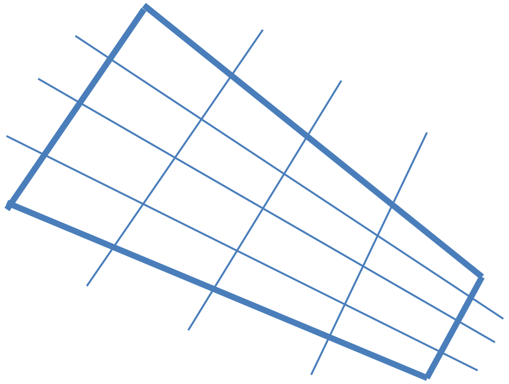


Discretization in SN Angle:

1) Use hexahedra on unit sphere and then put a rectangular grid across each square. See figure.
2) Have a power of 2 of the number of cells across on this rectangle e.g. 4 in the figure.
3) Each cell then represents an SN direction.

Discretization in Cartesian space:

1) Use DG in space with NxN elements and N is a power of 2 – needed for multi-grid method. See figure bottom right in which N=4.
2) For 4-node DG linear 2D element have 4 convolutional channels. One for each node.



Boltzmann Details: Convolutional Filters in SN Angle and Cartesian Space on Structured Grids

1) Assuming 1D in space and in angle (similar in multi-dimensions) have a different filter for each SN direction but same across space. See figure.

2) Add discretization filters together to form coarser filters.

3) In angle just add them together to form coarser SN discrete equations.

4) In Cartesian space use the prolongation and restriction matrices to form coarse discrete equations.

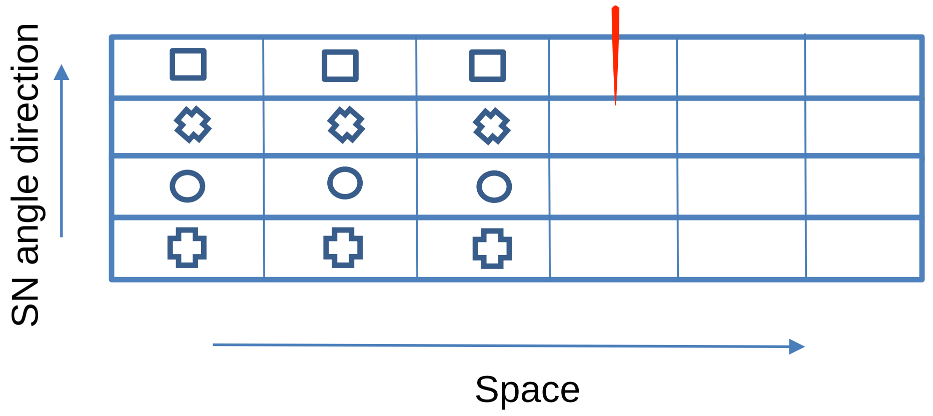


Figure: Space - SN angle CNN filters

Boltzmann Details: DG Prolongation and Restriction Matrices

Fine to coarse grid equation is:

$\int N_i^c (\psi_c - \psi_f)\, dV;$ $\psi_c = \sum N_j^c \psi_{cj};$ $\psi_f = \sum N_j^f \psi_{fj};$

$M^c \psi_c = N^{cf} \psi_f;$

Thus: $\psi_c = (M^c)^{-1} N^{cf} \psi_f = P^{cf} \psi_f$.

Course to fine grid:

$\int N_i^f (\psi_f - \psi_c)\, dV;$

$M^f \psi_f = N^{fc} \psi_c;$

$\psi_f = (M^f)^{-1} N^{fc} \psi_c = P^{fc} \psi_c$.

Therefore the fine grid matrix equation is: $A^f \psi_f = s_f$.

The coarse matrix: $P^{cf} A^f P^{fc} \psi_c = P^{cf} s_f = s_c$.

Or: $A^c \psi_c = s_c$.

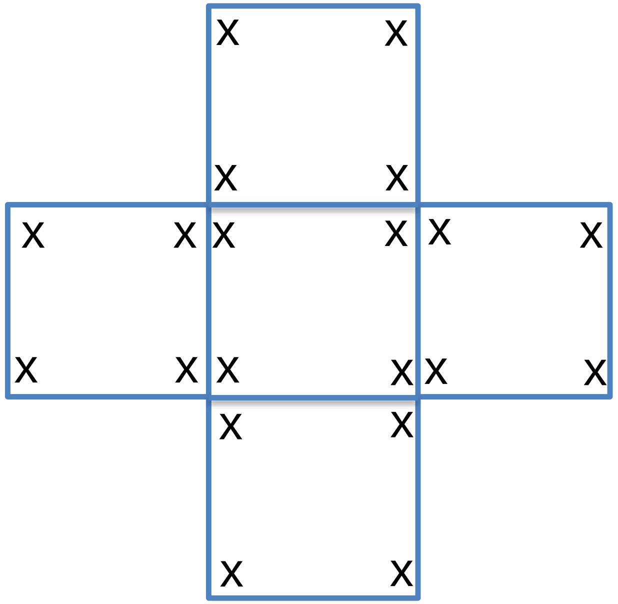
The filters are similarly operated on by these prolongation and restriction matrices: P^{fc}, P^{cf} .

Boltzmann Details: Structured mesh linear DG CNN filters in 2D

Suppose ψ^p is the SN direction angular flux with angle Ω^p and $\psi^p = \sum N_j \psi^p{}_j$.

The filter is the entire discretization centred on the inner element in the figure in which DG nodes are represented by x.
The filter is defined through the discretization for element e at the centre of figure:

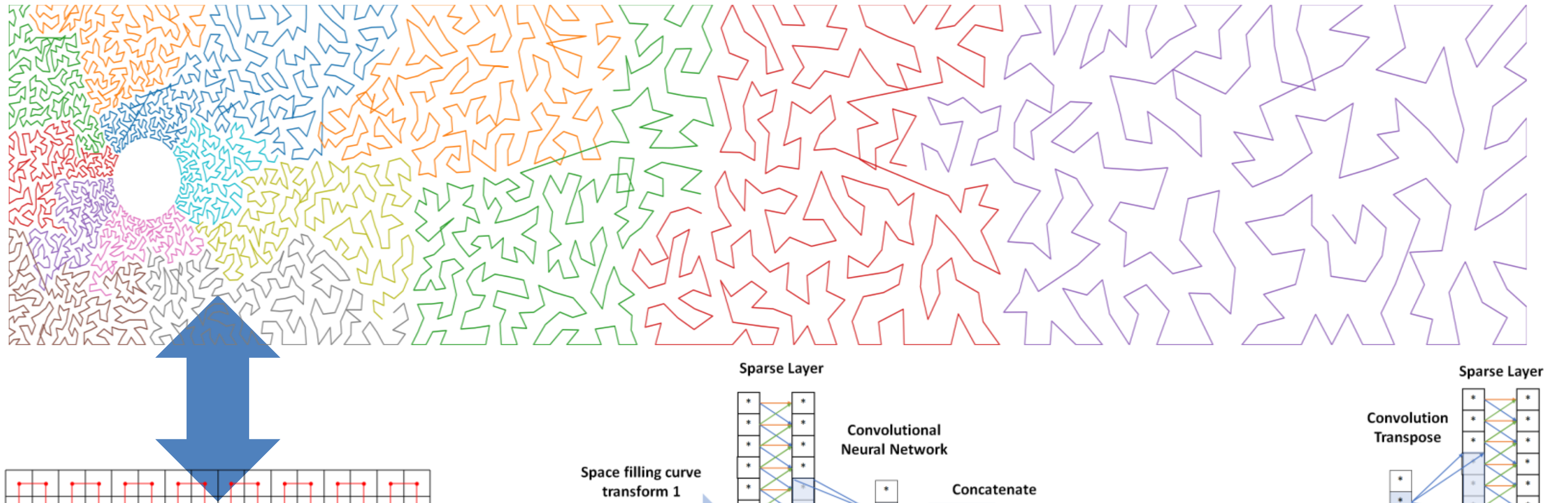
$$-\int_{Ve} (\nabla N_i) \cdot \Omega^p \psi^p \, dV + \int_{\Gamma e(\Omega.n<0)} N_i \Omega^p \cdot n \, \psi^p{}_{bc} \, d\Gamma + \int_{\Gamma e(\Omega.n>0)} N_i \Omega^p \cdot n \, \psi^p \, d\Gamma$$
$$+ \int_{Ve} N_i \sigma^p \psi^p \, dV = s^p$$

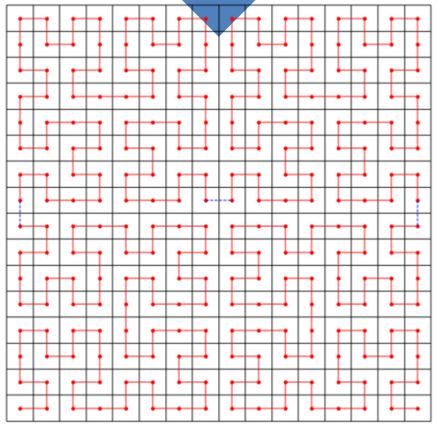


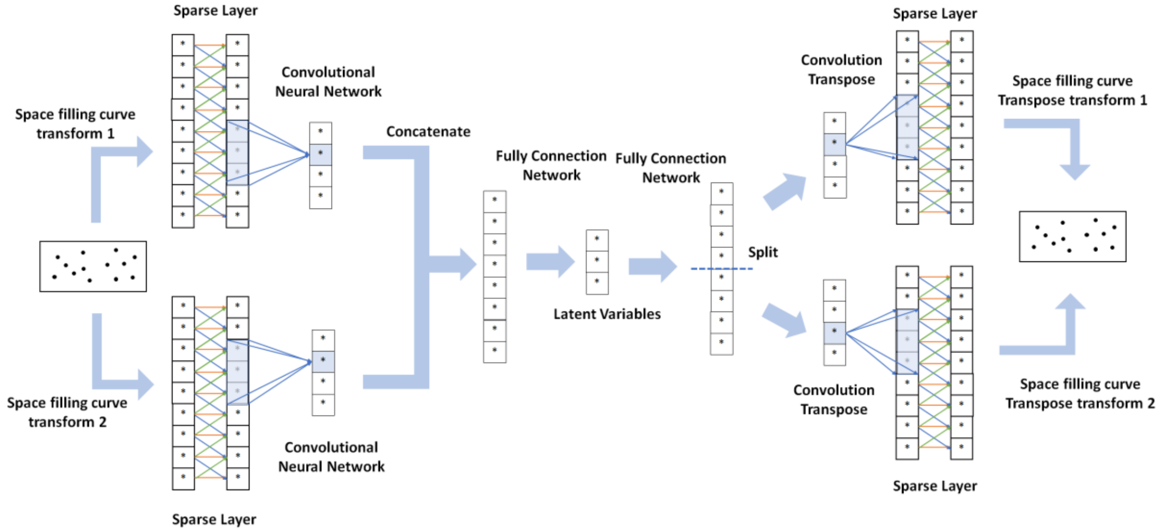
Test cases to demonstrate approach – simple to complex:

1) Time dependent 2D advection diffusion equation on structured grid with central difference discretization. (done)
2) 2D Burgers equation discretization and solution with non-linearity built into network. (done)
3) Multi-grid on 2D advection diffusion equation. (done)
4) Discretize fluids equation – flow past block – with central difference and projection method Poisson equation for pressure so can use an A grid.
5) 2D Burgers equation to demonstrate adjoint.
6) Differentiate fluids equation to find adjoint by defining activation functions.
7) Discretize Boltzmann Transport equation with 4D multi-grid on a structured grid – start with diamond differencing then DG in 2D and with SN.
8) Solve OPS/2 test cases: convection test case.
9) Parallel for structured grid advection-diffusion, fluids and AI model.
10) Unstructured and semi-structured mesh multi-grid based on space-filling curves or Earth mover distance optimization (see unstructured mesh slides).

Unstructured mesh CNN - Multi-dimensional convolutional methods – extension of 1D Space Filling Curve (SFC) method

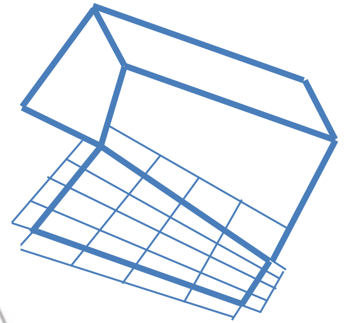


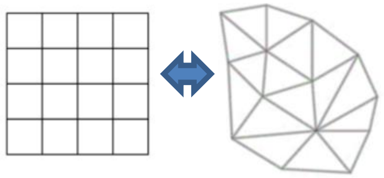


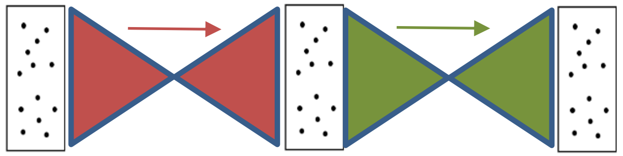

Sparse Layer
Space filling curve transform 1
Convolutional Neural Network
Concatenate
Fully Connection Network
Fully Connection Network
Latent Variables
Split
Convolution Transpose
Sparse Layer
Space filling curve Transpose transform 1
Space filling curve transform 2
Convolutional Neural Network
Sparse Layer
Convolution Transpose
Sparse Layer
Space filling curve Transpose transform 2

Unstructured meshes and parallel

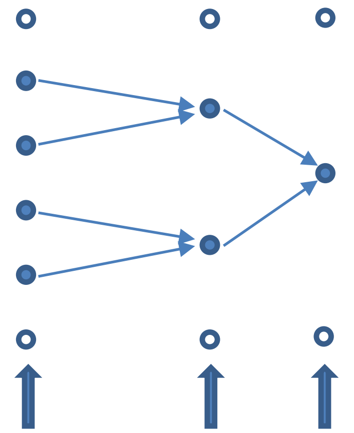
1) Use semi-structured or block approach with unstructured grid representing the interaction between the blocks to start with. See figure.
2) Use optimal mappings between unstructured and structured meshes (interaction between structured blocks or elements) based on the earth mover's distance (the Wasserstein metric) using a NN for: a) the mapping of the subdomains to the AI computer cores taking its architecture; b) mapping subdomains to a 2D or 3D structured grid on which the multi-grid CNN can be applied.
3) For multi-grid solver either use SFC approach on unstructured representation or structured CNN formed from earth movement distance optimization.
4) For fully unstructured meshes (not nec. block structured) use a graph structure for the CNN and have different weights on all the edges of this graph.
5) For parallel have a halo update on each coarsening level of the multi-grid CNN method.







2 SFC MG CNN unstructured



MG CNN halo update