# Horologium: Cycle‑Accurate, Hardware‑Aware NN Model — Detailed README

This document is a ground‑truth, code‑aligned description of the model engine. It cross‑checks the shipped notebook (model.ipynb) and the two HW/Arch profiles (HW\_Settings\_\*.py, Arch\_\*.py). After reading it, users can understand the model’s moving parts, the tick‑level lifecycle, and how capacity/bandwidth/energy are accounted.

## 0) Audience & scope

* **Audience:** AI accelerators engineers or researchers.
* **Scope:** run‑time scheduler, dataflow, arbitration (ITF), module contracts (SYA/POL/RES/CNC/FPS/KNN), statistics.
* **Out‑of‑scope:** algorithmic accuracy of networks; exact SRAM/NoC micro‑architecture.

## 1) Repository layout & roles (what each file owns)

* **HW\_Settings\_PTV2.py / HW\_Settings\_diffusion.py** — Chip processing technology, bit‑width, frequency, GLB geometry, off‑chip rate, innovative technology, and global switches. Examples:
  + BITWIDTH, BITWIDTH\_RATE, FREQ (MHz), Cnt\_Step, ITFPRE, FRE\_RATIO, GLB\_NUMSRAM, GLB\_NUMBYTE (Diffusion only), ENA\_RELY, ENA\_AHEAD, ENA\_PREDICT, power model knobs (e.g., Energy\_1bit\_EMA).
* **Arch\_PTV2.py / Arch\_diffusion.py** — Stage/layer architecture:
  + NN\_ARCH[stg][lay] node attributes (e.g., SRC, RES, CNC, OUT), NN\_POL\_ENA[stg]/NN\_KNN\_ENA[stg]/NN\_FPS\_ENA[stg], switch of per‑stage POL/KNN/FPS module.
  + NN\_NUM\_CNV[stg], number of each stage
  + NN\_NUM\_INPUT\_STG[stg]/NN\_NUM\_OUTPUT\_STG[stg]/NN\_NUM\_OUTPUT\_STG\_POL[stg], input number/output number of each stage(when NN\_POL\_ENA[stg]=True, output number rely on NN\_NUM\_OUTPUT\_STG\_POL[stg])
* **model.ipynb** — The cycle‑accurate engine: request generation, interface arbitration (ITF), GLB state updates, per‑module state machines, indexing, and statistics.

**Rule of thumb:** HW files define *physics* (rate, capacity, units); Arch files define *architecture*.

## 2) Key concepts

### 2.1 Blocks & GLB

* **GLB (Global Buffer)** is modeled in *words*. Each tensor is partitioned into **blocks** along points/channels so that transfers are sized in block‑words.
* Three canonical block roles:
  + **IFM** (input feature map) — what SYA consumes.
  + **FIL** (filter/weight) — what SYA consumes.
  + **OFM** (output feature map) — what SYA/POL/RES produce for the next consumer.
* Visibility is tracked with boolean maps (subset shown):
  + SYA\_IFMBlkOn, SYA\_FilBlkOn, SYA\_OFMBlkOn, SYA\_OFMBlkOn\_OUT
  + POL\_OFMBlkOn
  + KNN\_MapBlkOnC, KNN\_MapBlkExt
  + FPS\_InCrdOn
  + Each entry is also mirrored in a GLBMap (a list‑of‑indices registry) so we can iterate visible blocks without scanning full 4‑D arrays. Greatly improved the running speed of the model.

### 2.2 Units & timebase

* **Tick size:** Cnt\_Step cycles per scheduling update.
* **Time:** Stat\_InfTime = CntClk / (FREQ \* 1e6) (MHz → seconds). Energy is accumulated by integrating model power over time (Stat\_Energy = Stat\_Power \* Stat\_InfTime).
* **Transfer slope:** FRE\_RATIO maps interface ticks to words; ITFPRE adds a preamble (setup) latency.
* **Profiles differ:** PTV2 (8‑bit, 200 MHz, smaller GLB, FRE\_RATIO≈0.5×) vs. Diffusion (1‑bit, 1 GHz, larger GLB, GLB\_NUMBYTE=32, FRE\_RATIO≈16×).

### 2.3 Indexing & graph

* Four nested indices drive the run: (InfIdx, StgIdx, LayIdx, BlkIdx).
* NN\_ARCH holds per‑node flags (e.g., SRC, RES, CNC) and **OUT edges** describing who consumes the result.

## 3) The Interface (ITF)

**Mission:** Convert module‑side read/write requests into **one real transfer at a time**, with fixed priorities and atomic completion pulses that update GLB state and stats.

### 3.1 Lifecycle per transfer

1. **Arbitrate** (only when ITF\_Busy == False): Pick the first eligible request following a fixed order.
2. **Move**: Advance ITF\_CntComp with Cnt\_Step. After ITFPRE, move at a slope of FRE\_RATIO words/tick until the request’s \*\_Word\* threshold is met.
3. **Complete**: Emit the one‑cycle \*\_Fnh\* pulse, update GLB visibility maps and capacity counters, set ITF idle.
4. **Bookkeep**: Record direction (ITF\_GLBWr/ITF\_GLBRd) for bandwidth stats and increment module/ITF counters (e.g., Stat\_Cnt\_\*).

### 3.2 Fixed arbitration order (highest → lowest)

SYA\_RdIFM → SYA\_RdFil → SYA\_WrOFM → SYA\_WrIFM → POL\_RdMap → POL\_RdOFM → POL\_WrOFM → RES\_RdOFM(IN0) → RES\_RdOFM(IN1) → CNC\_RdIN0 → CNC\_RdIN1 → KNN\_WrMap → KNN\_RdCrd → FPS\_RdCrd → FPS\_WrCrd → idle

**ArbIdx mapping (for debugging)**

| ArbIdx | Request | Direction |
| --- | --- | --- |
| 20 | SYA\_RdReqIFM | GLB ← ext |
| 21 | SYA\_RdReqFil | GLB ← ext |
| 22 | SYA\_WrReqOFM | GLB → ext |
| 23 | SYA\_WrReqIFM | GLB → ext |
| 30 | POL\_RdReqMap | GLB ← ext |
| 31 | POL\_RdReqOFM | GLB ← ext |
| 32 | POL\_WrReqOFM | GLB → ext |
| 40 | RES\_RdReqOFMIN0 | GLB ← ext |
| 41 | RES\_RdReqOFMIN1 | GLB ← ext |
| 50 | CNC\_RdReqIN0 | GLB ← ext |
| 51 | CNC\_RdReqIN1 | GLB ← ext |
| 10 | KNN\_WrReqMap | GLB → ext |
| 11 | KNN\_RdReqCrd | GLB ← ext |
| 0 | FPS\_RdReqCrd | GLB ← ext |
| 1 | FPS\_WrReqCrd | GLB → ext |

**Note:** “GLB ← ext” denotes an **off‑chip → GLB** read; “GLB → ext” denotes a **GLB → off‑chip** write (or externalization).

### 3.3 Completions (one‑cycle pulses)

* SYA: ITFSYA\_RdFnhIFM, ITFSYA\_RdFnhFil, ITFSYA\_WrFnhOFM, ITFSYA\_WrFnhIFM
* POL: ITFPOL\_RdFnhMap, ITFPOL\_RdFnhOFM, ITFPOL\_WrFnhOFM
* RES: ITFRES\_RdFnhOFMIN0, ITFRES\_RdFnhOFMIN1
* CNC: ITFCNC\_RdFnhIN0, ITFCNC\_RdFnhIN1
* KNN: ITFKNN\_WrFnhMap, ITFKNN\_RdFnhCrd
* FPS: ITFFPS\_RdFnhCrd, ITFFPS\_WrFnhCrd

## 4) Module contracts

Each module has: **requests**, **completions**, **compute gating**, **index advance**, and **GLB updates**. The engine relies on these contracts.

### 4.1 SYA (convolution / main compute)

**Role:** Consume IFM + FIL; produce OFM and expose it to the next consumer.

**Requests (SYA → ITF):** - SYA\_RdReqIFM — Fetch an IFM block (may prefetch under AHEAD). - SYA\_RdReqFil — Fetch a FIL block. - SYA\_WrReqOFM — Externalize a completed OFM block. - SYA\_WrReqIFM — Spill/release a chosen IFM block (capacity relief path).

**Completions (ITF → SYA):** ITFSYA\_RdFnhIFM, ITFSYA\_RdFnhFil, ITFSYA\_WrFnhOFM, ITFSYA\_WrFnhIFM.

**Compute gating:** SYA\_Comp = IFMOn & FilOn & !GLB\_BWFull & UTI(SYA) & RES\_not\_blocking. When SYA\_Comp holds, SYA\_CntComp += Cnt\_Step. After SYAPRE, OFM slices start becoming ready; once a block is complete, issue SYA\_WrReqOFM.

**OFM exposure & routing:** - Default path: OFM becomes next layer’s IFM (same stage) or the first layer of the next stage. - If stage‑tail and NN\_POL\_ENA[stg] is true → route OFM to **POL** (see 4.2). - If residual is declared (RES=True branch exists) → OFM feeds **RES** as one of its inputs. - **Normal Turn (OFM→IFM)**: when the path is simple (no POL/RES/CNC), *rename* OFM→IFM in GLB: mark target SYA\_IFMBlkOn[...] = True, decrease consumed OFM words, increase IFM words; capacity is conserved.

**Capacity relief policy (when GLB is tight):** 1) Try to **write POL’s OFM** if a POL block is ready (POL\_WrReqOFM). 2) Else **release non‑current FIL** blocks (keep current/next rows pinned). 3) Else **spill one IFM** block via SYA\_WrReqIFM. This path is triggered when GLB\_WORD − GLB\_CapUlti ≤ threshold or when pending reads require capacity.

**Statistics:** Stat\_Cnt\_SYA\_\* (RdIFM/RdFil/WrOFM/WrIFM), Stat\_SYA\_CntComp, Stat\_SYA\_ActRatio, Stat\_Time\_SYA\_\* (e.g., IFM/FIL not on, wait for POL, wait for GLB).

### 4.2 POL (pooling / aggregation)

**Role:** Consume upstream OFM (often last SYA in a stage) and a **Map** (often from KNN), produce a pooled/aggregated OFM for the next consumer. User can set the sample rate “K” in ARCH.py, which means “K” blocks down-sample to 1 block or 1 block generate “K” blocks.

**Requests:** POL\_RdReqMap, POL\_RdReqOFM, POL\_WrReqOFM.

**Completions:** ITFPOL\_RdFnhMap, ITFPOL\_RdFnhOFM, ITFPOL\_WrFnhOFM.

**Sequencing:** Read **Map first**, then **OFM**, then **write OFM**. When KNN is disabled, treat the Map as identity (skip the read).

**Compute gating & timing:** POL\_Comp = MapOn & OFMOn & !GLB\_BWFull & UTI(POL). After warm‑up (POLPRE), issue POL\_WrReqOFM.

**GLB updates:** - On Map/OFM read finish: set KNN\_MapBlkOnC / SYA\_OFMBlkOn\_OUT visibility for the current block. - On write finish: set POL\_OFMBlkOn for the produced block and, if ENA\_RELY allows and no other consumer needs the source, release the consumed upstream OFM.

### 4.3 RES (residual)

**Role:** Align and combine **two** upstream OFMs (main + skip); typically **no explicit ITF write**—the combined result becomes visible to the downstream consumer.

**Requests:** RES\_RdReqOFMIN0, RES\_RdReqOFMIN1 (independent; each arbitrated separately).

**Completions:** ITFRES\_RdFnhOFMIN0, ITFRES\_RdFnhOFMIN1.

**Combine rule:** RES remains “two‑in one‑out” without extra writes. Usually use two existed OFM perform operations on corresponding elements.

**Timing:** When both inputs are on, RES\_Comp holds; accumulate RES\_CntComp and, after RESPRE, mark the residual output visible to the downstream (often by directly toggling next SYA’s IFM map). Then advance (Blk→Lay→Stg).

### 4.4 CNC (concat)

**Role:** Pure routing/reshape between nodes (e.g., concat along channel dim or fan‑out to multiple consumers). CNC **does not perform heavy compute**; it turns source OFM blocks into target IFM, optionally across stages.

**Requests / completions:** CNC\_RdReqIN0/IN1 with ITFCNC\_RdFnhIN0/IN1.

**Index advance:** When the target IFM block becomes fully ready (and sources are released as needed), advance CNC\_BlkIdx/LayIdx/StgIdx.

### 4.5 KNN (neighbor map producer)

**Role:** Produce the **Map** that POL consumes; also reads coordinates.

**Requests / completions:** KNN\_WrReqMap + ITFKNN\_WrFnhMap, KNN\_RdReqCrd + ITFKNN\_RdFnhCrd.

**Notes:** Map blocks are tracked via KNN\_MapBlkOnC / KNN\_MapBlkExt and sized consistently with POL’s consumer blocks.

### 4.6 FPS (feature/point sampler)

**Role:** Provide coordinate streams used by KNN or other consumers.

**Requests / completions:** FPS\_RdReqCrd / ITFFPS\_RdFnhCrd, FPS\_WrReqCrd / ITFFPS\_WrFnhCrd.

## 5) Tick‑level main loop

At each scheduler step (size = Cnt\_Step cycles):

1. **Clear one‑shot pulses** from the previous step and reset per‑step request bits (\*\_WrReq\* = False by default).
2. **Generate requests** in each module based on its readiness and policies:
   * SYA forms RdIFM/RdFil; may form WrOFM; under pressure may form WrIFM.
   * POL forms RdMap → RdOFM → WrOFM in order.
   * RES forms RdOFM0/RdOFM1.
   * CNC forms RdIN0/RdIN1 as needed to expose IFM for its consumer.
   * KNN/FPS form their reads/writes based on stage progress.
3. **Arbitrate in ITF** if idle → select exactly one request (see §3.2), set ITF\_Busy/ITF\_ArbIdx.
4. **Move data** until word threshold met (preamble ITFPRE, slope FRE\_RATIO).
5. **Complete** the transfer: raise \*\_Fnh\* pulse for exactly one step, update GLB visibility (\*\_BlkOn/\*\_BlkExt), and adjust GLB capacity counters for the producer/consumer.
6. **Compute** in modules whose inputs are on and which pass utilization gates (e.g., should\_compute(CntClk, SYAUTI)), advancing \*\_CntComp.
7. **Write** when the OFM is no longer consumed by downstream.
8. **Advance indices** when a block/layer/stage is finished.
9. **Statistics:** update Stat\_Cnt\_\*, Stat\_Time\_\*, Stat\_ITF\_\*, and power/energy/time accumulators.

## 6) Word & block sizing

* SYA **read IFM/FIL words** per block from current (stg, lay) and **write OFM words** for the target consumer.
* POL **read Map** and **read OFM** sized for its output block geometry; **write OFM** sized for the next stage/layer.
* RES **reads** two OFM blocks; its output visibility is an in‑GLB mark (no separate ITF write by default).
* CNC **reads** from its input edge(s) and marks target IFM on; it uses the target block size (generally consistent).
* KNN/FPS words are sized by their coordinate payload and sampling parameters.

The formulas in the notebook derive block counts roughly as: NUMBLK ~ ceil(points / (PE\_ROW \* PE\_BANK)), and NUMWORD ~ ceil( (PE\_ROW \* PE\_BANK) / GLB\_NUMBYTE \* ceil(channels / PE\_COL) ), with minor adjustments per operator.

## 7) Capacity & back‑pressure (what happens when GLB is near full)

* The engine monitors **GLB used words** (GLB\_CapUlti) vs **GLB total** (GLB\_WORD).
* If **nearly full** or if a module **requires capacity** for a pending read, the relief policy (see §4.1) engages in this order:
  1. externalize a ready POL OFM; 2) release a non‑current FIL block; 3) spill one IFM block.
* This policy respects **RELY** and **AHEAD** switches.

## 8) Initialization (t = 0)

* **ITF**: ITF\_Busy=False, ITF\_CntComp=0, ITF\_ArbIdx=-1, all completion pulses low, bandwidth flags clear.
* **GLB**: capacity counters zero; all \*\_BlkOn maps false.
* **Modules**: all requests low; counters zero. Seed only the **source** data flagged by SRC nodes in NN\_ARCH.
* **Main loop** then proceeds as in §5.

## 9) Signals & quick reference

### 9.1 Requests

* **SYA**: SYA\_RdReqIFM, SYA\_RdReqFil, SYA\_WrReqOFM, SYA\_WrReqIFM
* **POL**: POL\_RdReqMap, POL\_RdReqOFM, POL\_WrReqOFM
* **RES**: RES\_RdReqOFMIN0, RES\_RdReqOFMIN1
* **CNC**: CNC\_RdReqIN0, CNC\_RdReqIN1
* **KNN**: KNN\_WrReqMap, KNN\_RdReqCrd
* **FPS**: FPS\_RdReqCrd, FPS\_WrReqCrd

### 9.2 Completions (one‑shot pulses)

* **SYA**: ITFSYA\_RdFnhIFM, ITFSYA\_RdFnhFil, ITFSYA\_WrFnhOFM, ITFSYA\_WrFnhIFM
* **POL**: ITFPOL\_RdFnhMap, ITFPOL\_RdFnhOFM, ITFPOL\_WrFnhOFM
* **RES**: ITFRES\_RdFnhOFMIN0, ITFRES\_RdFnhOFMIN1
* **CNC**: ITFCNC\_RdFnhIN0, ITFCNC\_RdFnhIN1
* **KNN**: ITFKNN\_WrFnhMap, ITFKNN\_RdFnhCrd
* **FPS**: ITFFPS\_RdFnhCrd, ITFFPS\_WrFnhCrd

### 9.3 GLB visibility maps (subset)

SYA\_IFMBlkOn, SYA\_FilBlkOn, SYA\_OFMBlkOn, SYA\_OFMBlkOn\_OUT, POL\_OFMBlkOn, KNN\_MapBlkOnC, KNN\_MapBlkExt, FPS\_InCrdOn.

## 10) How to use

1. Pick **one** HW profile and import it (e.g., PTV2 for 8‑bit @ 200 MHz or Diffusion for 1‑bit @ 1 GHz). Make sure Cnt\_Step, FRE\_RATIO, GLB\_NUM\* align.
2. Pick the matching **Arch** (PTV2 or Diffusion). Confirm NN\_POL\_ENA and OUT edges.
3. Run the main loop. Inspect: GLB\_Cap\*, \*\_CntComp, Stat\_Time\_\*, Stat\_Cnt\_\*, Stat\_InfTime/Power/Energy, etc.

*End of README.*