Multimodal Intelligence for Therapeutics and Clinical Insight

Research Methods

1. Research Outline

Program goal. As shown in Figure 1. We will build two tightly coupled computational tracks that share a unified GPU/CPU workflow: (1) a Multimodal Generative Therapeutics Engine that fuses diffusion-based molecule generation with an agentic peptide design loop and a frozen-backbone bridge to large language models (LLMs) for chemically grounded reasoning; and (2) an Interpretable Clinical Vision Intelligence stack that uses a vision-language model (VLM) to extract standardized phenotypes from pediatric facial photos and extends to pathology-scale H-score quantification. Together these tracks convert large-scale compute into reproducible models, candidate libraries, and clinically relevant analytics.

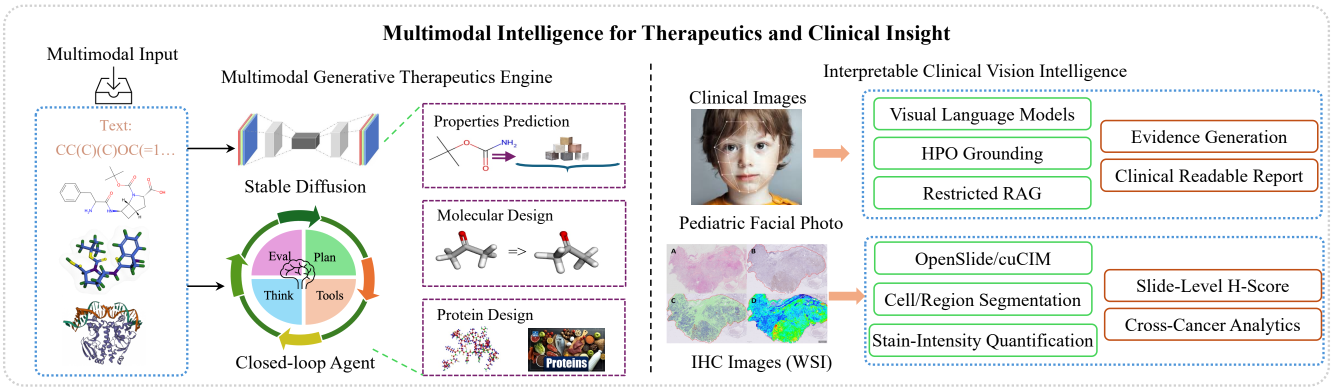

Figure 1: Research outline illustration for project multimodal intelligence for therapeutics and clinical insight.

Project 1 – Multimodal Generative Therapeutics Engine.

We will build an AI "design studio" for new therapeutics that can propose both small-molecule drugs and therapeutic peptides. For small molecules, we use a modern generative model (stable diffusion-based) that explores the 3D shape and chemistry of candidates and then automatically checks multiple requirements at once, drug-likeness, ease of synthesis, and predicted binding to a given target. The key idea is simple: the model not only scores each property but also knows how confident it is; we turn those multi-property scores into a single go/no-go signal that teaches the model to prefer strong, realistic candidates. For peptides, we run a closed loop: specialized evaluators (activity, toxicity, structure) act like "peer reviewers," give structured feedback, and the generator improves accordingly. Both parts plug into a novel lightweight bridge that lets a large language model read chemical features (topology graph and geometry) and explain why a design was proposed, so that decisions can be auditable instead of a black box. This track needs GPUs because we train large foundation models and screen millions of candidates to yield ranked, synthesis-ready shortlists for wet-lab testing.

Project 2 – Interpretable Clinical Vision Intelligence.

We will turn clinical images into standardized, evidence-backed assessments. For pediatric rare-disease faces, the system first proposes a small set of likely conditions with calibrated confidence and then translates visible traits into standard medical terms (HPO, Human Phenotype Ontology), citing the knowledge sources it used, so that clinicians can see both the suggestion and the rationale. For pathology, we automate H-score (the 0–300 immunohistochemistry score pathologists use) on whole-slide images by consistently tiling slides, segmenting cells, and measuring stain intensity, producing numbers that are comparable across cancer types. Outcomes are practical: faster triage for rare-disease clinics, and reproducible, quantitative pathology readouts that link to prognosis or treatment decisions.

2. Expected Outcomes

Therapeutics (Project 1). We will deliver (1) a ranked, synthesis-ready library of small molecules and therapeutic peptides for 1–2 predefined targets (e.g., Pin1, a cancer-relevant prolyl isomerase), and (2) reusable generative pipelines (diffusion models + agentic reinforcement learning) with auditable LLM rationales. Success metrics: screening throughput (≥10^6 candidates/target), property pass rates against drug-likeness/SA (synthetic accessibility)/ADMET (absorption, distribution, metabolism, excretion, toxicity) thresholds, docking/affinity gains over strong baselines, and diversity/novelty (Tanimoto, Bemis–Murcko). We aim for ≥5× hit enrichment in prospective wet-lab assays (top-N) and, for peptides, ≥2× activity improvement at matched toxicity with ≤100 high-priority sequences per campaign. Outputs: trained checkpoints, containerized workflows (Slurm-ready), and complete design to explanation audit logs.

Clinical vision (Project 2). We will release (1) a VLM (vision-language model) prototype for pediatric rare-disease triage that provides calibrated top-k suggestions plus HPO (Human Phenotype Ontology)–grounded explanations, and (2) a scalable H-score pipeline for whole-slide IHC (immunohistochemistry) images. Diagnostic utility will be quantified by top 5 recall and risk-controlled coverage (conformal prediction), calibration error (ECE, expected calibration error), and explanation faithfulness (precision/recall against expert HPO annotations). For H-score, we target inter-rater reliability ICC ≥0.85 across sites and robust performance on multi-cancer cohorts. Outputs: inference notebooks, model weights (where licensing permits), and Standard Operating Procedures for deployment and quality assurance.

3. Progress Over the Past Year

Over the past year our lab consolidated an end-to-end pipeline that now directly supports Project 1 (Multimodal Generative Therapeutics Engine) and Project 2 (Interpretable Clinical Vision Intelligence)**.** On the therapeutics side, we advanced uncertainty-aware diffusion + Reinforcement Leaning (RL) for multi-objective molecule design; this work, led by our team, was **accepted at NeurIPS 2025**[1], with validated gains on QM9/ZINC15/PubChem and plans for target-conditioned discovery (e.g., Pin1). In parallel, we built MAC-AMP, a closed-loop multi-agent system for antimicrobial peptide design that performs AI "peer review," is fully implemented on multiple bacteria, and is under external review to provide the agentic RL substrate for Project 1's peptide track. These advances were underpinned by a string of peer-reviewed outputs in top venues that strengthen our property prediction and representation learning: MolGraph-xLSTM (**Communications Chemistry**)[2], FusionLCLM (**Journal of Cheminformatics**)[3], CL-MFAP (**ICLR 2025**)[4], and GraphBAN (**Nature Communications**)[5]. Collectively, they provide stronger encoders/predictors that we now plug into the generative loop for screening, docking, and prioritization.

On the clinical vision side, we prototyped an **image-first, interpretable VLM** for pediatric rare-disease triage that combines calibrated visual agents, HPO-grounded phenotype extraction, and restricted RAG, transitioning from black-box classification to auditable reasoning for Project 2. In computational pathology, we completed and validated a **PDAC Pin1 H-score** pipeline (tiling/normalization → cell segmentation → stain-intensity quantification) and began scaling to **TCGA** breast/lung/colorectal cohorts with consistent cross-cancer performance, supplying quantitative histology signals that pair with the VLM explanations. Finally, we designed the **DQ-Former** multimodal bridge to align frozen molecular encoders with frozen LLMs via entropy-guided dynamic queries, enabling instruction following and chemically faithful rationales without expensive end-to-end fine-tuning–an integration point shared by both projects for scalable, auditable experiments.

Together, these results demonstrate steady publication in top conferences/journals and concrete system builds (agentic peptide loop, diffusion-RL generator, VLM triage, H-score at scale) that directly feed the two proposed projects and justify the requested Alliance resources.

4. Computational Methods

All pipelines are implemented in Python 3.11 with reproducible, version-pinned containers (Docker + Apptainer/Singularity), fixed CUDA/cuDNN stacks, and lockfiles (Conda/Mamba + hashed requirements.txt). Core ML uses PyTorch 2.5.x and transformers ≥4.44 with accelerate for multi-GPU launch and DeepSpeed ZeRO-3 when sharding is needed; long-context efficiency leverages flash-attention-2. Vision backbones come from timm/torchvision; tokenization uses tokenizers/sentencepiece. Datasets (all training dat are sourced from publicly available, de-identified datasets under research-use licenses) are managed with Hugging Face datasets ≥2.20 (Arrow/Parquet) and webdataset for streaming; all assets carry checksums with large binaries tracked via Git-LFS. Experiment tracking is MLflow (params/metrics/artifacts); configs are centralized with Hydra (YAML). Jobs run on Slurm (GPU & CPU partitions, array jobs, checkpoint/resume). During runs, logs/checkpoints/metrics are staged on $SCRATCH and, after completion, exported to our institutional storage; scratch is then purged.

Domain toolchains are standardized per track. Therapeutics uses RDKit 2024.x (SMILES ↔ 3D, descriptors), Open Babel, and NetworkX for graph features; diffusion/RL generators and calibrated property predictors are PyTorch modules; CPU-side docking uses AutoDock Vina / smina with receptor preparation scripted and batched via Slurm arrays (GNU Parallel where useful), employing fast pose filtering then slower rescoring for CPU efficiency. Clinical imaging uses OpenSlide (and cuCIM where available) for whole-slide tiling, scikit-image/OpenCV for color normalization and segmentation, and histology backbones (MONAI/timm) for robust cross-site features; rare-disease VLMs pair timm encoders with a frozen LLM via our bridge for HPO-grounded outputs. Each large campaign begins with 1–5% shard trial runs to validate throughput, memory, and calibration, then scales to full workloads (for example, ~10^6 molecules per target and site-stratified WSIs), directly tying engineering choices to the compute estimates in Section 5.

5. Resource Request Justification

5.1 Resource Request Summary

As shown in Table 1, our program is organized into two projects with **8 team members** in total. To execute the planned training, screening, and clinical analytics, we request **~135.01 CPU core-years** for docking/rescoring and whole-slide (WSI) processing, and **~61.0 RGU-years** on **H100-80GB or A100-40G** class GPUs for diffusion/RL training (Project 1) and VLM adaptation/inference (Project 2). For storage, our workload is engineered to be space-efficient (no WSI tiles retained; docking intermediates pruned; embeddings/libraries sharded and compressed). We therefore **do not request Alliance persistent storage** (/project or /nearline). During runs, data is staged on scratch; upon completion, the small audit bundles (configs, pruned checkpoints, compressed tables) are exported to our institutional storage and scratch is purged. We **do not require dCache or cloud resources**; all runs will execute on Nibi/Trillium/Fir using scratch and standard POSIX access.

Table 1: Summary of resource request for the two projects.

Project	**Project 1**	**Project 2**	**Total**
Team members	4	4	8
CPU (core-years)	73.64	61.37	135.01
GPU (RGU-years)	31.72	29.28	61.00
/project (TB)*	0	0	0
/nearline (TB)*	0	0	0

5.2 Compute Requests

5.2.1 Cluster Selection

We can run primarily on **Nibi, Trillium,** or **Fir** because together they cover our mix of GPU-heavy training and CPU/IO-heavy post-processing, with reliable scratch performance and mature Slurm queues. Nibi/Trillium offer general-purpose H100/A100 partitions for diffusion and VLM adaptation plus large CPU queues for docking and WSI arrays; Fir provides comparable GPU/CPU capacity for overflow and balancing queue pressure. We are not requesting Alliance persistent storage; data is staged on scratch during runs and exported to our institutional storage at completion. If queues are congested, we can spill to other Alliance clusters with similar GPU profiles using the same containers and job templates.

5.2.2 Size of Compute Request

We estimated compute by decomposing both projects into a small set of representative job profiles, then calculating totals with the Alliance spreadsheet (GPU-years → RGU-years using the H100-80GB weight 12.2). Table 2 and Table 3 show the summary of resources requested.

Table 2: Resources request for jobs NOT using GPUs.

Project	Number of cores per job	System memory per core (GB)	Number of jobs to run	Duration of each job in hours	Core-years requested
1	64	4	840	12	73.64
2	32	4	2100	8	61.37
Total					**135.01**

Table 3: Resources request for jobs using GPUs.

Project	GPU model or fraction	RGU per GPU	Number of GPUs per job	Number of CPU cores per GPU	System memory per GPU (GB)	Number of jobs to run	Duration of each job (hours)	GPU-years	RGU-years
1	H100.80GB	12.2	2	16	96	1900	6	2.60	31.72
2	H100.80GB	12.2	2	16	64	1314	8	2.40	29.28
		Total						5.00	61.00

CPU jobs (no GPUs) — total ≈ 135.01 core-years.

• Project 1 (64 cores/job, 6 GB/core, 840 jobs × 12 h → 73.64 core-years). These arrays cover (a) high-throughput docking of generated small molecules (pose generation, scoring, and fast re-ranking), (b) rescoring on top hits with slower physics-based protocols, and (c) peptide secondary-structure proxies (batch DSSP-like evaluations) and synthetic-route estimation. Jobs are embarrassingly parallel and I/O bound on large ligand/receptor grids; 8 GB/core avoids paging when loading grids and batched ligand bundles.

• Project 2 (32 cores/job, 4 GB/core, 2,100 jobs × 8 h → 61.37 core-years). These jobs process whole-slide images (WSI) for H-score: tiled reading with OpenSlide/cuCIM, color normalization, cell segmentation, stain-intensity quantification, and slide-level aggregation. We also include CPU-side feature extraction passes (where GPU is not required), plus periodic reindexing of large slide sets for multi-site evaluation. 16 cores balance parallel I/O with per-tile compute; 4 GB/core accommodates tile caches and masks.

GPU jobs (H100-80GB) — total ≈ 5.0 GPU-years ≈ 61.0 RGU-years.

• Project 1 (2×H100, 1900 jobs × 6 h → 2.60 GPU-years → 31.72 RGU-years). Each job represents a full 3D diffusion pretraining or target-conditioned run with integrated RL fine-tuning (Proximal policy optimization strategy, PPO) and calibrated property heads. Batch sizes and 3D atomistic contexts require ≥80 GB HBM; we use 2 GPUs/job to fit large contexts and speed ablations. Typical epochs include (i) diffusion training with mixed precision and flash attention, (ii) property-head calibration (drug-likeness, SA, ADMET), (iii) RL inner loops over replay buffers, and (iv) periodic checkpoint/export for downstream screening.

• Project 2 (2×H100, 1314 jobs × 8 h → 2.40 GPU-years → 29.28 RGU-years). These cover VLM adaptation for rare-disease faces and GPU-accelerated passes on WSI (feature backbones, stain-invariant encoders). A typical job cycles through (i) image encoder tuning and conformal calibration for risk-controlled top-k triage, (ii) HPO-grounding via a frozen LLM bridge (DQ-Former), (iii) large-batch inference to produce site-stratified embeddings, and (iv) replication runs for cross-site robustness. Two GPUs/job let us keep long contexts and higher-resolution crops in memory while maintaining throughput.

Why these sizes. CPU totals reflect the scale needed to (i) dock and re-score ~10^6 small-molecule candidates/target and (ii) tile/segment thousands of WSIs with repeatable quality assurance. GPU totals cover complete training cycles, ablations, and replication for both tracks without being wasteful: Project 1 is GPU-intensive (generators + RL), while Project 2 mixes moderate GPU with substantial CPU/I/O. The resulting ~135 core-years + ~61.0 RGU-years map one-to-one to the experimental plan and allow us to deliver auditable candidate libraries, calibrated clinical prototypes, and reproducible benchmarks within the allocation year.

5.2.3 Memory Requirements

Our workflows are engineered to run at or below 4 GB per CPU core, so we do not request the higher-memory surcharge. For CPU-only jobs (docking/rescoring and WSI tiling/segmentation), we stream data in shards and use memory-mapped formats (Arrow/Parquet for molecules; tile-wise I/O via OpenSlide/cuCIM for WSIs). Ligand bundles and slide tiles are batched to bounded sizes, and intermediate arrays are written to scratch to avoid resident growth. For GPU jobs, host RAM is provisioned by core count (≤4 GB/core); the heavy tensors live in GPU HBM (H100-80 GB). We therefore keep CPU RAM modest and rely on GPU memory, checkpointing, and gradient accumulation to fit models.

≤4 GB/core is sufficient as:

1. Docking/rescoring: receptor grids and ligand batches are pre-chunked; we cap concurrent ligands per process to keep RSS under 4 GB/core.
2. WSI pipeline: tiles are processed sequentially in small windows; color normalization and masks are computed per tile and flushed.
3. Model training: tensors reside on GPU; CPU holds small data loaders and host-side queues; we use mixed precision, gradient accumulation, and checkpoint-resume to bound memory.

With these practices, our largest CPU jobs remain within 4 GB/core, and GPU jobs do not rely on elevated host memory. This keeps scheduling flexible across Nibi/Trillium/Fir while minimizing footprint on shared nodes.

5.2.4 Software Details

Core ML / LLM–VLM stack (open source). We use PyTorch (2.5.x) for all training/inference because it supports GPU acceleration, mixed precision, and distributed jobs; models are launched with Hugging Face transformers (≥4.44) for LLM/VLM scaffolding and accelerate/DeepSpeed ZeRO-3 for data/model parallelism (tensor & optimizer sharding when needed). Long-context attention uses Flash-Attention 3. Vision encoders come from timm/torchvision; histology pipelines also use MONAI backbones. Classical ML components (calibrated classifiers, regressors) rely on scikit-learn/LightGBM. This stack lets us (i) train 3D diffusion generators and RL heads for molecular design, (ii) adapt a VLM for rare-disease triage, and (iii) run large-batch inference with reproducible configs. Parallelism: multi-GPU data parallel on H100 nodes; CPU-side workers for data loading and evaluation; Slurm job arrays for ablations.

Cheminformatics & docking (open source). RDKit (2024.x) is used for SMILES/3D conformer generation, descriptors, substructure queries, and fragmentation as it is reliable, has well-validated chemistry kernels and permissive license. Open Babel provides format interconversion. Structure-based screening and rescoring run with AutoDock Vina/smina on CPU nodes (why: robust search + scoring, widely cited, easy to parallelize); ligand/receptor preparation and grid generation are scripted to ensure repeatability. These tools power our small-molecule workflows (enumeration → property prediction → docking/rescoring) and our peptide pipeline (basic structure proxies), with Slurm arrays enabling embarrassingly parallel execution.

Clinical imaging, data, and orchestration (open source + standard HPC). Whole-slide image handling uses OpenSlide (and cuCIM when available) for tiling/IO; preprocessing and segmentation rely on scikit-image/OpenCV. Datasets are stored/streamed via Hugging Face datasets (≥2.20) and WebDataset (why: memory-mapped Arrow/Parquet and sharded tar for high-throughput, resumable pipelines). Reproducibility and MLOps use Docker/Apptainer for containers, Git/Git-LFS for code/large artifacts, Hydra (YAML) for centralized configs, and MLflow for metrics, artifacts, and model cards. Scheduling is through Slurm (GPU & CPU partitions, checkpoint/resume) with job arrays for large CPU workloads (WSI tiling, docking). All software is open source; no commercial licenses are required.

5.2.5 Code Performance and Utilization

GPU codes (H100-80GB). Training and fine-tuning run in PyTorch 2.5 (bf16 AMP, Flash-Attention-2). On 2×H100 we see ~220–270 optimizer steps/hour for the 3D-diffusion generator at our target batch/resolution; adding PPO-style RL heads reduces to ~170–210 steps/hour due to replay/evaluator passes. Scaling from 1→2

GPUs gives ~1.8× speed-up (≈90% efficiency) with data parallel; beyond 2–4 GPUs the marginal gain is small for our I/O pattern and checkpoint cadence, which is why the job profile uses 2 GPUs/job. VLM adaptation/inference reaches ~1.2–1.6k images/s (224–256 px crops) on 2×H100 with cut-out/augmentation; large-batch embedding sweeps are bounded by host I/O, not GPU HBM. Host RAM per GPU stays ≤64-96 GB/node; tensors reside in HBM and we keep CPU allocs ≤4 GB/core.

CPU codes. Docking/rescoring (AutoDock Vina/smina) is embarrassingly parallel: ~0.8–1.2 ligands/min/core at our exhaustiveness/seed settings; a 32-core/12-h job processes ~36k–55k ligands including pose filters and rescoring. The WSI pipeline (OpenSlide/cuCIM + scikit-image) tiles at 40–60 tiles/s on 32 cores with color normalization and cell masks; this yields ~5–8 slides/hour depending on resolution. All CPU jobs run comfortably at ≤4 GB/core. No vendor-specific instructions are required; standard nodes on Nibi/Trillium/Fir work well.

Parallel efficiency & job sizes. GPU jobs use data-parallel training (DDP) with gradient accumulation; CPU workloads use Slurm arrays with coarse-grained sharding (ligand batches; slide lists). We chose 2×H100 as the sweet spot for throughput vs. queueability and to minimize checkpoint synchronization overhead. CPU jobs are sized at 32 cores to keep per-job wall-times <12 h and to avoid saturating shared filesystems.

5.2.6 I/O Requirements

Diffusion/RL (GPU): checkpoints 5–20 GB each; keep latest 3–5 plus the final (~50–80 GB per job), logs/metrics 1–5 GB. Generated candidate libraries 20–60 GB per campaign. Retain >3 weeks on institutional storage after export; scratch is purged.

VLM (GPU): model checkpoints 10–25 GB per job (retain best-of-N); embedding sweeps 50–150 GB per site; explanations/metadata 1–5 GB. Retain >3 weeks on institutional storage.

Docking/rescoring (CPU): poses and score tables 150–300 GB per 64-core/12-h job; keep top-N summaries 5–20 GB, purge raw intermediates within 2 weeks after export.

WSI H-score (CPU): do not store tiles; features and H-score tables are 100–300 MB per slide (site-stratified). Retain >3 weeks on institutional storage.

Scratch usage and behavior: inputs and intermediates are staged on $SCRATCH, typically 200–300 GB per GPU job (checkpoints/caches) and 300–600 GB per CPU job (dock bundles or slide batches), with 2–4 TB concurrent peaks during arrays. Jobs checkpoint every 30–60 minutes (training) and at shard boundaries (docking batches; slide lists) to enable restart. After completion, finals are exported to institutional storage (checkpoints, ranked libraries, embeddings, H-score outputs) and temporary shards are deleted to keep shared filesystems clear.

5.3 Storage Resource Requests (no Alliance storage requested)

5.3.1 Storage Details

We **do not request** Alliance persistent storage (/project or /nearline). As shown in Table 4, during runs, data is staged on scratch; large intermediates (WSI tiles, docking batches, bulk embeddings) remain on scratch and are purged at job end. Only small, audit-critical bundles are exported to our institutional storage: for Project 1, a pruned fp16 checkpoint (~1.2 GB) plus configs/MLflow logs and tiny validation slices (~0.6 GB) — ≤1.8 GB total; for Project 2, a VLM adapter (LoRA/delta) checkpoint (~0.6–0.8 GB), notebooks/scripts, and small validation outputs (~1.3–1.6 GB) — ≤2.2 GB total. These GB-scale bundles are sufficient for auditability and replication; anything larger is recomputed as needed.

Table 4: Storage request details for each project.

Project	What we keep (examples)	Size (approx.)	Where	Retention & note
P1	Current best model checkpoint (compressed), run configs, MLflow logs	1.8 GB	Institutional storage	Needed for live jobs
P1	Top-N candidate list (SMILES+scores, compressed Parquet/CSV), final checkpoint	≤8 GB	Institutional storage	≥12 mo; full libraries re-generated if needed
P2	Active H-score scripts, small validation slices, current report notebooks	2.2 GB	Institutional storage	For in-progress analysis only
P2	Final H-score tables (compressed), VLM summary stats (no raw tiles/embeddings)	≤8 GB	Institutional storage	≥12 mo; tiles/embeddings recomputed on demand

5.3.2 Storage Performance and Utilization

Seasonal utilization. We **do not request** /project or /nearline. Throughout the year, data is staged on scratch during runs and exported to our institutional storage at job completion. Usage ramps from pilot runs (Jan–Mar) to a peak in April–September when diffusion/VLM jobs and CPU arrays overlap; we then consolidate in Oct–Dec by purging intermediates from scratch after export. Persistent artifacts are GB-scale bundles only (configs, pruned checkpoints, compressed tables).

Table 5: Summary of storage performance and utilization.

Workload	Access pattern	Typical per-job I/O	Aggregate peaks	Where it lives during runs	After job completes
GPU training / checkpoints	Large sequential writes (periodic)	100–200 MB/s; checkpoint 5–20 GB every 30–60 min	Low (few jobs)	**Scratch** for bulk writes	Export pruned fp16/LoRA checkpoint + logs to institutional storage; purge rest
Docking arrays (CPU)	Large sequential reads + small writes	200–400 MB/s per node; modest IOPS (hundreds)	Up to **1–2 GB/s** per project in bursts	**Scratch** for shard bundles	Export compressed top-N tables; purge intermediates
WSI pipeline (CPU)	Sequential tile reads; batched feature writes	200–400 MB/s per node	Similar to docking (bursty)	**Scratch** for tiles/features	Export H-score tables (compressed); tiles not stored
Finalized artifacts	Infrequent reads	N/A	N/A	N/A	Institutional storage only (GB-scale bundles); no Alliance persistent storage

5.5 Description of High Demand Periods

Our peak usage is April–September, when diffusion pretraining/target-conditioned reruns and VLM adaptation run in parallel–driving sustained GPU demand (H100 jobs with frequent checkpoints) and overlapping CPU arrays for docking and WSI processing. CPU/IO load also spikes after major checkpoints as we launch large screening/evaluation batches. To reduce contention, we stagger array starts, shift jobs across Nibi/Trillium/Fir based on queue pressure and data locality, and once runs finish, export audit bundles to our institutional storage and purge scratch so the shared filesystems remain clear.

Resource Management and Computational Expertise

6. Funding Available to Use Advanced Research Computing Resources

Our program is supported by two CIHR project grants with a combined total award of CAD $2,497,102. These funds ensure sustained access to advanced research computing by covering personnel time to operate pipelines, data curation, and ancillary costs related to compute, storage, and reproducibility (Table 6).

Table 6: Summary of funding available for our projects.

Project	Funding source	Grant	Grant start date	Grant expiry date	Total grant award	Portion allocated for computational project(s)
Project 1	CIHR	2025 Spring Project Grants	2025-09-01	2030-08-30	CAD $1,067,176	CAD $300,000
Project 1	CIHR	Tier 2 Canada Research Chair Award	2022-09-01	2027-08-30	CAD $600,000	CAD $200,000
Project 2	CIHR	2023 Spring Project Grants	2023-09-01	2028-08-30	CAD $830,026	CAD $200,000
	Total	3 grants			CAD $2,497,102	CAD $700,000

7. Computational Expertise of the Team

Our lab has been running large-scale workloads on Alliance systems for two years, with sustained use of NVIDIA GPUs (A100/H100), Slurm job arrays, and containerized environments (Apptainer) for fully reproducible runs. The Ph.D. student, Zihao Jing, leads day-to-day operations, and every team member is trained to submit, monitor, and checkpoint jobs; we routinely coordinate queue-aware scheduling, data hygiene, and failure-safe restarts. On the software side we are comfortable with PyTorch/Transformers, Flash-Attention, CUDA Toolkit builds, mixed precision (bf16/fp16), and profile/optimize I/O for CPU-heavy arrays. Knowledge is shared through common job templates, internal docs, and weekly ops reviews so new staff quickly reach production competency. In short, the group already operates at scale on Alliance hardware and is prepared to use the requested CPU/GPU/storage efficiently from day one (Table 7).

Table 7: Summary of computational expertise of our team.

Required Expertise	Description of internal or planned resources
HPC operations & scheduling (Slurm)	Lab has **2 years on Alliance**; shared Slurm array templates, checkpoint/restart, queue-aware scheduling across Nibi/Trillium/Fir; every member trained to submit/monitor jobs.
GPU training & optimization (NVIDIA A100/H100)	PyTorch/Transformers with **CUDA Toolkit**, **Flash-Attention**, AMP (bf16/fp16); profiling and memory tuning on multi-GPU; stable throughput on long runs.
Containers & reproducibility	Version-pinned **Apptainer/Docker** images; consistent environments across clusters; Standard Operating Procedure/ Quality Assurance for data hygiene and run manifests.

Data engineering & I/O	Arrow/Parquet & WebDataset sharding; staged I/O to scratch; scalable CPU arrays for docking/WSI; metadata and checksum policies to avoid tiny-file overhead.
Cheminformatics & clinical imaging pipelines	**RDKit/Vina/smina** for molecules; **OpenSlide/cuCIM** + scikit-image for WSI; parameterized scripts for ligands/tiles; reproducible, site-portable workflows.
MLOps & monitoring	GitHub Actions CI (lint/tests/container builds), MLflow logging, automated smoke tests before scaling; weekly ops review to share know-how and resolve failures quickly.

8. Management Strategy

We plan and operate strictly under shared-cluster constraints (fair-share, queues, wall-time limits). Zihao Jing will track a rolling RGU-year burn-rate and CPU core-years, approves large submissions, and shifts work across Nibi / Trillium / Fir based on queue pressure and data locality. All jobs use containerized environments (Apptainer) and common Slurm templates with checkpoint/restart, so runs can be paused, resubmitted, or migrated without data loss. We size GPU jobs to be scheduler-friendly (typical profile: 2×H100-80GB per job; fallbacks to H100/A100 or MIG fractions for smaller fine-tunes), keep wall-times modest (≤20 days for training; ≤12 h for CPU arrays), and stagger large arrays to avoid filesystem contention. Weekly ops stand-ups review queue stats, failure reports, and scratch usage/cleanup (stage on scratch → export to our institutional storage), and we backfill with short jobs when heavier work is waiting (Table 8&Table 9).

Table 8: Management strategy for project 1 – Multimodal generative therapeutics engine.

Full Name	Position	CPU (core-years)	GPU (RGU-years)	/project (TB)	/nearline (TB)
Zihao Jing	PhD Student	18.00	18.00	0	0
Morgan He	PhD Student	5.60	4.00	0	0
Gen Zhou	PhD Student	10.00	7.62	0	0
Yan Yi Li	PhD Student	40.00	2.10	0	0
Totals	4 team members	73.60	31.72	0	0

Table 9: Management strategy for project 2 – Interpretable clinical vision intelligence.

Full Name	Position	CPU (core-years)	GPU (RGU-years)	/project (TB)	/nearline (TB)
Kathy Feng	PhD Student	23.37	14.00	0	0
Kyle Chen	PhD Student	24.00	8.00	0	0
Yuxi Long	PhD Student	6.00	4.58	0	0
Yan Sun	PhD Student	8.00	2.70	0	0
Totals	4 team members	61.37	29.28	0	0

Each student has authority for their slice of CPU/GPU and storage; jobs follow a **checklist** (data ready → smoke test → array/train → checkpointing → promotion/pruning). The PI balances utilization across clusters and reassigns capacity if bottlenecks appear, keeping GPU occupancy high while avoiding filesystem contention.

Role & resource envelopes (not reservations; we flex based on queues)

Table 10: Role and resource envelopes for project 1.

Member (role)	Resource envelope & duties
Zihao Jing (recourse manager)	Approves big runs; moves work between clusters; enforces burn-rate; handles preemptions/requeues.
Morgan He (diffusion model lead)	Launches 2×H100 training/ablations; can split to 1×H100 or MIG when queues are tight.
Gen Zhou (peptide RL)	Runs shorter GPU cycles and evaluator bursts; backfills small GPU slots.
Yan Yi Li (docking/rescoring)	Manages large CPU arrays; staggers starts; purges intermediates; promotes top-N only.

Table 11: Role and resource envelopes for project 2.

Member (role)	Resource envelope & duties
Kathy Feng (VLM + HPO)	Uses 2×H100 for adaptation/inference; falls back to A100/MIG when needed; strict checkpointing.
Kyle Chen (pathology / H-score)	Runs WSI CPU arrays (≤12 h shards); throttles concurrency to match I/O capacity.
Yuxi Long (MLOps / data)	Maintains containers/CI, dataset shards, and job health dashboards; handles requeues.
Yan Sun (sequence / eval support)	Small GPU/CPU jobs for metrics and replication; ideal for backfilling.

In Table 10 and Table 11, quotas shown elsewhere (CPU core-years, RGU-years, storage) are annual totals; team members draw from the same pool and adjust in real time to queue availability. This approach (small, checkpointed jobs; flexible GPU sizing; multi-site spillover) has been our practice on Alliance for the past two years, yielding high utilization without blocking other users.