Learning Guide: Advanced

HCLS AI Factory

Graduate / Professional Level

Deep technical analysis of the HCLS AI Factory architecture, from BWA-MEM2 seed-and-extend algorithms through diffusion-based molecular docking, with emphasis on algorithmic design decisions, scaling bottlenecks, and clinical translation barriers.

NVIDIA DGX Spark | Parabricks | BioNeMo | Milvus | Claude

02/2026 | Version 1.0 | Apache 2.0 License

Author: Adam Jones

Table of Contents

Chapter 1: Computational Genomics — From FASTQ to VCF

1.1 Sequencing Data Characteristics

The HCLS AI Factory processes Illumina short-read data: 2×250 bp paired-end reads from 30× whole-genome sequencing of HG002 (NA24385), a GIAB Ashkenazi Jewish reference standard. The FASTQ files total approximately 200 GB and contain ~800 million read pairs.

Why HG002?

The Genome in a Bottle (GIAB) Consortium provides extensively validated truth sets for HG002, enabling rigorous benchmarking. The high-confidence regions cover >95% of the GRCh38 reference, with variant calls validated by multiple orthogonal technologies (PacBio HiFi, Oxford Nanopore, Hi-C, optical mapping).

1.2 GPU-Accelerated Alignment: BWA-MEM2 on Parabricks

NVIDIA Parabricks 4.6.0-1 (container: nvcr.io/nvidia/clara/clara-parabricks:4.6.0-1) provides a GPU-accelerated implementation of BWA-MEM2.

Algorithm Overview

BWA-MEM2 uses a seed-and-extend approach:

1. **Seeding:** Extract fixed-length k-mers from the query read and look them up in the FM-index of the reference genome
2. **Chaining:** Group collinear seeds into chains representing candidate alignment locations
3. **Extension:** Perform Smith-Waterman local alignment around each chain to produce the final alignment
4. **Scoring:** Select the best alignment and assign a MAPQ (mapping quality) score

GPU Acceleration Strategy

Parabricks parallelizes the computationally intensive Smith-Waterman extension step across GPU cores. The FM-index lookup (seeding) remains CPU-bound but constitutes a small fraction of total compute. The fq2bam command also integrates coordinate sorting and duplicate marking, eliminating separate samtools sort and picard MarkDuplicates steps.

Performance on DGX Spark (GB10)

Metric	Value
Wall time	20-45 minutes
GPU utilization	70-90%

Peak memory	~40 GB (of 128 GB unified)
Output	Sorted BAM + BAI index
Mapping rate	>99.5%
Duplicate rate	~8-12%

1.3 Deep Learning Variant Calling: DeepVariant

Google DeepVariant reframes variant calling as an image classification problem. For each candidate variant site, it constructs a pileup image — a visual representation of aligned reads at that position — and classifies it using a convolutional neural network (CNN).

Architecture Details

- **Input:** Pileup image (channels: read bases, base qualities, mapping qualities, strand, etc.)
- **Network:** Inception-v3 CNN architecture
- **Output:** Three-class softmax (homozygous reference, heterozygous variant, homozygous variant)
- **Training:** Supervised on GIAB truth sets, with data augmentation and hard example mining

Why DeepVariant Outperforms GATK HaplotypeCaller

1. The CNN learns complex error patterns that statistical models cannot capture
2. No explicit error model required — the network learns directly from data
3. Better performance on indels and complex variants
4. Transferable across sequencing platforms (Illumina, PacBio, ONT)

Performance

Metric	Value
Wall time	10-35 minutes (GPU-accelerated via Parabricks)
GPU utilization	80-95%
Peak memory	~60 GB
SNP F1	>99.7% on HG002
Indel F1	>99.4% on HG002
Total variants	~11.7M (unfiltered)
QUAL>30 variants	~3.5M

1.4 VCF Quality Metrics

Metric	Expected Range	Interpretation

Ti/Tv ratio	2.0-2.1	Transition/transversion ratio; deviation suggests systematic error
Het/Hom ratio	1.5-2.0	Heterozygous/homozygous ratio; population-dependent
SNP count	~4.2M	Consistent with Ashkenazi ancestry
Indel count	~1.0M	Normal range for WGS
Novel variant rate	<5%	Variants not in dbSNP; higher rates suggest error

Chapter 2: Variant Annotation — Multi-Database Integration

2.1 ClinVar: Clinical Variant Classification

ClinVar (NCBI) is a freely accessible archive of relationships between human variants and phenotypes. The HCLS AI Factory integrates the February 2026 release containing 4.1 million variant-condition records.

Classification System (ACMG/AMP)

- **Pathogenic (P) —** Strong evidence of disease causation
- **Likely Pathogenic (LP) —** Moderate evidence
- **Variant of Uncertain Significance (VUS) —** Insufficient evidence
- **Likely Benign (LB) —** Moderate evidence against pathogenicity
- **Benign (B) —** Strong evidence against pathogenicity

Review Status Tiers

ClinVar classifies assertion confidence using star ratings (0-4 stars). The pipeline weights variants with ≥2 stars (multiple submitters with concordant interpretations) more heavily.

Annotation Performance

Of ~3.5M QUAL>30 variants, approximately 35,616 (1.0%) match ClinVar entries. The low match rate reflects that most variants in a healthy individual are common polymorphisms not represented in a clinical database focused on rare disease.

2.2 AlphaMissense: AI Pathogenicity Prediction

AlphaMissense (Cheng et al., Science 2023) predicts the pathogenicity of all possible human missense variants using features derived from AlphaFold protein structure predictions and evolutionary conservation.

Model Architecture

- **Input features:** amino acid sequence context, evolutionary conservation (from MSA), and structural features from AlphaFold
- **Output:** pathogenicity score (0-1, continuous)
- **Total predictions:** 71,697,560 unique missense variants

Calibrated Thresholds

- **Pathogenic:** >0.564 (90% precision on ClinVar pathogenic set)
- **Ambiguous:** 0.34-0.564
- **Benign:** <0.34 (90% precision on ClinVar benign set)

Critical Limitation

AlphaMissense only predicts missense variant effects. Stop-gain, frameshift, splice site, and non-coding variants require other prediction tools. The pipeline uses VEP for functional consequence annotation to complement AlphaMissense.

2.3 Ensembl VEP: Functional Consequence Prediction

The Variant Effect Predictor maps variants to genes, transcripts, and regulatory regions, annotating each with standardized Sequence Ontology (SO) terms.

Impact Classification

Impact Level	Example Consequences	Typical Action
HIGH	stop_gained, frameshift_variant, splice_donor_variant	Likely loss of function
MODERATE	missense_variant, inframe_deletion	Protein function may change
LOW	synonymous_variant, splice_region_variant	Unlikely to affect protein
MODIFIER	intron_variant, upstream_gene_variant	Non-coding effects

2.4 The Annotation Pipeline Architecture

The three annotation databases are applied sequentially in annotator.py (23 KB):

Annotation Pipeline Flow

```
VCF (11.7M variants)
  → parse_vcf(min_qual=30)     → 3.5M variants
  → annotate_clinvar()          → Clinical significance
  → annotate_alphamissense()    → AI pathogenicity scores
  → annotate_vep()              → Functional consequences
  → generate_text_summary()     → Natural language descriptions
  → embed_variants()            → 384-dim BGE embeddings
  → index_in_milvus()           → Searchable vector database
```

Chapter 3: Vector Database Architecture — Milvus and RAG

3.1 Milvus Schema Design

The genomic_evidence collection in Milvus 2.4 uses a 17-field schema designed to support both vector similarity search and scalar filtering:

Field	Type	Rationale
id	INT64 (PK, auto)	Milvus-managed primary key
embedding	FLOAT_VECTOR(384)	Semantic search vector
chrom	VARCHAR(10)	Genomic coordinate filtering
pos	INT64	Positional queries
ref/alt	VARCHAR(1000)	Allele matching
qual	FLOAT	Quality score filtering
gene	VARCHAR(100)	Gene-level queries
consequence	VARCHAR(200)	Functional filtering (e.g., missense only)
impact	VARCHAR(20)	Impact level filtering
genotype	VARCHAR(10)	Zygosity queries
text_summary	VARCHAR(2000)	Human-readable context for RAG
clinical_significance	VARCHAR(200)	ClinVar classification
rsid	VARCHAR(20)	dbSNP lookup
disease_associations	VARCHAR(2000)	Disease context for RAG
am_pathogenicity	FLOAT	AlphaMissense score filtering
am_class	VARCHAR(20)	Pathogenicity class filtering

3.2 Index Configuration and Performance

Index Type: IVF_FLAT (Inverted File with Flat Vectors)

Why IVF_FLAT? At 3.5M vectors with 384 dimensions, IVF_FLAT provides the best recall-latency tradeoff. HNSW would use more memory; IVF_PQ would sacrifice recall.

nlist=1024: Partitions vectors into 1024 clusters. Query searches ~16 clusters (nprobe=16), examining ~55K vectors per query.

Metric: COSINE similarity (normalized dot product)

Search Performance

Metric	Value
Index build time	~8 minutes (3.5M × 384-dim)

Index memory	~2 GB
Search latency (nprobe=16)	8-15 ms
Recall@20	>95%

3.3 RAG Architecture with Claude

The RAG pipeline in rag_engine.py (23 KB) implements a multi-stage retrieval strategy:

1. Query Expansion

User queries are enriched using 10 therapeutic area keyword maps. For example, a query about "neurodegeneration" is expanded with terms like "frontotemporal dementia," "ALS," "motor neuron," "tau protein."

2. Hybrid Retrieval

The expanded query is embedded and used for vector search (top_k=20). Results are optionally filtered by scalar fields (e.g., impact=HIGH, am_class=pathogenic).

3. Context Assembly

Retrieved variants are formatted into structured context:

Context Template

```
## Variant Evidence
- chr9:35065263 G>A | Gene: VCP | Consequence: missense_variant
  ClinVar: Pathogenic | AlphaMissense: 0.87 (pathogenic)
  Disease: Frontotemporal Dementia, ALS, IBMPFD
```

4. Claude Inference

The assembled context + knowledge base + user query are sent to claude-sonnet-4-20250514 (temperature=0.3, max_tokens=4096).

Why temperature=0.3?

Lower temperature produces more deterministic, factual responses. For clinical genomics, hallucination is dangerous — the model should report only what the evidence supports.

Chapter 4: Drug Discovery Pipeline — Deep Dive

4.1 The 10-Stage Architecture

The drug discovery pipeline in pipeline.py (18 KB) implements a sequential 10-stage workflow:

Stage	Module	Key Algorithm
1. Initialize	pipeline.py	Pydantic model validation
2. Normalize Target	pipeline.py	Gene → UniProt → PDB mapping
3. Structure Discovery	cryoem_evidence.py	RCSB PDB REST API query
4. Structure Preparation	cryoem_evidence.py	Multi-factor scoring
5. Molecule Generation	nim_clients.py	MolMIM masked LM inference
6. Chemistry QC	molecule_generator.py	RDKit valence/kekulization
7. Conformer Generation	molecule_generator.py	RDKit ETKDG algorithm
8. Molecular Docking	nim_clients.py	DiffDock diffusion inference
9. Composite Ranking	pipeline.py	Weighted multi-objective
10. Reporting	pipeline.py	ReportLab PDF generation

4.2 Cryo-EM Structure Scoring

The cryoem_evidence.py (6 KB) module implements a multi-factor structure scoring algorithm:

Python
```
score += max(0, 5.0 - resolution)              # Resolution: 0-5 scale
if has_inhibitor_bound: score += 3.0             # Binding site defined
score += num_druggable_pockets * 0.5             # Pocket count bonus
if 'Cryo-EM' in method: score += 0.5              # Method bonus
```

Design Rationale

Resolution: the primary factor (0-5 scale). The 5 Å cutoff excludes low-resolution structures unsuitable for docking.

Inhibitor bonus (+3): Inhibitor-bound structures provide a pre-defined binding site and reference ligand geometry.

Pocket count (+0.5 each): More druggable pockets increase therapeutic options.

Cryo-EM bonus (+0.5): Reflects the growing prevalence and quality of Cryo-EM structures for drug targets.

4.3 MolMIM: Molecular Masked Inverse Modeling

MolMIM applies masked language modeling (the technique behind BERT in NLP) to molecular SMILES strings. Given a seed molecule, it:

1. Tokenizes the SMILES into a vocabulary of molecular substructures
2. Randomly masks 15-30% of tokens
3. Predicts the masked tokens using a transformer architecture
4. The predicted tokens create novel molecular structures

Critical Considerations

SMILES output: MolMIM generates SMILES strings, not 3D structures. Chemical validity must be verified by RDKit.

Stochastic generation: Different random seeds produce different molecules.

Temperature control: Higher temperature = more diverse but potentially less valid molecules.

4.4 DiffDock: Diffusion-Based Molecular Docking

DiffDock (Corso et al., ICLR 2023) models molecular docking as a generative diffusion process over the product space of rotations, translations, and torsion angles.

Key Innovation

Unlike grid-based docking methods (AutoDock Vina, Glide), DiffDock does not require a pre-defined search box around a binding site. It learns to predict binding poses directly from protein-ligand pairs, making it suitable for blind docking.

Score Interpretation

Confidence score (0-1): indicates the model's certainty about the predicted pose

Binding affinity (kcal/mol): estimates the free energy of binding; more negative = stronger binding

Limitations

Training bias: DiffDock was trained primarily on crystal structures; performance may degrade on Cryo-EM structures with lower resolution

No kinetics: The model predicts pose and affinity but not binding kinetics (on/off rates)

Rigid protein: Protein flexibility is not modeled — the protein is treated as rigid

4.5 Composite Scoring and Normalization

The composite scoring formula balances three objectives:

Python
```python
dock_normalized = max(0.0, min(1.0, (10.0 + dock_score) / 20.0))
composite = 0.30 * gen_score + 0.40 * dock_normalized + 0.30 * qed_score
```

Normalization Rationale

Docking scores: range from ~-15 to ~0 kcal/mol. The formula (10 + dock) / 20 maps this to approximately 0-1, with -10 kcal/mol mapping to 0.0 and +10 mapping to 1.0.

Generation scores: already 0-1 (MolMIM confidence).

QED scores: inherently 0-1.

Weight Rationale

Docking (40%): receives the highest weight because binding affinity is the most direct predictor of therapeutic activity

Generation (30%): balances novelty of the molecular design

QED (30%): balances practical drug-likeness

Chapter 5: Nextflow DSL2 Pipeline Architecture

5.1 Module Design

The pipeline uses Nextflow DSL2's module system for composable workflow design:

Directory Structure

```
hls-orchestrator/
├── main.nf                 # Entry point, mode routing
├── nextflow.config         # Profiles, parameters
├── run_pipeline.py         # Python CLI launcher
└── modules/
    ├── genomics.nf         # Stage 1 processes
    ├── rag_chat.nf         # Stage 2 processes
    ├── drug_discovery.nf   # Stage 3 processes
    └── reporting.nf        # Report generation
```

5.2 Execution Modes and Data Flow

Mode	Data Flow	Use Case
full	FASTQ → VCF → Target → Candidates	Complete pipeline
target	VCF → Target → Candidates	Pre-existing VCF
drug	Target → Candidates	Known gene target
demo	Pre-configured FASTQ → Candidates	VCP/FTD demonstration
genomics_only	FASTQ → VCF	Variant calling only

5.3 Profile Configuration

The nextflow.config defines six execution profiles optimized for different environments:

dgx_spark: GPU resource requests, memory limits tuned for 128 GB unified memory

docker: Docker container execution with GPU passthrough

singularity: Singularity containers for HPC environments without Docker

slurm: SLURM scheduler integration for cluster execution

Chapter 6: Clinical Translation and Limitations

6.1 From Computational Hits to Drug Leads

The HCLS AI Factory generates computational drug candidates — not approved medications. The path from computational hit to clinical drug requires:

1. **In vitro validation:** Test top candidates in biochemical assays (e.g., VCP ATPase activity inhibition)
2. **Cell-based assays:** Confirm activity in relevant cell lines
3. **ADMET profiling:** Absorption, Distribution, Metabolism, Excretion, and Toxicity studies
4. **Lead optimization:** Iterative cycles of design, synthesis, and testing
5. **Preclinical studies:** Animal models for efficacy and safety
6. **Clinical trials:** Phase I (safety), Phase II (efficacy), Phase III (large-scale)

Estimated Timeline

10-15 years from computational hit to approved drug. The HCLS AI Factory accelerates the earliest stage — computational lead generation — from months to minutes.

6.2 Limitations and Caveats

Genomics

- DeepVariant accuracy varies by variant type (SNPs > indels > structural variants)
- Short-read WGS has limited sensitivity for structural variants and repeat expansions
- Population-specific biases in GRCh38 may affect variant calling in non-European ancestries

RAG/Annotation

- ClinVar has known biases toward well-studied genes and European ancestry variants
- AlphaMissense is limited to missense variants; non-coding variants are not scored
- The 201-gene knowledge base covers common drug targets but not the full druggable genome

Drug Discovery

- MolMIM-generated molecules have not been synthesized or tested
- DiffDock docking scores are predictions, not experimental measurements
- Protein flexibility is not modeled; induced-fit effects are ignored
- The composite scoring weights (30/40/30) are heuristic, not optimized on clinical outcomes

6.3 Ethical Considerations

Informed consent: Patient genomic data requires explicit consent for research use

Data sovereignty: NVIDIA FLARE federated learning keeps data local; essential for HIPAA/GDPR compliance

Return of results: Incidental findings (e.g., BRCA1 pathogenic variants) may require clinical reporting

Equity: Pipeline performance should be validated across diverse ancestries to avoid exacerbating health disparities

Chapter 7: Scaling Analysis

7.1 DGX Spark Bottleneck Analysis

Component	Bottleneck	Phase 1 Impact
Parabricks (fq2bam)	GPU compute	20-45 min, acceptable
DeepVariant	GPU memory (60 GB peak)	Leaves 68 GB for other tasks
Milvus indexing	CPU + I/O	24 min for 3.5M vectors
MolMIM inference	GPU compute	2 min for 100 molecules
DiffDock inference	GPU compute + memory	8 min for 98 candidates
Sequential total	GPU time-sharing	~4 hours end-to-end

7.2 Phase 2: DGX B200 Scaling

With 8× B200 GPUs and 1-2 TB HBM3e:

- **Parallel Parabricks:** 4-8 simultaneous samples
- **Dedicated Milvus GPU:** GPU-accelerated vector search (sub-millisecond)
- **NIM replicas:** 2-4 MolMIM + 2-4 DiffDock instances
- **Estimated throughput:** 10-20 patients per day

7.3 Phase 3: DGX SuperPOD

- **Hundreds of B200 GPUs** with NVLink and InfiniBand
- **Distributed Milvus cluster:** Billions of variants across institutions
- **NVIDIA FLARE:** Federated model training without data sharing
- **Estimated throughput:** Thousands of patients per day

Chapter 8: Advanced Topics and Extensions

8.1 Alternative Embedding Strategies

BGE-small-en-v1.5 (384-dim) was chosen for its balance of quality and efficiency. Alternatives:

Model	Dimensions	Size	Trade-off
BGE-small-en-v1.5	384	33M params	Current choice: fast, efficient
BGE-base-en-v1.5	768	109M params	Better recall, 2× memory
BGE-large-en-v1.5	1024	335M params	Best recall, 3× memory
BiomedBERT	768	109M params	Domain-specific, biomedical text
PubMedBERT	768	109M params	PubMed-trained, clinical text

8.2 Multi-Objective Optimization

The current composite scoring uses fixed weights (30/40/30). Advanced approaches:

Pareto optimization: Identify the Pareto frontier of generation, docking, and QED
Bayesian optimization: Learn optimal weights from experimental feedback
Active learning: Prioritize candidates that reduce uncertainty in the scoring model

8.3 Long-Read Sequencing Integration

Oxford Nanopore and PacBio long-read technologies can detect structural variants (SVs) and repeat expansions that short-read WGS misses. Future extensions could:

- Add ONT/PacBio alignment with minimap2
- Detect SVs with Sniffles2 or PEPPER-Margin-DeepVariant
- Phase haplotypes for compound heterozygosity detection

8.4 Pharmacogenomics Integration

The knowledge base includes 11 pharmacogenomics genes (CYP2D6, CYP2C19, CYP3A4, DPYD, TPMT, etc.). Future extensions could:

- Star allele calling with PharmCAT
- Drug-drug interaction prediction
- Dosing recommendations based on metabolizer status

Review Questions

Graduate-level questions for self-assessment and discussion. These require synthesis across multiple chapters.

1. Explain why DeepVariant reframes variant calling as an image classification problem. What advantages does this provide over statistical models like GATK HaplotypeCaller?

2. Describe the IVF_FLAT index configuration (nlist=1024, nprobe=16) and calculate the approximate number of vectors examined per query from a collection of 3.5M vectors.

3. Why does the RAG pipeline use temperature=0.3 for Claude? What are the trade-offs of lower vs. higher temperature in clinical genomics applications?

4. Explain the docking score normalization formula max(0, min(1, (10 + dock_score) / 20)). What docking score maps to 0.5? Why is this mapping appropriate?

5. Compare the AlphaMissense thresholds (pathogenic >0.564, benign <0.34) with ClinVar classifications. What does the "ambiguous" zone represent, and why is it clinically significant?

6. Describe three limitations of DiffDock that could affect the reliability of the drug candidate rankings.

7. Explain why inhibitor-bound PDB structures (like 5FTK) receive a +3 bonus in the structure scoring algorithm. What information does an inhibitor-bound structure provide that an apo structure does not?

8. Design a validation experiment to test the top 5 drug candidates from the VCP/FTD demo pipeline. What assays would you use, and what would constitute a positive result?

9. Calculate the approximate memory budget for the Milvus vector index: 3.5M vectors × 384 dimensions × 4 bytes per float. How does this compare to the available memory on DGX Spark?

10. The composite scoring weights (30% generation, 40% docking, 30% QED) are heuristic. Propose an approach to optimize these weights using experimental feedback from in vitro screening.