Roman Knyazhitskiy

cv@knyaz.tech — knyaz.tech — knyazer

Summary

Education

MPhil in MLMI (Machine Learning), University of Cambridge 10/2025 - 09/2026

BSc Computer Science and Engineering, TU Delft 09/2022 - 07/2025

- GPA: 8.7/10. Cum Laude + Honours.

Work Experience

Machine Learning Engineer, Delft Mercurians 05/2023 - 09/2025

- Led a team of 2-5 engineers in developing AI control systems for RoboCup competition.
- Designed and implemented a Model Predictive Control (MPC) system for real-time trajectory optimization.
- Developed PyQt interface for Kalman filter calibration with protobuf data visualization.
- Integrated Python-based models into Rust codebase and worked on communication protocols.
- Built JAX+Equinox continuous time differentiable simulator tailored for robotic systems.
- Contributed to open-source JAX-based ML libraries during the project.

Research Associate, TU Delft 03/2023 - 08/2025

- Worked on several AutoML and Meta Learning projects under supervision of Professor Tom Viering.
- Researched LLM applications in software engineering, including vulnerability detection and code generation.
- Investigated Prior-Data Fitted Networks (PFNs) and MCMC acceleration methods.
- Delivered talks on optimization methods including Adam's L_∞ norm properties and Shampoo/Muon spectral characteristics.
- Participated in departmental research meetings and academic discussions on diffusion models.

Applied Machine Learning Intern, Central Robotics Institute 06/2020 - 07/2020

- Developed image-to-line conversion tool for robotic drawing applications.
- Implemented custom image segmentation and processing logic for line extraction.
- Created algorithms to identify optimal drawing paths based on image intensity analysis.

Publications

[1] J. Luijmes, A. Gielisse, R. Knyazhitskiy, and J. van Gemert. ARC: Anchored representation clouds for high-resolution INR classification. In *ICLR 2025 Workshop on Weight Space Learning*, 2025. Accepted.

[2] R. Knyazhitskiy and P. R. Van der Vaart. A simple scaling model for bootstrapped DQN. 2025. Under review.

Honours and Awards

- **1st Place**, Bunq Hackathon 6 (2025) – Team of 4 against 50+ teams, €30,000 prize.
- **2nd Place & Special Prize**, Epoch AI Hackathon (2024).
- **Best Software Solution**, RoboCup World Championships, Sydney (2019).

- **1st Place**, RoboCup Junior National Competitions (2017, 2018, 2019).
- **Silver Medal**, AIIJC International AI Competition for Juniors – Sign language recognition application.
- **Winner**, International High School Astrophysics Competition.
- Winner of numerous regional robotics, physics, mathematics, competitive programming, and astronomy competitions.

Open Source Contributions

- Enhanced functionalities in jaxtyping and Equinox, resolving multiple issues and enabling IPython runtime type checking.
- Contributed to Gymnax, a widely used JAX RL environments collection with 800+ stars on GitHub.
- Contributed to tooling around beartype, building most of the beartype pytest plugin and other tools around it.
- Improved libccd (collision detection library in C++) fixing a critical corner-case with an infinite loop.

Selected Projects

Weather forecasting and option trading system, Python, JAX/Equinox, Machine Learning 2025

- Developed an end-to-end algorithmic trading system for Polymarket weather prediction markets, integrating real-time meteorological data processing, probabilistic machine learning models, and automated trading strategies. The system processes multiple weather data streams (METAR minutely observations, NBM hourly forecasts, ICON/ECMWF ensemble forecasts) through an event-driven architecture with 5,500 lines of Python code. Core ML components include: (1) Continuous Normalizing Flow (CNF) model using neural ODEs (diffrax) for uncertainty-calibrated temperature predictions with weighted likelihood training, (2) Direct Max ensemble predictor with calibration diagrams and temperature-band embeddings for classification tasks. Both models feature sophisticated preprocessing pipelines handling relative temperature representations, missing data imputation, temporal embeddings, and extensive data augmentation (correlated noise injection, temporal shifting, adjacent swapping). Implemented frontier-based decision framework using epistemic uncertainty bounds (upper/lower probability intervals) for risk-aware position sizing. Trading simulation engine evaluates strategies through historical backtesting with order book liquidity constraints and Sharpe ratio metrics. System architecture includes hot-reloadable trader modules, asynchronous data polling (aiohttp), event logging with replay capability, XMPP notifications, and Polymarket API integration. Leverages JAX's XLA compilation and hardware acceleration for efficient training and inference. Models trained using AdamW with cosine annealing, dropout regularization, and W&B experiment tracking. Demonstrates practical application of generative models, probabilistic forecasting, financial risk management, and production ML systems design.

Lyapunov-Stabilized Truncated Backpropagation Through Time, Python, JAX, Equinox, Brax 2024

- Implemented a reinforcement learning framework for continuous control tasks using truncated backpropagation through time with Lyapunov stability factors to stabilize gradient flow in long-horizon sequential decision-making. The system employs an actor-critic architecture with log-normal policy parameterization trained on physics-based locomotion environments. Core algorithmic components include gradient truncation with configurable window sizes (50-100 steps), Lyapunov multipliers (0.88-0.92) applied to state transitions for controlled gradient propagation, adaptive observation normalization using Welford's online algorithm for running mean/variance estimation, and cosine annealing learning rate schedules with AdamW optimization. The architecture features modular design with separate components

for policy networks (3-layer MLPs with 64 hidden units and tanh activations), probability distributions (diagonal Gaussian with learnable log-standard deviations), state transitions, and evaluation metrics. Implemented gradient clipping, weight decay regularization, and careful initialization (0.1x scaling for output layers) to ensure training stability. The system supports multi-host distributed training through JAX's vmap for parallel hyperparameter sweeps across Lyapunov factors, automatic differentiation with custom gradient stopping points for truncation, and JIT compilation for 10x+ performance improvements. Integrated WandB experiment tracking with configurable logging intervals. Tested on Brax's Ant locomotion task with 1000-step episodes across 40 parallel environments, achieving stable policy learning through careful gradient flow management. Demonstrates practical application of automatic differentiation, functional programming patterns in JAX/Equinox, and stability techniques for recurrent/sequential gradient computation.

pytest-mut: Mutation Testing Framework, Python, pytest, AST 2024

- Developed a mutation testing plugin for pytest that systematically evaluates test suite quality by introducing code mutations and verifying test failure rates. The system uses Python's Abstract Syntax Tree (AST) manipulation to generate mutants and custom import loaders to intercept module loading for runtime code modification. Core architecture includes a mutation engine with 15+ operator categories (arithmetic, logical, bitwise, comparison operators), constant transformations (strings, numerics, collections), and control flow modifications (conditional inversion, statement removal). The plugin integrates with pytest's collection system through hooks, enabling automatic test replication across all generated mutants with progress tracking. Implemented custom MetaPathFinder and Loader classes to intercept Python's import system, allowing transparent code mutation without modifying source files. Features a two-phase execution model: RecordingLoader for initial AST parsing and mutation point enumeration, followed by MutatingLoader for selective code transformation during test execution. The mutation engine includes intelligent filtering to skip documentation strings, license headers, and other non-functional code elements. Additional components include a type inference system using typeshed for context-aware mutations and AST node enumeration for precise mutation targeting. The framework handles module reloading, cache invalidation, and dynamic test generation, creating isolated execution environments for each mutant. Demonstrates practical application of compiler techniques (AST traversal, code generation), metaprogramming (custom import hooks, runtime code modification), and software testing methodologies (mutation testing, test quality assessment). Approximately 550 lines of production Python code across four core modules, with comprehensive handling of edge cases and Python language semantics. Achieves 10-15x speedup over alternatives thanks to test-wise grouping and parallelization with automatic import conflict resolution.

Stack-Associated Beam Tracing, C++20, Graphics Rendering Project 2022

- Implemented a 3D rendering engine based on NVIDIA's 2010 sparse voxel octree traversal paper, introducing optimizations to reduce empirically-chosen constants in the original algorithm. The system uses hierarchical octree spatial partitioning for scene representation and employs stack-based beam tracing for efficient ray-geometry queries. Core components include a from-scratch linear algebra library (vectors, matrices, transformations), Gilbert-Johnson-Keerthi algorithm for convex collision detection, custom octree traversal with distance-based front-to-back sorting, and real-time rendering pipeline with interactive camera controls. The architecture features 35+ classes organized into modules for geometric primitives (rays, beams, polytopes), spatial structures (octrees with three-state nodes), mesh processing (OBJ parsing, automatic voxelization), and rendering (beam subdivision, adaptive LOD). Implemented optimizations include beam reuse for temporal coherence, angular size-based early termination, and cache-friendly data structures. The project demonstrates practical application of spatial acceleration structures, computational geometry algorithms, and modern C++ practices including template metaprogramming, RAII patterns, and comprehensive unit testing with CI/CD integration. Tested with production 3D models and

configurable precision levels supporting arbitrary octree depths.

RFK: Generalized Hash-Based Multiple Pattern Matching Algorithm, C++17/20, Algorithms and Data Structures 2023

- Designed and implemented a novel hash-based algorithm for exact multiple pattern matching, addressing fundamental performance limitations in intrusion detection systems (NIDS/IPS), bioinformatics, and large-scale text processing. Core contributions include the Length Hashing Search Tree (LHST), a novel data structure guaranteeing $O(1)$ false positives with $O(m)$ worst-case complexity per position, improving upon Wu-Manber's $O(nmk)$ bound through two variants: Full LHST using memory-efficient 2-bit arrays with $O(\log m)$ traversal and Sparse LHST with adaptive hash tables achieving $O(1)$ average-case performance; Cumulative Hash Array (CHA) enabling $O(1)$ arbitrary substring hash extraction through rolling hash schemes supporting both polynomial hashing (collision-resistant) and rotation-based hashing (single-instruction bitwise operations) with three update algorithms and heuristic selection achieving $O(\min(R, n))$ complexity; and adaptive shift computation with block size optimization using convex analysis of character probability distributions, implementing ShiftBloom compressed shift tables with MurMurHash3 for cache locality. Applied systems-level optimization including SIMD-friendly layouts, cache-conscious access patterns, circular buffers with power-of-2 masking, and dynamic collision handling with fallback mechanisms. Conducted formal complexity analysis including adversarial average-case bounds (O_{adv}) critical for security applications, proving $O(km)$ preprocessing, $O(n)$ average-case matching, $O(nm)$ worst-case (competitive with Aho-Corasick), and $O(k)$ average memory with graceful degradation. Authored comprehensive research paper with formal proofs, pseudocode, and comparative analysis against established algorithms (Aho-Corasick, Wu-Manber, Commentz-Walter, DAWG-MATCH, MDH). Implementation features custom data structures (CircularBuffer, FastVector, BitArray), template-heavy design, and comprehensive testing infrastructure with 20+ correctness tests covering edge cases (binary/ternary alphabets, pathological inputs) and 60+ performance benchmarks evaluating uniform/Zipfian/adversarial distributions across DNA sequences, English text corpora (100MB), and real-world ClamAV antivirus signatures. Built automated comparison framework testing alphabet sizes (2-60+ characters), pattern counts (1-10,000), and lengths (5-10,000+ characters) against reference implementations. Demonstrates expertise in algorithm design, complexity theory, hash functions, performance optimization, modern C++ practices, formal methods, and academic writing, with applications to network security (NIDS/IPS packet inspection), bioinformatics (genomic pattern search optimized for 4-character alphabets), and antivirus systems (malware signature detection).

Silver-qt: Russian Sign Language Recognition System, Python, PyQt5/QML, ONNX Runtime, Computer Vision 2019

- Developed a real-time Russian Sign Language recognition application with dual-mode operation (webcam streaming and video file processing) using PyQt5/QML for cross-platform GUI and ONNX Runtime for optimized deep learning inference. The system implements a three-stage neural pipeline: YOLOv5-based hand detection (384×384 input), convolutional autoencoder for feature compression (32×32 crops to 64-dimensional latent vectors), and LSTM-based temporal sequence classification across 51+ sign classes. Core architecture features multi-threaded processing with QThreadPool for concurrent video capture, inference, and UI rendering, preventing blocking operations during computationally expensive model execution.
- Implemented comprehensive video processing pipeline including custom letterbox preprocessing with stride-aware padding, coordinate space transformations for bounding box scaling, Non-Maximum Suppression (NMS) with configurable IoU thresholds for robust multi-hand detection, and adaptive frame sampling with performance-based step adjustment. The detection system processes video sequences through LoadImages iterator supporting 7 video formats and 8 image formats, applies center-crop normalization around detected hand regions with automatic size equalization, and employs max-pooling aggregation across multiple

detected hands per frame to generate frame-level feature representations. Integrated torchvision.ops.nms for efficient duplicate detection filtering with class-agnostic batched processing.

- Engineered production-grade UI with Material Design styling, featuring asynchronous progress tracking with dynamic time estimation (exponential moving average for smoothing), dual-buffer frame rendering to prevent tearing, thumbnail generation for video preview, and CSV export functionality for batch predictions. Implemented probabilistic pseudo-labeling system (CLASSIFIER dictionary) enabling online model adaptation through user feedback, where classification confidence is redistributed across related sign classes using Bayesian-inspired probability updates. The architecture demonstrates practical application of computer vision preprocessing (xywh2xyxy conversion, clip-coords boundary handling, scale-coords transformations), multi-threaded Python patterns (QRunnable with signal/slot communication), and modern ML deployment practices (ONNX interoperability between PyTorch training and production inference). Successfully processes video streams at 20 FPS with configurable quality-speed tradeoffs, supporting interactive translation with ¡90-second batch processing estimates.

Nano JAX Implicit Neural Representations, Python, JAX, Equinox 2024

- Developed a minimalistic implementation of implicit neural representations (INRs) for image compression and encoding using JAX and Equinox frameworks. The system learns continuous neural functions that map 2D coordinates to pixel values, enabling efficient image storage through learned parameters rather than explicit pixel arrays. Core architecture consists of a coordinate-to-latent encoder using K-nearest neighbors interpolation (configurable neighbors, typically 4) with gradient-weighted initialization via Sobel filters, and an MLP decoder with optional parameter sharing across images. Implemented two encoding schemes: STANDARD mode with exponentially-spaced harmonic frequencies (0.125-4096 Hz range) using sinusoidal positional encoding, and RELPOS mode with relative position embeddings. Key components include adaptive latent point sampling based on image gradient magnitude, coordinate subsampling during training for efficiency, NaN-aware loss computation for batched variable-size images, and comprehensive configuration system supporting multiple architectures and hyperparameter sweeps. Performance optimizations include JIT compilation with persistent caching, multi-GPU/TPU data parallelism via JAX sharding, grid-based image size quantization to reduce recompilation overhead, and efficient batching with dynamic padding strategies. Supports multiple datasets (MNIST, FashionMNIST, CIFAR-10, ImageNette) with configurable preprocessing pipelines. The implementation features 8 modular Python files with strong typing via jaxtyping, functional programming patterns with Equinox modules, automatic differentiation for end-to-end training, and robust error handling with skip/stop/attempt modes. Training pipeline includes pre-training phase for shared decoders across image collections, per-image encoder optimization with frozen decoders, and PSNR-based evaluation with intermediate checkpoint saving. Demonstrates practical application of neural fields, coordinate-based networks, and differentiable rendering techniques in modern JAX ecosystem with production-grade engineering practices including comprehensive type annotations, modular design patterns, and distributed training support.

Nano JAX GPT, JAX, Equinox, Deep Learning 2023

- Implemented a high-performance GPT transformer language model from scratch using JAX and Equinox, featuring custom Flash Attention with GPU/TPU acceleration via JAX Pallas operations and fallback CPU implementation. The system supports distributed training across multiple accelerators using JAX's PositionalSharding with automatic data parallelism, mixed-precision training (bfloat16 compute with float32 accumulation), and memory-efficient gradient accumulation with memory donation for 2x efficiency. Core architecture includes token and positional embeddings, multi-head causal self-attention with NaN masking for variable-length sequences, pre-normalization transformer blocks with GELU activations, and weight-tied language modeling head. Explored multiple advanced optimization techniques across experimental branches: (1) early-exit/speculative decoding with per-layer prediction heads enabling adaptive computation and 30-50% inference speedup by skipping

later transformer layers based on confidence thresholds, tracking expected compute dynamically, (2) RoPE (Rotary Position Embeddings) replacing learned positional encodings, (3) RMSNorm replacing LayerNorm for 15-20% faster normalization, (4) custom implicit optimizer implementations including hand-rolled AdamW with selective weight decay masking, Adagrad variants, and Heun's method (second-order Runge-Kutta) for gradient updates, (5) multi-stage training schedules with adaptive gradient scaling, (6) structured embedding sparsity masking zeroing 10% of dimensions to encourage representational efficiency. Training infrastructure features AdamW optimizer with gradient clipping, warmup cosine decay scheduling, asynchronous data loading with memory-mapped datasets, WandB logging, and checkpoint management. Implemented two model configurations: CharGPT (6-layer, 384-dim, character-level) and GPT-2 (12-layer, 768-dim, 117M parameters, BPE tokenization). Architecture organized into 8 modular components (1000 LOC) with type-safe arrays via jaxtyping. Demonstrates advanced JAX features including JIT compilation with persistent caching, filter transformations for pytree manipulation, and strict dtype promotion for numerical stability. Follows GPT-2 initialization with depth-scaled residual projections.

IEFT-PFN: inference efficient freeze thaw prior fitted networks for improved hyperparameter optimization, JAX, Equinox, Deep Learning, Bayesian Optimization 2025

- Implemented a multi-stage Prior-Data Fitted Network (PFN) system for hyperparameter optimization (HPO) using transformer-based in-context learning to predict machine learning training curves and accelerate AutoML workflows. The system employs freeze-thaw Bayesian optimization, enabling early stopping decisions by predicting future performance from partial learning curves. Core architecture features an 8-layer causal transformer encoder (64-dim hidden, 4-head attention) with joint time-value embeddings, temporal conditioning via learned position encoders (50 time steps), and histogram-based probabilistic decoder (500 bins with learned boundaries) for uncertainty-aware predictions. Supports up to 10 hyperparameters with configurable context lengths (100-3200 tokens). Implements three hyperparameter weighting schemes for multi-curve aggregation: (1) covariance-based with learned SPD inverse covariance matrix for Mahalanobis distance, (2) learned attention mechanism using query-key transformations with embedding-conditioned hyperparameter representations, (3) identity-based Euclidean distance fallback. Training infrastructure uses three-stage curriculum: Stage 1 trains on synthetic single curves generated via parametric models (pow4, exp4, ilog4, hill4 basis functions with saturation, randomized 6-layer MLPs with sparsity 0.05-0.8 for hyperparameter-to-curve mapping per IFBO Appendix A1, marginal normalization via inverse-CDF estimation), Stage 2 trains on synthetic multi-curve batches with shared hyperparameter targets, Stage 3 fine-tunes on real benchmark datasets (LCBench, TaskSet, PD1) with configurable encoder freezing strategies. Synthetic data generation follows IFBO methodology with 500 pre-calibrated PiConfig instances mapping 10-dim hypercube to 26-dim curve parameters via fitted MLPs ensuring U(0,1) marginals, supporting variance injection and four parametric basis functions with learned convex combinations. Fine-tuning supports three adaptation modes: full model training, combination layer only (encoder frozen, 100x faster convergence), and combination with decoder. Evaluation infrastructure measures log-likelihood and median log-likelihood (MMedLL) across three benchmark suites, achieving 6.5+ MMedLL on learned weighting vs 5.5 IFBO baseline with 3200 context tokens. Implements memory-efficient embedding caching with automatic padding and masking for variable-length curves, gradient accumulation via JAX filter transformations, numerical stability through error checking for NaNs/Infs, and comprehensive experiment tracking with WandB. Architecture organized into modular components (2300 LOC): PFN model with TransformerLayer blocks, JointEncoder for time-value pairs, HistogramDecoder with softmax normalization, Conditioner for temporal embeddings, HyperMap networks for learned attention, and Config system with automatic seed hashing for reproducible experiments. Training uses AdamW with cosine decay (5e-4 to 5e-6 over 5000 steps, 1e-6 weight decay), gradient clipping (max norm 1.0), warmup scheduling for fine-tuning (50 steps), and apply-if-finite wrapper for numerical safety. Demonstrates advanced JAX features including filter_vmap for selective transformations, JIT compilation with donated buffers, pytree serialization with

mixed precision (custom filter specs for bfloat16 handling), and efficient tree operations via eqx.tree_at for non-destructive updates. Supports distributed evaluation on 800+ real hyperparameter configurations per benchmark, automatic result aggregation per dataset, and checkpoint management with YAML metadata. Achieves 15-20% improvement over IFBO baseline on real benchmarks through learned attention weighting and multi-stage training, while being order of magnitude faster (and asymptotically) on certain sparse problems (and never slower).

Spectral: Atypical Speech Analysis Platform, Docker, Python, TypeScript, Microservices Architecture 2023-2024

- Designed and deployed containerized microservices architecture for web-based atypical speech analysis platform serving clinical researchers (deployed at spectral.ewi.tudelft.nl, certificates may be expired). Orchestrated multi-container system using Docker Compose with FastAPI backend, SvelteKit frontend, PostgreSQL database, and Nginx reverse proxy. Implemented production-grade infrastructure including connection pooling (psycopg), environment-based configuration management, service health monitoring (Dozzle), volume management for persistent storage, and streamlined single-command deployment with configurable port bindings. Architected plugin-based analysis framework enabling modular integration of six acoustic analysis modes (spectrograms, formant tracking, transcription, error metrics) with RESTful API contracts and async processing pipelines. Established DevOps workflow supporting hot-reload development and production environments. Backend contributions include FastAPI endpoint implementation, integration of acoustic processing libraries (Praat-Parselmouth), and multi-model transcription orchestration (Whisper, Deepgram). Frontend contributions include analysis mode plugin system, authentication flow (Lucia), and ORM schema design (Drizzle). System demonstrates practical application of containerization, service orchestration, API gateway patterns, database optimization, and cross-platform deployment strategies, bridging scientific computing (NumPy, SciPy for signal processing) with modern web infrastructure.