Garrett Fincke

Full-stack developer specializing in clean code, scalable systems, and modern web/mobile architecture
garrettfincke@gmail.com | 724-777-7186 | fincke.dev | github.com/ggfincke | linkedin.com/in/garrett-fincke

Education

The Pennsylvania State University Aug 2021 – Dec 2024
Bachelor of Science in Computer Science *University Park, PA*

South Fayette High School Aug 2017 – Jun 2021
High School Diploma *McDonald, PA*

Experience

Scale AI May 2024 – Jul 2025
Software Engineer (contract, part-time) *Remote*

- Built internal tooling and harnesses to evaluate LLM-generated code for top-tier technology clients
- Automated compilation, execution, linting, and test orchestration to measure pass@k and correctness
- Designed training/eval data pipelines (schema versioning, validation, deterministic sampling) to replace manual review with scripted checks
- Instrumented quality metrics and dashboards (syntax/style error rates, failure modes) and partnered with research/eng to triage regressions
- Enforced ingest-time data quality (normalization, deduplication, constraint checks) to improve dataset reliability and iteration speed

Pink Ocean Collectibles Apr 2020 – Present
Owner / Operator *Online Ecommerce*

- Build and operate multi-channel storefront with 2000+ sales, $75k+ lifetime profit, and 100% positive feedback
- Develop comprehensive automation suite including listing pipelines (Hopper + marketplace APIs for eBay, TCGPlayer, Facebook Marketplace, StockX), dynamic pricing tools based on market trends, and analytics dashboard for performance tracking
- Source diverse product inventory including refurbished modern technology (iPhones, Game Boys/GameCubes, vintage PCs), Pokemon cards, comic books, and collectible books from estate sales, thrift stores, auctions, online marketplaces, and wholesale suppliers
- Repair and mod consumer electronics (e.g., iPhones, Game Boys/GameCubes, vintage PCs) and sell restored units through storefront

Pennsylvania State University Aug 2024 – Dec 2024
CMPSC 475 Learning Assistant (iOS / Applications Programming) *University Park, PA*

- Mentored students in iOS/mobile application development using Swift and SwiftUI, emphasizing clean code practices and sound design patterns
- Reviewed and debugged student code, enhancing their understanding of application programming concepts and code quality standards
- Collaborated with faculty to tailor instruction based on student progress and technical challenges with focus on strong coding fundamentals

Chipotle Mexican Grill Oct 2022 – May 2024
Crew Member *Bridgeville, PA and State College, PA*

- Prepared and cooked food while maintaining strict adherence to food safety standards and quality control
- Washed dishes, cleaned prep stations, and ensured cleanliness of store with attention to operational efficiency
- Served customers on the line with focus on speed, accuracy, and hospitality during high-volume periods
- Worked collaboratively with coworkers and managers to maintain smooth operations during peak hours and busy periods

Scott Township Swimming Pool May 2022 – Aug 2022
Pool Manager *Carnegie, PA*

- Oversaw daily pool operations to ensure safe, fun, and welcoming environment for all patrons with focus on safety protocol compliance

- Trained and mentored junior lifeguards on Red Cross and Scott Pool safety protocols, developing leadership and training skills
- Coordinated and executed community events and holiday celebrations at the pool, demonstrating project management abilities
- Enforced safety regulations and responded to incidents with professionalism, composure, and adherence to emergency procedures

Scott Township Swimming Pool Jun 2019 – Aug 2021
Lifeguard *Carnegie, PA*

- Monitored pool activity and enforced safety rules in alignment with Red Cross and Scott Pool protocols for patron safety
- Maintained cleanliness and presentation of pool area and surrounding park grounds with attention to detail and public service
- Taught group and private swim lessons to individuals of varying ages and skill levels, developing instructional and communication skills
- Collaborated with fellow lifeguards to ensure safe and fun environment for all patrons during summer seasons

Projects

SwimMate v2 — *Swift, SwiftUI, watchOS, HealthKit, WatchConnectivity, Swift Charts, Django 5.2 + DRF 3.16, PostgreSQL 16 (pgvector), OpenAI API, SimpleJWT, Docker Compose, pytest*
Full-stack swimming platform (monorepo: apps/, infra/, docs/) with native iOS/watchOS apps, HealthKit sync, AI-powered workout generation, and cursor-based pagination.

- Built Django REST API (DRF 3.16.1, SimpleJWT 5.5.1, drf-spectacular) with JWT auth + rotation, OpenAPI 3.0/Swagger UI, tier-aware throttling for AI endpoints, request-ID distributed tracing, and Docker Compose deployment (PostgreSQL 16 + pgvector auto-init, Gunicorn)
- Implemented RAG pipeline with 768-dimensional pgvector semantic search, constraint-based reranking (stroke/difficulty/intensity with configurable weights), role-based composition (warmup/main/cooldown via RoleComposer), unified constraint processor (extract → merge → classify → build), audit logging, and HTTP 207 partial generation support
- Designed workout builder supporting sets, templates, components (swim/drill/kick/pull/rest/warmup/cooldown), equipment, effort levels (recovery/easy/moderate/threshold/sprint), pace profiles (SCY/SCM/LCM), and learned paces auto-computed from workout history via `populate_learned_paces` management command
- Built actor-based Swift networking layer with async/await, automatic 401 token refresh, Keychain JWT storage, background HealthKit deduplication sync (`hkWorkoutId` unique constraint), and cursor-based paginated swim set library
- Developed SwiftUI iOS app (MVVM, `@Observable`/`ObservableObject`) and watchOS app (`@Observable` macros, water lock via WKInterfaceDevice) with live workout metrics (pace, HR, SWOLF, distance, calories), Swift Charts analytics (weekly stats, pace trends, stroke distribution), and goal-based workouts (distance/time/calories)
- Structured iOS project as git submodule within monorepo; maintained comprehensive dev-docs suite (CLAUDE.md, AGENTS.md, MOBILE-API-INTEGRATION.md, design_doc.md) and Bruno API testing collection

Reactive Workbench — *TypeScript, Node.js, VS Code Extension API, ESLint, Prettier*
Context-aware VS Code automation (v1.0.2, 2.5K+ lines TypeScript) that adapts themes, panels, layouts, and editor features based on workspace state.

- Built first-match-wins rule engine with overlap detection, mode bundles for reusable action sets, analytics/history tracking, and undo/restore for last applied change
- Implemented 15+ action types: `theme`, `editorFeature`, `vscodeProfile`, `panel`, `sidebar`, `activityBar`, `statusBar`, `tabBar`, `zenMode`, `centeredLayout`, `terminalProfile`, `notification`, `runTask`, `runCommand`, `commandChain` (sequential multi-command execution)
- Supported 17+ condition types: file patterns (glob), language IDs, workspace names, debug/test session state, Git branch/status, diagnostics (error count thresholds), time of day, remote session type (SSH/WSL/container), window focus, editor layout, workspace trust, and timer intervals

- Shipped 14+ commands: toggle, safe mode, reload rules, refresh state, create rule from file, manage rules, lint rules, test rule (preview against context), explain workbench, import/export rules, switch mode, show analytics, show history, undo last change
- Ensured safe, reversible operation with config-based reliable actions and best-effort toggle-based actions applied through VS Code APIs; documented known state-drift limitation for toggle-based actions with restore command

Portfolio Website v2 — *Vite 7, React 19, TypeScript 5.8 strict, Tailwind CSS 4, React Router 7, ESLint 9 (flat config), Prettier 3, simple-icons*
Modern portfolio rebuild live at fincke.dev; section-based architecture with content-driven development patterns and static deployment (Vercel/Netlify/GitHub Pages).

- Architect section-based modular system isolating features into dedicated route directories (home, experience, featured-projects, projects-archive) with consistent internal structure for components, content, hooks, and utilities; home route includes hero, about, social CTA, job history, and featured projects grid
- Implement content-driven typed pipeline separating type definitions (`src/shared/types/`), structured content modules (`projects.tsx`, `experienceTimeline.ts`, `skills.ts`), and presentation components; projects archive route (`/projects`) with responsive table/card layouts and expandable per-project details
- Build comprehensive shared resource system with reusable UI components (SkillPill, StatusBadge, StatusCircle, VersionBadge), simple-icons for branded social links, centralized type definitions with barrel exports, and cross-cutting utilities
- Configure TypeScript 5.8 strict mode with project references (`tsconfig.app.json`, `tsconfig.node.json`), verbatim module syntax, `~/` path aliases across Vite and TypeScript configs, zero-config Tailwind 4 via Vite plugin, and consistent named exports pattern
- Design single-file token system in `src/styles/globals.css` (CSS variables for accent/bg/card/fg/muted/border/ring) with 6-status badge variants (live, in-development, paused, complete, experimental, planned) and staggered slide-in animations with accessibility support
- Enforce developer experience standards with ESLint 9 flat config, Prettier 3, consistent type imports rule, and Vite 7 HMR dev server

Modern MDX Preview — *TypeScript, React 18, MDX 3, VS Code Extension API, Vite 7, esbuild, Comlink, Shiki, KaTeX, DOMPurify, Tailwind CSS 4, Mermaid, PlantUML, Graphviz*
VS Code extension and standalone npm library (42k+ LOC, 6-package monorepo) providing live MDX preview with React 18 component rendering and two-mode security architecture.

- Architect 6-package npm workspace monorepo with dual-build pipeline (esbuild for Node.js extension host, Vite 7 for React 18 webview client) and extract core runtime into standalone MIT-licensed npm library (mdx-forge) with 11 subpath exports
- Implement two-mode rendering gated by 4 security checks (workspace trust, user setting, local environment, file scheme): Safe Mode compiles MDX to DOMPurify-sanitized static HTML with strict CSP; Trusted Mode compiles to executable JavaScript for full React 18 component rendering with dynamic Content Security Policy generation; per-project `.mdx-previewrc.json` configuration in Trusted Mode
- Build 4-strategy module resolver chain (framework shim → TypeScript paths → enhanced-resolve → file probe) with auto-detection from `package.json`, dual MDX compilation pipelines, and 10+ custom remark/rehype plugins including Shiki (100+ languages, 23 code block themes), KaTeX math, and lazy-load diagram placeholders
- Design component registry supporting 5 documentation frameworks (Docusaurus, Starlight, Nextra, Next.js, generic) with 35+ React component shim implementations, build-time codegen, and 15 preview themes with auto light/dark switching; render diagrams (Mermaid, PlantUML, Graphviz via WASM); webview constraints include no Service Workers or Local Storage
- Ship 54 user-configurable settings, 14 commands, and enforce quality with Vitest test suites (88 test files, 15k+ LOC), dependency cruiser (11 architectural rules), and custom ESLint rules

OpenCode to ccusage — *TypeScript, Node.js, OpenCode CLI, ccusage*
CLI tool (Node.js 18+) that exports OpenCode sessions to ccusage-compatible JSONL and generates combined usage reports; largely superseded as of January 2026 after Anthropic patched the underlying method.

- Converted OpenCode session data into ccusage JSONL with `flat/project/directory` grouping strategies and field mapping (timestamp, sessionId, cwd, requestId, model, token counts including cache read/write)

- Built `report` command merging OpenCode and Claude Code usage data for unified metrics with incremental export (only re-exports changed sessions) and pass-through flags to ccusage (`--monthly`, `--since`)
- Added `export` command with full controls: since-date/days filtering, dry run, custom output dir, overwrite, skip-validation, configurable concurrency, and incremental tracking via session modification timestamps
- Implemented schema validation, configurable concurrency with semaphore, deduplication by (messageId, timestamp), and deterministic timestamp-ascending sort
- Exposed three CLI commands (`report`, `export`, `advanced`) with verbose progress logging throughout

Minecart — *TypeScript, Node.js, Discord.js, Prisma ORM, PostgreSQL, Docker Compose, RCON, AWS (EC2, SSM, CloudWatch, S3), Vitest*
Discord bot for managing Minecraft servers across local Docker and AWS EC2 with multi-tenant support.

- Built interactive Discord slash command dashboards (buttons, modals, select menus) for status, start/stop/restart, logs, backups, worlds, performance, players, and configuration with real-time updates
- Implemented provider abstraction (`IServerProvider`) with Local Docker and AWS EC2 implementations sharing a common API with runtime switching, health checks, live status via RCON + mcstatus.io, and circuit breakers
- Added background services for scheduling (idle shutdown, start/stop windows, automated backups with retention), crash detection (auto-restart with configurable attempts and cooldown), server notifications (player deaths, advancements, join/leave broadcast to Discord), and performance monitoring with lag spike analysis
- Shipped 20+ slash commands: `/server`, `/worlds`, `/performance`, `/config`, `/schedule`, `/notifications`, `/players`, `/admin`, `/gameplay`, `/server-create`, `/server-delete`, `/server-members`, `/permissions`, `/channel-bind`, `/audit`, `/maintenance`, `/stats`, `/lag`, `/usage`, `/help`
- Delivered multi-tenant controls via Prisma ORM + PostgreSQL: role-based permissions (view/operate/admin/owner), channel bindings, guild defaults, usage metering/quotas with Discord notifications, and service container with dependency injection (factory functions + service keys)
- Built world and content management: named worlds, disposable test environments with auto-expiration, blue-green cutover, Modrinth mod search/install, resource pack management, and server version switching via Paper API
- Added governance and monitoring: audit logs with retention, maintenance mode with lease-based auto-recovery, real-time performance metrics, player statistics with session history, optional S3 backup storage with presigned URLs, and Vitest test suite with 15+ service mocks and MSW HTTP mocking

Hopper — *Java 21/24, Spring Boot 3.5.5, Spring Security, JWT, Go, PostgreSQL 15, Flyway, H2, Spring Data JPA, Spring Batch, Hibernate Validator, Spring Boot Actuator, Gradle, Docker Compose*
Hybrid microservices inventory management system for e-commerce with Java/Spring Boot orchestration and Go marketplace connector.

- Architect hybrid microservices system with Java/Spring Boot orchestration layer and standalone Go marketplace connector service for platform integrations
- Design 11-entity domain model (User, Role, Platform, PlatformFee, PlatformCredential, Product, Listing, Order, OrderItem, Buyer, OrderAddress) with comprehensive service layer, RESTful controllers, and repository pattern
- Build JWT authentication system with access/refresh tokens, role-based authorization (ADMIN, USER, API_CLIENT), secure token rotation, and comprehensive storage guidelines
- Implement AES-GCM-256 credential encryption with PBKDF2 key derivation (100k iterations), encryption metadata storage for versioning, and key rotation support via CredentialEncryptionService
- Create normalized PostgreSQL schema with 13 Flyway migrations, foreign keys, composite unique constraints (platform + external_order_id), and performance indexes
- Develop Go marketplace connector microservice with OpenAPI 3.1 specification (391 lines), RESTful endpoints for listings/orders, idempotency support, error taxonomy, and sample fixtures
- Implement Spring Batch integration with background jobs for order import and marketplace synchronization with MarketplaceClient interface abstraction
- Build comprehensive testing suite with 27+ test files including integration tests, contract validation, and Bruno HTTP collection for API testing

- Design order state machine (pending→confirmed→paid→processing→shipped→delivered) with transition validation, business rule enforcement, stock management, and insufficient stock handling
- Configure multi-profile deployment (dev/test/prod) with H2 development database, PostgreSQL production schema, and environment-based security configuration
- Configure Docker dev stack (5 services: Spring Boot API port 8080, PostgreSQL 15 port 5432, Go marketplace stub port 8090, optional Vite frontend port 5173, Flyway seed runner) with Makefile helpers (`dev-up`, `dev-logs`, `api-shell`) and override-file pattern for local port remapping
- Integrate Spring Boot Actuator for health checks and monitoring; enforce bean validation with Hibernate Validator; multi-profile configuration (dev with H2, test, prod with PostgreSQL) via `application-{profile}.properties`

Loom — *Python, Typer, Rich, OpenAI, Anthropic, Gemini, Mistral, Ollama, LM Studio, DOCX, LaTeX, Typst, JSON*

AI-powered resume tailoring CLI with structured edits, diff review, and versioned workflows.

- Built Typer-based CLI (Python 3.12, Conda) with 13 commands: `sectionize`, `tailor`, `generate`, `apply`, `snapshot`, `versions`, `ats`, `export`, `bulk`, `cache`, `templates`, `config`, `models`; Rich-based themed help UI; configurable defaults via `~/.loom/config.json` or CLI
- Implemented structured JSON edit pipeline with interactive diff resolution, risk/on-error policies (strict/soft/retry), and format-preserving DOCX (in-place or rebuild) / LaTeX / Typst handling
- Integrated 6 AI providers (OpenAI, Anthropic/Claude, Gemini, Mistral, Ollama, LM Studio) with model selection, automatic LRU response caching with TTL to reduce API costs, and file-watch mode for auto-rerun on input changes
- Added snapshot-based version management (stored in `.loom/versions/`): create, list, diff, restore, prune; comparison matrices for bulk job processing with configurable parallelism
- Shipped ATS compatibility checks (structural + AI modes, CI-friendly `--fail-on critical`), bulk job processing generating individual tailored resumes plus fit-score comparison matrix, and exports to txt/pdf/html
- Enforced quality with pytest and smoke tests; validated via sectionize → tailor → ats end-to-end workflow

TrackBasket — *TypeScript, Next.js 15, React, Tailwind CSS, Radix UI, Recharts, Python 3.12+, Supabase, PostgreSQL, OpenAI*

Price tracking platform monitoring 30k+ products across major retailers with AI-powered features.

- Developed chat-to-basket feature using OpenAI API converting natural language into structured product baskets with Supabase data integration and real-time price subscriptions
- Implemented advanced web crawling system (Python 3.12+, 10+ concurrent workers) with CAPTCHA solving, User-Agent spoofing, intelligent rate limiting, error recovery, and retailer-specific parsing logic for Amazon, Target, and Walmart
- Created comprehensive Supabase backend with PostgreSQL row-level security (multi-tenant), Edge Functions, fuzzy search, UPC-based cross-retailer product matching with confidence scoring, and real-time subscriptions for live price updates
- Built AI-powered product matching with intelligent alternatives, cross-retailer correlation, and category discovery; Recharts interactive price history charts with zoom/filter; dark/light theme support via CSS variables
- Engineered sophisticated notification system with granular user preferences for price drops, availability changes, and product updates; mobile-first responsive design (100% mobile compatibility)
- Developed responsive basket management with collaborative sharing, drag-and-drop product organization, and smart AI recommendations via OpenAI
- Built solo for Bolt Hackathon: Next.js 15 App Router, TypeScript, Tailwind CSS, Radix UI accessible components, Recharts visualizations

SwimMate — *Swift, SwiftUI, HealthKit, WatchKit, Swift Charts, WatchConnectivity, WKInterfaceDevice, iOS 17.4+, watchOS 10.4+*

Native iOS/watchOS app for swimmers with comprehensive tracking and Apple Watch integration.

- Develop comprehensive swimming app for tracking, finding, and saving workouts with progress visualization over time
- Build custom components using HealthKit and SwiftUI for workout entry, lap timing, and charting performance trends with Swift Charts

- Connect iOS app to Apple Watch for real-time workout data tracking, sending premade workouts, and displaying rich metrics
- Implement goal-based workouts for distance, time, or calories with real-time progress tracking and pace monitoring
- Create full-featured app supporting offline workout tracking, cross-device sync via WatchConnectivity, custom workout builder, workout history with detailed analytics, and data export/sharing capabilities
- Design Apple Watch interface displaying real-time metrics including pace, heart rate, laps, SWOLF, and calories burned
- Support both pool and open-water swims with GPS distance tracking and comprehensive HealthKit integration
- Achieved grade of 100% on original submission, demonstrating excellence in iOS/watchOS development and app architecture
- Implemented MVVM architecture: Model (Swim, SwimSet), ViewModel (Manager/HealthKit, WatchConnector), Views; data persisted locally via JSON storage; all health data stored via HealthKit with no third-party sharing
- Tracked 7 swim metrics: distance (m/yd), duration, laps, pace (per 100m/yd), SWOLF efficiency score, calories, and heart rate; water lock activated via WKInterfaceDevice; original CMPSC 475 course project at Penn State

Portfolio Website v1 — *Next.js 15.5.2, React 19, TypeScript 5, Tailwind CSS 4.1.12, next-themes, GitHub Actions, Lighthouse CI, ESLint 9, Turbopack, Figma*
First portfolio iteration (now archived; v2 rebuilt on Vite 7/React 19); established full CI/CD pipeline, theming, and performance baseline.

- Built responsive portfolio with Next.js 15.5.2 App Router, React 19, TypeScript 5, and Tailwind CSS 4.1.12; fixed desktop sidebar layout with custom breakpoint utilities, animated hero typing effect, and pagination dots
- Implemented multi-theme support via `next-themes` with CSS variables generated by `scripts/generate-theme-css.j` from a centralized `src/themes/config.ts` source of truth; floating theme selector on desktop, inline on mobile
- Built sections: Hero (animated typing), About, Experience (redirect to resume PDF), Projects with archive page; interactive skill pills with hover tooltips showing related projects; Vercel Analytics integration
- Configured full CI/CD with GitHub Actions (lint, typecheck, build, Lighthouse CI) and Lighthouse budget thresholds: performance $\geq$0.80, accessibility/best-practices/SEO $\geq$0.85 (`lighthouserc.json`)
- Automated release workflow: Dependabot dependency health, pre-commit git hook auto-bumps `package.json` to prerelease version and prepends CHANGELOG section on `dev` branch
- Optimized with Next.js 15 Turbopack dev server, Geist Sans/Mono fonts, and performance-first App Router architecture; deployed to Vercel
- Designed layout and logo in Figma; used `useTypingAnimation`, `useSectionNavigation`, and responsive `useBreakpoint`/`useNav` custom hooks

InStock — *Python 3.12, Django, Celery, PostgreSQL, Redis, Chrome/Selenium, React, Discord.py, Docker Compose, Conda*
High-performance price & stock tracking system; foundation project that evolved into TrackBasket.

- Designed optimized tracking system for detecting restocks and price changes across Amazon, Walmart, Target, Best Buy using Chrome/Selenium scrapers with ARM compatibility (`DOCKER_DEFAULT_PLATFORM=linux/amd64`)
- Created custom Django ORM schema for products tracking price changes, stock availability, and price trend analytics; efficient retrieval with PostgreSQL indexing
- Engineered 6-service Docker Compose architecture: `web` (Django), `postgres`, `redis`, `celery_worker` (background task processing), `celery_beat` (scheduled tasks), `discord_bot`; Conda-managed Python 3.12 environment for local tooling
- Built comprehensive RESTful API endpoints for product management, price tracking, and user authentication serving React frontend; email + Discord notifications via Discord.py for price drops and restock events
- Integrated Discord bot with `/test scrapers` and `/test scraper <platform> <url>` commands for scraper validation
- Project laid groundwork for TrackBasket, which expanded the architecture with AI product matching, UPC cross-retailer correlation, and Supabase real-time infrastructure

Deep Learning Architecture Comparison & Analysis for CIFAR-10 — *Python, TensorFlow, Keras, NumPy, pandas, matplotlib, seaborn, scikit-learn, Deep Learning, CNN, ResNet, DenseNet, LaTeX, Random Fourier Features*

Comprehensive deep learning study benchmarking four approaches on CIFAR-10 dataset.

- Benchmarked four approaches on CIFAR-10 dataset (60k images): baseline CNN, ResNet50, DenseNet121, and Random Feature Model (RFM) with 5,000 Random Fourier Features
- DenseNet121 achieved top test accuracy at 74%, outperforming baseline CNN (69%), RFM (51.6%), and ResNet50 (47%)
- Engineered ResNet-inspired CNN with residual blocks, data augmentation, dropout, and L2 regularization
- Documented optimization challenges and remedies throughout model development process
- Built lightweight RFM pipeline (StandardScaler → RBFSampler → LogisticRegression) with performance logging at 10 checkpoints
- Generated confusion matrices, full metric suite (accuracy, precision, recall, F1, log-loss), and detailed loss/accuracy curves; proposed future work including hybrid CNN-RFM ensemble, advanced ResNet scheduling, and larger datasets
- Achieved grade of 100% on this comprehensive report and implementation, demonstrating mastery of deep learning architectures and analysis
- Course: MATH 452 (Fall 2023, Penn State); team of 6 (Jacob Goulet, Tyler Rossi, Diego Bueno, Javier Pozo Miranda, Duong Bao, Garrett Fincke) for Final Project 1; Garrett Fincke primary author on Final Project 2 (CNN vs RFM); same course as MNIST midterm work

TCGhub — *React 18, React Router, Tailwind CSS, Express.js, SQLite3, bcrypt, Chart.js, Google Maps API*

Trading card marketplace platform replicating tcgplayer.com functionality.

- Developed React 18 + React Router + Tailwind CSS trading card marketplace; CMPSC 431W (Database Management Systems), Fall 2024, team of 2 (Garrett Fincke & Yash Tumuluri)
- Customized complex database schema in BCNF (Boyce-Codd Normal Form) with SQLite3; hand-wrote all SQL queries; Express.js backend (port 3001) serving React frontend (port 3000)
- Implemented user authentication with bcrypt password hashing; user profiles with order history and profile editing; shopping cart system with checkout flow
- Built store locator feature using React Google Maps API to find nearby card shops; Chart.js visualizations for card price trend tracking
- Implemented 6 custom hooks: `useAuthentication`, `useCardData`, `useCart`, `useCollectionManagement`, `useLocation`, `useOrders`; comprehensive filtering and search across card sets and rarities
- Achieved grade of over 100% demonstrating mastery of relational database design, normalized schemas, and full-stack React/Express development

Computer Architecture Projects — *C/C++, SimpleScalar, Python, Cache Hierarchies, Branch Prediction, Performance Analysis*

Two comprehensive projects exploring architecture design and branch prediction analysis.

- Built heuristic-driven DSE framework on SimpleScalar exploring 18-dimensional configuration space (pipeline width, L1/L2 cache hierarchy, scheduling, branch prediction, memory latency) across 5 benchmarks per configuration; geomean-based scoring for both Energy-Delay Product (EDP) and execution time optimization; shell scripting for benchmark automation
- Implemented 6 branch predictors in Python with trace-based simulation: static (always-taken/not-taken), one-bit BHT, two-bit saturating counter, bimodal (configurable size, default 1024 entries), GShare (XOR of PC and global history), and hybrid meta-predictor with chooser table selecting between GShare and Bimodal based on per-address accuracy
- Measured prediction accuracy and misprediction rates across traces; generated visualizations of predictor performance comparisons
- Produced detailed reports outlining cache configuration trade-offs, predictor performance, and quantitative recommendations grounded in normalized geomean metrics; CMPEN 431 (Fall 2022, Penn State)

Traditional Machine Learning Methods Exploration for MNIST — *Python, scikit-learn, NumPy, pandas, matplotlib, seaborn, Machine Learning, LaTeX*

Comprehensive ML analysis implementing and comparing KNN, Logistic Regression, and SVM.

- Implemented and compared KNN (94.4% accuracy), Logistic Regression (91.1%), and SVM with RBF kernel (95.3%) on 10k-image MNIST subset
- Applied rigorous preprocessing pipeline: random sampling, normalization to [0,1], flattening to 784-D vectors, stratified 80/20 train-test split
- Performed hyperparameter tuning (k=3 for KNN, C=1 and RBF kernel for SVM) using grid search cross-validation with comprehensive evaluation
- Executed unsupervised learning using K-Means (k=10) with PCA dimensionality reduction and examined elbow method and silhouette scores
- Discussed computational constraints (high-dimensional SVM training) and proposed CNNs as future work to approach state-of-the-art accuracy
- Achieved grade of 100% on report and implementation, showcasing excellence in traditional machine learning methods and statistical analysis
- Course: MATH 452 (Fall 2023, Penn State); Midterm 1 covered KNN/LR/SVM on MNIST (10k samples); Midterm 2 covered K-Means + PCA unsupervised clustering; same course as CIFAR-10 deep learning final projects

COVID-19 Case Surveillance Analysis — *Python, Machine Learning, pandas, NumPy, scikit-learn, matplotlib, seaborn, Jupyter*

Comprehensive data science project analyzing large-scale public health data.

- Completed 13 modules of Python data science coursework (STAT 319 Applied Statistics, Penn State): NumPy, pandas, matplotlib, seaborn, preprocessing, model validation, logistic/polynomial/ridge regression, KNN, SVM, and decision trees
- Implemented regression algorithms including Ridge, Lasso, and ElasticNet regularization; classification models including logistic regression, SVM (linear/RBF kernels), KNN, and decision trees; cross-validation with grid search hyperparameter tuning
- Collaborated on COVID-19 case surveillance final project (team: Edwin Clatus, Garrett Fincke, Sahit Karan Botta) performing multi-class classification to predict patient outcomes on large-scale public health dataset
- Built end-to-end ML pipeline: large-scale preprocessing/cleaning, exploratory visualization, feature engineering, model comparison with confusion matrices and classification reports
- Achieved grade of 100% on final project, demonstrating excellence in collaborative data science and statistical modeling

BetterBettor — *Solidity ^0.8.2, Next.js 14, React 18, Styled Components, Web3.js 4.8, MetaMask, Ethereum*

Decentralized sports betting platform using Ethereum smart contracts.

- Built decentralized sports betting platform on Ethereum using Solidity ^0.8.2 smart contracts for transparent, trustless wagering; CMPSC 263 (Blockchain and Modern Web Development), Spring 2024, Penn State
- Implemented `SportsBetting` contract with key functions: `addGame(oddsA, oddsB)` (owner), `placeBet(gameId, team)` (payable), `setOutcome(gameId, outcome)` (owner), `getUserBets(address)` (public), `withdrawFunds()` (owner); configurable payout limits to manage contract risk
- Developed Next.js 14 + React 18 frontend with Styled Components and Web3.js 4.8 for MetaMask wallet integration; pages for basketball/baseball betting and user bet history (`/mybets`)
- Created multi-sport extensible architecture supporting basketball and baseball with real-time odds display, total pool amounts, and automatic payout calculation

OPTIMUS — *Python, PyTorch, HuggingFace Transformers, Discord.py, APScheduler*

Fine-tuned Discord chatbot using Microsoft's GODEL-v1.1 model for contextual conversation generation.

- Built fine-tuned Discord chatbot using Microsoft GODEL-v1.1-base-seq2seq model (weights in `OPTIMUSv1/`: `config.json`, `model.safetensors`, `tokenizer.json`) for advanced contextual conversation generation
- Implemented 3 interaction modes: **Puppeteer Mode** (respond only when mentioned), **Free Rein Mode** (autonomous responses after configurable random message interval), **Del Mode** (biased generation toward specific person with configurable retry attempts); up to 5 messages of conversation history with 512-token context window
- Automated profile customization via APScheduler: matched avatar/banner pair rotation (prevents consecutive

duplicates), rotating custom status messages on configurable intervals; configurable probability-based emote reactions with user-specific targeting
- Shipped 8 slash commands: `/respond_context` (configurable context depth + instruction), `/togglefreerein`, `/toggle_del_mode`, `/changevars` (emote %, free rein bounds, profile intervals), `/show_settings`, `/reset_settings`, `/add_restricted_channel`, `/remove_restricted_channel`
- Persisted all settings across restarts via JSON (`settings.json`): emote percentages, intervals, channel restrictions, mode states, and current profile index

iOS Application Development Projects — *Swift, SwiftUI, UIKit, MapKit, Core Data, JSON, Custom Shapes, Gesture Handling, MVC/MVVM, Xcode*
Five comprehensive projects demonstrating advanced iOS development skills.
- Built LionSpell word puzzle game with custom polygon shapes, multi-language preferences, hints system, and New York Times-style UI design
- Developed Pentominoes puzzle game implementing drag gestures, 3D rotation animations, JSON data parsing, and automated solve/reset functionality
- Created Campus mapping applications using both SwiftUI Map and UIKit MKMapView with user location tracking, route planning, turn-by-turn directions, and annotation clustering
- Designed Pok'edex catalog app featuring card-based UI, type filtering, capture/release persistence, evolutionary chains, and comprehensive data management
- Achieved grade of 98% average across all projects, demonstrating mastery of iOS development patterns, gesture handling, and advanced SwiftUI/UIKit integration

Operating Systems Projects — *C, Systems Programming, x86-64, Dynamic Memory Allocation, Virtual Memory, Concurrent Channels, Semaphores*
Three CMPSC 473 (Operating Systems, Fall 2023, Penn State) projects implementing real systems programming: custom allocator, x86-64 paging, and concurrent channels. Team: Garrett Fincke & Avanish Grampurohit.
- Implemented custom `malloc`/`free`/`realloc` using 61 segregated free lists organized by power-of-2 size classes, explicit free list with LIFO insertion, boundary tags (header/footer) for 4-case coalescing, and 16-byte alignment; key functions: `mm_malloc()`, `mm_free()`, `mm_realloc()`, `coalesce()`, `find_fit()` (best-fit across size classes)
- Built x86-64 four-level page table hierarchy (PML4 → PDPE → PDE → PTE) with 4GB kernel space identity mapping, user space mappings for program and stack, permission bits (present/R-W/user-supervisor/NX), and `syscall_entry()` handler for user-kernel transitions
- Implemented thread-safe concurrent channels (analogous to Go channels) with bounded buffer, semaphore-based space tracking (`available`/`used`), mutex protection, blocking and non-blocking send/receive, `channel_select()` multiplexing across multiple channels via linked-list waiting thread tracking, and full return code taxonomy (SUCCESS/WOULDBLOCK/CLOSED_ERROR/DESTROY_ERROR)
- Achieved grade of 100% average on all assignments with comprehensive testing using provided trace files and stress tests

MIPS Processor — *Verilog, FPGA, Digital Design, Xilinx Vivado*
Complete single-cycle MIPS processor implementation in Verilog HDL.
- Implemented complete single-cycle MIPS processor in Verilog HDL with 32-bit Harvard architecture; CMPEN 331 (Spring 2023, Penn State); team: Garrett Fincke & Avanish Grampurohit; toolchain: Xilinx Vivado 2022.2+
- Supported R-type instructions (`add`, `sub`, `and`, `or`, `xor`, `sll`, `srl`, `sra`, `jr`), I-type (`addi`, `andi`, `ori`, `xori`, `lui`, `lw`, `sw`, `beq`, `bne`), and J-type (`j`, `jal`)
- Designed modular architecture with core modules: `main.v` (top-level), `controlUnit.v` (opcode decode), `ALU.v`, `regfile.v` (32 registers), `instructionMemory.v`, `dataMemory.v`, `programCounter.v`; datapath: PC adder, branch target adder, PC mux (sequential/branch/jump/jr), ALU B mux, sign/zero-extend mux, mem-to-reg mux, JAL writeback mux
- Generated 10 control signals: `wreg` (register write), `wmem` (memory write), `m2reg` (memory to writeback), `aluimm` (immediate ALU input), `regrt` (rt as destination), `sext` (sign-extend), `pcsrc` (00=PC+4/01=branch/10=jr/11=
`aluc` (ALU op), `shift` (shift amount as ALU A), `jal` (JAL flag)

- Achieved grade of 100% with thorough testbench (`testbench.v`) simulation validating all instruction types, branch targets, and jump/link register writeback

JBOD Storage System with Caching & Network Communication — *C, Systems Programming, Storage Systems, Networking, Caching*

Complete JBOD storage system with block-level operations and distributed architecture.

- Implemented 5-lab progressive JBOD storage system (CMPSC 311, Fall 2022, Penn State): C fundamentals → JBOD basics → extended read/write → LFU caching → TCP network layer
- Built JBOD (16 disks, 256 blocks/disk, 256 bytes/block = 1 MB total) with linear address-to-disk/block translation; implemented `op_create()` (32-bit operation packing), `mdadm_mount()`, `mdadm_unmount()`, `mdadm_read()`, and full cross-disk/cross-block boundary I/O
- Added LFU cache layer (2–4096 entries) with `cache_create()`, `cache_lookup()` (hit/miss tracking), `cache_insert()` (evict lowest `num_accesses` on full), `cache_update()`; reduces disk I/O for repeated block access
- Developed TCP/IP client-server network layer: client sends JBOD operations over socket, server executes on local disk array and returns results; new files `net.c`/`net.h` and `jbod_server` binary
- Achieved grade of 100% on all 5 assignments using provided test harness (`./tester`) and trace-based workload files

USBAP — *Python, Web Scraping, Data Analysis, BeautifulSoup*

Experimental web scraping project for sports betting data extraction and analysis.

- Developed DraftKings MLB scraper extracting run lines (spread+odds), totals (over/under point+odds), and moneylines via structured table layout parsing with per-event row extraction
- Built FanDuel MLB scraper with User-Agent header spoofing to bypass basic bot detection; extracts run lines, moneylines, and totals from FanDuel's DOM structure
- Implemented structured parsing for 3 betting market types (run lines, totals, moneylines) across both platforms; console output with labeled market sections; experimental/proof-of-concept architecture
- Established conceptual groundwork for automated sports betting analytics and arbitrage detection pipelines, influencing later work on automated data collection in InStock and TrackBasket

Technical Skills

Languages: Python, Swift, JavaScript/TypeScript, Java, Kotlin, C, C++, SQL, Solidity, Verilog

Frontend & Mobile: React, Next.js, SwiftUI, UIKit, MapKit, WatchKit, Swift Charts, Tailwind CSS, HealthKit, Core Data, Web3.js

Backend & Data: Django, Node.js, Spring Boot, Spring Security, FastAPI, PostgreSQL, Redis, SQLite, Supabase, Celery, Flyway, REST APIs

AI/ML: OpenAI/Anthropic APIs, PyTorch, TensorFlow, Keras, scikit-learn, HuggingFace, pandas, NumPy, matplotlib, Jupyter

Cloud & DevOps: AWS (EC2, SSM, CloudWatch), Docker, Docker Compose, GitHub Actions, CI/CD, Vercel, Selenium

Tools: VS Code, Xcode, Figma, LaTeX, Git, Gradle, Discord.py, BeautifulSoup, Typer, ESLint

Activities & Interests

- Endurance sports — Ironman 70.3 finisher (Summer 2024); ongoing swim/run training and structured race prep
- Cycling — training component for triathlon; enjoys exploring local bike trails and endurance rides
- Swimming — pool & open water; technique/metrics nerd; dogfooding SwimMate features on real workouts
  - Consistent swimmer of 16 years specializing in distance freestyle
  - 2x SFHS Swim Team Captain (elected)
  - Over 1 million meters, 2000 hours in pool logged since 2018
- Running — local 5K/10K/HM races; enjoys tempo/interval work and course planning
- Strength training & mobility work — complements endurance training; focus on injury prevention and performance enhancement
- Open-source contributions & hackathons — active in developer community; enjoys collaborative coding and rapid prototyping

- Photography — captures travel moments and outdoor adventures; pairs well with exploration and design sensibility
- Journaling — weekly reflection and goal tracking; personal knowledge base for ideas & projects
- Music production — compose & mix instrumentals; experiment with sound design and arranging
- Video games — systems/strategy and indie titles; interest in game design and UX
- Travel — hike-friendly, budget-conscious trips; itinerary building and exploring local food/culture

Courses

ANTH 140 – Anth of Alcohol
CAS 100B – Effective Speech
CMPEN 270 – Digital Design
CMPEN 331 – Comp Org and Design
CMPEN 431 – Intro Comput Arch
CMPSC 111 – Logic Comp Sci
CMPSC 131 – PROG & COMP I
CMPSC 132 – PROG & COMP II
CMPSC 221 – Oop With Web
CMPSC 263 – Blockchain and Modern Web Dev
CMPSC 311 – Intro Sys Progmg
CMPSC 360 – Discrete Math/Cs
CMPSC 431W – Database Mgmt Syst
CMPSC 461 – Prog Lang Concepts
CMPSC 464 – Intro Theory Comp
CMPSC 465 – Data Struc and Algor
CMPSC 466 – Intro to Quantum Computation
CMPSC 473 – Operating Sys
CMPSC 475 – App Programming
COMM 150N – Cinema Art
CYBER 100S – COMPSYS LIT
ECON 102 – Microeconomic Analysis
EGEE 101 – Energy and Envirnmnt
ENGL 15 – Rhetoric and Comp
ENGL 202C – Technical Writing
ENGL 229 – Digital Studies
GAME 160N – Video Game Culture
GER 2 – Elem German II
KINES 61 – Fit Thry&Practice
MATH 140 – CALC ANLY GEOM I
MATH 141 – CALC ANLY GEOM II
MATH 220 – Matrices
MATH 230 – Calc/Vector Anly
MATH 319 – Elem. Mathematical Statistics
MATH 441 – Matrix Algebra
MATH 451 – Numer Computations
MATH 452 – Deep Learning Algorithms
MATH 486 – Math Thy of Games
MUSIC 8 – Rud of Mus
PHYS 211 – Mechanics
PHYS 212 – Elect. and Mag.
PSYCH 100 – Intro Psychology
STAT 318 – Elem Probability
STAT 319 – Elem. Mathematical Statistics