JAX 90-Day Learning Challenge: GitHub-Driven

Overall Goal: By the end of 90 days, you'll have a solid understanding of JAX fundamentals, built several small JAX-based projects (likely ML-focused), and consistently pushed your learning and code to a dedicated GitHub repository.

Repository Structure Suggestion:

```
your-jax-challenge/
├── day_001_installation_and_basics/
│   └── main.py
│   └── README.md (brief notes on what you learned/did)
├── day_002_jax_vs_numpy/
│   └── main.py
│   └── README.md
├── ...
└── day_090_final_project/
    ├── src/
    ├── models/
    ├── data/
    └── README.md
```

Phase 1: Foundations & Core JAX (Days 1-30)

- **Focus:** Installation, JAX vs. NumPy, jax.jit, jax.grad, basic array manipulation, PRNG.
- **GitHub Focus:** Daily commits with small, self-contained examples for each concept. Each day's commit should demonstrate understanding of a new JAX feature.

Week	Days	Topic/Activity	GitHub Commit Goal
Week 1: Getting Started	1-2	**Day 1: Setup & First JAX Array.** Install JAX. Create	day_001_install_and_array_basics.py (simple array ops)

		your first JAX array. Basic arithmetic.	
	3-4	**Day 2: JAX vs. NumPy.** Replicate some NumPy operations in JAX. Highlight immutability.	day_002_jax_vs_numpy_comparison.py
	5-7	**Day 3-5: Introduction to jax.jit.** Write a simple function, JIT it. Measure performance difference. Experiment with static arguments.	day_003_intro_jit.py, day_004_jit_performance.py, day_005_jit_static_args.py
Week 2: Automatic Differentiation	8-10	**Day 6-8: jax.grad Basics.** Compute gradient of simple scalar functions (e.g., f(x)=x2, f(x)=sin(x)).	day_006_simple_grad.py, day_007_multi_variable_grad.py
	11-14	**Day 9-12: value_and_grad, Higher-order**	day_009_value_and_grad.py, day_010_higher_order_grad.py, day_011_hessian_example.py

		Gradients. Compute both value and gradient. Explore jax.hessian (second derivative).	
Week 3: Randomness & Control Flow	15-18	**Day 13-16: JAX PRNG Keys.** Understand and implement JAX's explicit PRNG key management. Generate random numbers.	day_013_prng_basics.py, day_014_splitting_keys.py, day_015_random_sampling.py
	19-21	**Day 17-19: Control Flow in JAX.** jax.lax.cond, jax.lax.while_loop, jax.lax.fori_loop. How JIT interacts with control flow.	day_017_lax_cond.py, day_018_lax_while_loop.py, day_019_lax_fori_loop.py
Week 4: Basic ML Model	22-25	**Day 20-23: Linear Regression from Scratch.** Implement a basic linear regression model with JAX, including a loss	day_020_linear_regression_setup.py, day_021_loss_and_grad.py, day_022_gd_update.py

		function and gradient descent updates.	
	26-30	**Day 24-28: Logistic Regression (Binary).** Extend to binary logistic regression. Implement sigmoid and cross-entropy loss.	day_024_logistic_regression_setup.py, day_025_sigmoid_and_loss.py, day_026_logistic_gd.py
		Day 29-30: Refactor & Document. Clean up code, add comments, write a good README for the first month's learning.	day_029_refactor_ml_models.py, day_030_documentation_review.md

Phase 2: Advanced JAX & First Project (Days 31-60)

- **Focus:** jax.vmap, jax.pmap (conceptual or simple example), building a small neural network.
- **GitHub Focus:** Continue daily commits. Start building slightly more complex, cohesive scripts that combine JAX features. Aim for a small, end-to-end ML project by the end of this phase.

Week	Days	Topic/Activity	GitHub Commit Goal

Week 5: Vectorization	31-35	**Day 31-35: jax.vmap Deep Dive.** Understand in_axes and out_axes. Vectorize a function that processes multiple inputs. Apply vmap to your linear/logistic regression.	day_031_vmap_basics.py, day_032_vmap_in_out_axes.py, day_033_vmap_ml_model.py
	36-38	**Day 36-38: More vmap Applications.** Consider batching operations, processing multiple neural network layers.	day_036_vmap_mlp_layer.py
Week 6: Parallelization (Conceptual/Simple)	39-42	**Day 39-42: Introduction to jax.pmap (Theory & Simple Example).** Understand the concept of pmap, data sharding. If you have multiple	day_039_pmap_intro.md (for conceptual understanding), day_040_simple_pmap_if_devices.py

		devices, try a simple pmap example. Otherwise, focus on understanding its use cases.	
	43-45	**Day 43-45: Manual Parameter Management in JAX.** How to represent neural network parameters as PyTrees.	day_043_pytree_for_params.py, day_044_init_mlp_params.py
Week 7: Building a Simple Neural Network	46-50	**Day 46-50: Multi-Layer Perceptron (MLP) from Scratch.** Implement a feed-forward neural network with hidden layers, activation functions, and forward pass.	day_046_mlp_architecture.py, day_047_mlp_forward_pass.py
	51-53	**Day 51-53: Training an MLP.**	day_051_mlp_loss_grad.py, day_052_mlp_training_loop.py

		Integrate loss function, gradient computation, and an optimization loop for your MLP. Train on a simple dataset (e.g., MNIST digits subset).	
Week 8: First Mini-Project	54-58	**Day 54-58: Mini-Project: Image Classifier (JAX Core).** Use your MLP to classify a small image dataset (e.g., Fashion MNIST or a subset of CIFAR-10). Focus on using only core JAX.	day_054_image_classifier_setup.py, day_055_data_loading_simple.py, day_056_train_image_classifier.py
	59-60	**Day 59-60: Project Refinement & README.** Clean code, add clear documentation, and write a detailed	day_059_project_refactor.py, day_060_project_readme.md

		README.md for your mini-project.	

Phase 3: JAX Ecosystem & Deeper Dive (Days 61-90)

- **Focus:** Exploring JAX libraries (Equinox, Flax, Optax), more complex model architectures, potential for a small research-inspired project.
- **GitHub Focus:** Build on previous projects, integrate libraries, and potentially work on a final, more substantial project that showcases your JAX skills.

Week	Days	Topic/Activity	GitHub Commit Goal
Week 9: Exploring JAX Ecosystem	61-65	**Day 61-65: Introduction to Equinox.** Learn how to define models and train them with Equinox. Implement your previous MLP using Equinox.	day_061_equinox_intro.py, day_062_equinox_mlp.py
	66-67	**Day 66-67: Introduction to Optax.** Use Optax optimizers (e.g., Adam) with your JAX or Equinox model.	day_066_optax_intro.py, day_067_optax_with_mlp.py

Week 10: Flax or Deeper Equinox	68-72	**Day 68-72: Introduction to Flax (Option A).** Explore Flax modules, Linen API. Implement a simple CNN with Flax. OR **Deeper Equinox (Option B).** Explore more advanced Equinox features (e.g., custom layers, different optimization techniques).	day_068_flax_intro.py, day_069_flax_cnn.py OR day_068_equinox_advanced.py
	73-75	**Day 73-75: Data Loading and Pipelining.** Learn how to efficiently load data for JAX training (e.g., with tf.data or dm-haiku-datasets).	day_073_data_loading_jax.py
Week 11: Application & Project Ideas	76-80	**Day 76-80: Recurrent Neural Networks (RNNs) or Transformers**	day_076_simple_rnn.py OR day_076_attention_block.py

		(Simplified). Explore how to implement simple sequence models in JAX. This could be a small language model or a time-series predictor.	
	81-83	**Day 81-83: JAX for Scientific Computing/Physics (Optional).** Explore a non-ML application of JAX (e.g., solving differential equations, simulating a simple physical system).	day_081_jax_ode_solver.py OR day_081_physics_sim.py
Week 12: Final Project & Review	84-88	**Day 84-88: Final Project.** Choose a slightly more ambitious project. Examples: a small image generative model (e.g., VAE), a reinforcement	day_084_final_project_start.py... daily commits to project folderJAX 90-Day Learning Challenge: GitHub-Driven Overall Goal: By the end of 90 days, you'll have a solid understanding of JAX fundamentals, built several small JAX-based projects (likely ML-focused), and consistently pushed your learning and code to a dedicated GitHub repository.

		learning agent for a simple environment, or a more complex scientific simulation.	Repository Structure Suggestion: your-jax-challenge/ ├── day_001_installation_and_basics/ │ └── main.py │ └── README.md (brief notes on what you learned/did) ├── day_002_jax_vs_numpy/ │ └── main.py │ └── README.md ├── ... └── day_090_final_project/ ├── src/ ├── models/ ├── data/ └── README.md Phase 1: Foundations & Core JAX (Days 1-30) Focus: Installation, JAX vs. NumPy, jax.jit, jax.grad, basic array manipulation, PRNG. GitHub Focus: Daily commits with small, self-contained examples for each concept. Each day's commit should demonstrate understanding of a new JAX feature. Week

			Days Topic/Activity GitHub Commit Goal Week 1: Getting Started 1-2 Day 1: Setup & First JAX Array. Install JAX. Create your first JAX array. Basic arithmetic. day_001_install_and_array_basics.py (simple array ops) 3-4 Day 2: JAX vs. NumPy. Replicate some NumPy operations in JAX. Highlight immutability. day_002_jax_vs_numpy_comparison.py 5-7 Day 3-5: Introduction to jax.jit. Write a simple function, JIT it. Measure performance difference. Experiment

			with static arguments. day_003_intro_jit.py, day_004_jit_performance.py, day_005_jit_static_args.py Week 2: Automatic Differentiation 8-10 Day 6-8: jax.grad Basics. Compute gradient of simple scalar functions (e.g., f(x)=x2, f(x)=sin(x)). day_006_simple_grad.py, day_007_multi_variable_grad.py 11-14 Day 9-12: value_and_grad, Higher-order Gradients. Compute both value and gradient. Explore jax.hessian (second derivative). day_009_value_and_grad.py, day_010_higher_order_grad.py, day_011_hessian_example.py Week 3: Randomness & Control Flow 15-18

			Day 13-16: JAX PRNG Keys. Understand and implement JAX's explicit PRNG key management. Generate random numbers. day_013_prng_basics.py, day_014_splitting_keys.py, day_015_random_sampling.py 19-21 Day 17-19: Control Flow in JAX. jax.lax.cond, jax.lax.while_loop, jax.lax.fori_loop. How JIT interacts with control flow. day_017_lax_cond.py, day_018_lax_while_loop.py, day_019_lax_fori_loop.py Week 4: Basic ML Model 22-25 Day 20-23: Linear Regression from Scratch. Implement a basic linear regression model with JAX, including a loss function and gradient descent updates. day_020_linear_regression_setup.py, day_021_loss_and_grad.py, day_022_gd_update.py

			26-30 Day 24-28: Logistic Regression (Binary). Extend to binary logistic regression. Implement sigmoid and cross-entropy loss. day_024_logistic_regression_setup.py, day_025_sigmoid_and_loss.py, day_026_logistic_gd.py Day 29-30: Refactor & Document. Clean up code, add comments, write a good README for the first month's learning. day_029_refactor_ml_models.py, day_030_documentation_review.md Phase 2: Advanced JAX & First Project (Days 31-60) Focus: jax.vmap, jax.pmap (conceptual or simple example), building a small neural network. GitHub Focus: Continue daily commits. Start building slightly more complex, cohesive scripts that combine JAX features. Aim for a small, end-to-end ML project by the end of this phase. Week

			Days Topic/Activity GitHub Commit Goal Week 5: Vectorization 31-35 Day 31-35: jax.vmap Deep Dive. Understand in_axes and out_axes. Vectorize a function that processes multiple inputs. Apply vmap to your linear/logistic regression. day_031_vmap_basics.py, day_032_vmap_in_out_axes.py, day_033_vmap_ml_model.py 36-38 Day 36-38: More vmap Applications. Consider batching operations, processing multiple neural network layers. day_036_vmap_mlp_layer.py Week 6: Parallelization (Conceptual/Simple)

			39-42 Day 39-42: Introduction to jax.pmap (Theory & Simple Example). Understand the concept of pmap, data sharding. If you have multiple devices, try a simple pmap example. Otherwise, focus on understanding its use cases. day_039_pmap_intro.md (for conceptual understanding), day_040_simple_pmap_if_devices.py 43-45 Day 43-45: Manual Parameter Management in JAX. How to represent neural network parameters as PyTrees. day_043_pytree_for_params.py, day_044_init_mlp_params.py Week 7: Building a Simple Neural Network 46-50 Day 46-50: Multi-Layer Perceptron (MLP) from Scratch. Implement a feed-forward neural network with hidden layers, activation functions, and forward pass.

			day_046_mlp_architecture.py, day_047_mlp_forward_pass.py 51-53 Day 51-53: Training an MLP. Integrate loss function, gradient computation, and an optimization loop for your MLP. Train on a simple dataset (e.g., MNIST digits subset). day_051_mlp_loss_grad.py, day_052_mlp_training_loop.py Week 8: First Mini-Project 54-58 Day 54-58: Mini-Project: Image Classifier (JAX Core). Use your MLP to classify a small image dataset (e.g., Fashion MNIST or a subset of CIFAR-10). Focus on using only core JAX. day_054_image_classifier_setup.py, day_055_data_loading_simple.py, day_056_train_image_classifier.py 59-60

			Day 59-60: Project Refinement & README. Clean code, add clear documentation, and write a detailed README.md for your mini-project. day_059_project_refactor.py, day_060_project_readme.md Phase 3: JAX Ecosystem & Deeper Dive (Days 61-90) Focus: Exploring JAX libraries (Equinox, Flax, Optax), more complex model architectures, potential for a small research-inspired project. GitHub Focus: Build on previous projects, integrate libraries, and potentially work on a final, more substantial project that showcases your JAX skills. Week Days Topic/Activity GitHub Commit Goal Week 9: Exploring JAX Ecosystem 61-65 Day 61-65: Introduction to Equinox. Learn how to

			define models and train them with Equinox. Implement your previous MLP using Equinox. day_061_equinox_intro.py, day_062_equinox_mlp.py 66-67 Day 66-67: Introduction to Optax. Use Optax optimizers (e.g., Adam) with your JAX or Equinox model. day_066_optax_intro.py, day_067_optax_with_mlp.py Week 10: Flax or Deeper Equinox 68-72 Day 68-72: Introduction to Flax (Option A). Explore Flax modules, Linen API. Implement a simple CNN with Flax. OR Deeper Equinox (Option B). Explore more advanced Equinox features (e.g., custom layers, different optimization techniques). day_068_flax_intro.py, day_069_flax_cnn.py OR day_068_equinox_advanced.py 73-75

			Day 73-75: Data Loading and Pipelining. Learn how to efficiently load data for JAX training (e.g., with tf.data or dm-haiku-datasets). day_073_data_loading_jax.py Week 11: Application & Project Ideas 76-80 Day 76-80: Recurrent Neural Networks (RNNs) or Transformers (Simplified). Explore how to implement simple sequence models in JAX. This could be a small language model or a time-series predictor. day_076_simple_rnn.py OR day_076_attention_block.py 81-83 Day 81-83: JAX for Scientific Computing/Physics (Optional). Explore a non-ML application of JAX (e.g., solving differential equations, simulating a simple physical system). day_081_jax_ode_solver.py OR day_081_physics_sim.py Week 12: Final Project & Review

			84-88 Day 84-88: Final Project. Choose a slightly more ambitious project. Examples: a small image generative model (e.g., VAE), a reinforcement learning agent for a simple environment, or a more complex scientific simulation. day_084_final_project_start.py... daily commits to project folder 89-90 Day 89-90: Project Polish & Portfolio Prep. Finalize your code, write a comprehensive README.md for your final project, and prepare your GitHub repository as a portfolio piece. Reflect on your learning journey. day_089_final_project_polish.py, day_090_reflection_and_readme_update.md GitHub Strategy: Dedicated Repository: Create a public GitHub repository named something like jax-90-day-challenge or learning-jax.

			Daily Commits: Aim for at least one meaningful commit every day (or almost every day). Even if it's just a small script or a README.md update with notes, the act of committing regularly reinforces the habit. Clear Commit Messages: Use descriptive commit messages (e.g., "Day 1: Initial JAX installation and array basics," "Day 15: Implemented PRNG key splitting for random sampling"). READMEs for Each Day/Folder: Inside each day_XXX folder, include a README.md that briefly explains what you learned, what the code does, and any challenges or insights. Main Repository README: Keep the main README.md of your repository updated with your overall progress, links to daily challenges, and a summary of what you've achieved. Milestones/Branches (Optional): For larger projects, consider using branches to manage features or milestones. Important Considerations: Flexibility: This is a guide, not a rigid schedule. If a topic takes longer, adjust. If you feel confident, accelerate.

			Breaks: Don't burn out. If you miss a day, just pick up where you left off. The goal is consistent effort, not perfection. Active Learning: Don't just copy-paste. Type out the code, debug it, and try to modify it slightly to test your understanding. Documentation: Refer constantly to the official JAX documentation. Community: If you get stuck, don't hesitate to search for answers or ask questions on forums like Stack Overflow, Reddit (r/MachineLearning, r/deeplearning), or JAX's GitHub issues. Focus on Understanding: JAX is about understanding transformations and the functional paradigm. Don't just get code to run; understand why it works that way. Celebrate Small Wins: Each successful concept learned and each commit pushed is a win! This 90-day challenge will give you a robust foundation in JAX and a fantastic portfolio to showcase your learning journey. Good luck, and have fun with it! This is an excellent, detailed plan! The 90-day

			challenge with a GitHub focus is a perfect way to internalize JAX concepts and build a tangible portfolio. Here are some additions and refinements to make your challenge even more effective: Enhancements to Your 90-Day Challenge Intermediate Checkpoints/Review Days: After each phase (Days 30, 60, 90): Dedicate a day or two to reviewing all the code you've written, refactoring where necessary, and writing a comprehensive summary in a _README.md file for that phase. This helps consolidate learning and ensures your code is clean and understandable for others (and future you!). Mini-Project Reviews: For the mini-project days, include time for testing, basic evaluation metrics, and thorough documentation. Debugging in JAX: Early Introduction: Introduce jax.debug.print and jax.lax.mark_array early on (e.g., around Day 10-15). These are crucial for understanding what's happening inside JIT-compiled functions. Python's print() often won't work as expected with JIT.

			Error Handling: Discuss common JAX errors (e.g., shape mismatches, immutable array errors, issues with non-pure functions) and how to interpret them. Performance Profiling: Basic Timing: Beyond timeit, introduce jax.block_until_ready() when timing JAX operations, as JAX operations are asynchronous. Advanced Profiling (Later Stage): Briefly mention JAX's profiling tools (jax.profiler) for deeper performance analysis, especially when working on the final project. Beyond jax.grad: jax.jit, jax.vmap, jax.grad Composition: Emphasize how these transformations compose. Show examples like jax.jit(jax.vmap(jax.grad(f))). This is where JAX truly shines. Jacobians, Hessians (Revisit): After vmap is covered, revisit jax.jacfwd, jax.jacrev, and jax.hessian to show how to compute full Jacobians and Hessians efficiently. PyTrees and Custom Data Structures: Formal Introduction: Dedicate a specific day to

			PyTrees (e.g., around Day 40-45). Explain how JAX handles arbitrary nested Python structures and how jax.tree_util functions work (tree_map, tree_flatten, tree_unflatten). This is fundamental for managing model parameters in JAX. Benefit: Explain why PyTrees are so powerful for functional programming and transformations. "Sharp Bits" and Best Practices: Dedicated Section/Day: Have a day (maybe mid-challenge, Day 45-50) to discuss JAX's "sharp bits" – common pitfalls and design patterns: Immutability: Reiterate why x = x.at[idx].set(val) is used instead of x[idx] = val. Pure Functions: Stress the importance of pure functions for JAX transformations. Static vs. Traced Arguments: How static_argnums works and its implications for recompilation. Side Effects: Explain why side effects (like print without jax.debug.print) inside JIT'd functions don't behave as expected. Randomness Review: Reinforce the PRNG key management, especially in loops and with vmap.

			Data Handling: Dataset Integration: For your ML projects, briefly touch on integrating with common data loading libraries (e.g., torch.utils.data.DataLoader for PyTorch users, tf.data.Dataset for TensorFlow users). This helps bridge the gap between JAX and existing data pipelines. Project Ideas Expansion: More Diverse Projects: Suggest a wider range of project ideas for the later phases to keep things interesting: Simple Reinforcement Learning: Implement a basic policy gradient method for a small Gym environment. Physics Simulations: A simple N-body simulation, a heat equation solver, or a cellular automaton (e.g., Conway's Game of Life) using JAX's array operations. Optimization Algorithms: Implement different optimization algorithms (e.g., L-BFGS, conjugate gradient) from scratch using JAX's autodiff. Basic Generative Models: A simple Variational Autoencoder (VAE) or Generative Adversarial Network (GAN).

			Refined GitHub Strategy: Detailed READMEs: Ensure each day's README.md (or the README.md for a larger module) clearly states: Goal for the Day: What concept was explored. Key Learnings: 1-3 bullet points summarizing takeaways. Code Explanation: Brief overview of the accompanying Python script. Challenges/Notes: Any difficulties encountered or interesting observations. Version Control Best Practices: Branching: Encourage creating a new branch for each week or major project, then merging back to main. This is good practice for larger projects. Commit Frequency: Reiterate that smaller, more frequent commits are better than large, infrequent ones. Even committing "Day X: Read tutorial on JIT" with just a markdown file is valuable. Showcase Branch (Optional): Towards the end, create

			a showcase branch or a final_projects directory in main where your best work is presented clearly for a portfolio. Motivational Tips: Share Your Progress: Post on social media (LinkedIn, Twitter/X) about your daily progress using a consistent hashtag (e.g., #JAX90DaysChallenge). This builds public accountability and connects you with others learning JAX. Find a Study Buddy: If possible, find someone else interested in JAX to learn alongside you. You can review each other's code and discuss concepts. Set Rewards: Small, personal rewards for hitting milestones (e.g., after 30, 60, 90 days). Visualize Progress: GitHub's contribution graph is a great visual motivator. Seeing that green grid fill up is satisfying! By incorporating these points, your 90-day JAX challenge will be incredibly structured, comprehensive, and effective for both learning and building a valuable portfolio. Good luck!
	89-90	**Day 89-90:**	day_089_final_project_polish.py,

		Project Polish & Portfolio Prep. Finalize your code, write a comprehensive README.md for your final project, and prepare your GitHub repository as a portfolio piece. Reflect on your learning journey.	day_090_reflection_and_readme_update.md

GitHub Strategy:

1. **Dedicated Repository:** Create a public GitHub repository named something like jax-90-day-challenge or learning-jax.
2. **Daily Commits:** Aim for at least one meaningful commit every day (or almost every day). Even if it's just a small script or a README.md update with notes, the act of committing regularly reinforces the habit.
3. **Clear Commit Messages:** Use descriptive commit messages (e.g., "Day 1: Initial JAX installation and array basics," "Day 15: Implemented PRNG key splitting for random sampling").
4. **READMEs for Each Day/Folder:** Inside each day_XXX folder, include a README.md that briefly explains what you learned, what the code does, and any challenges or insights.
5. **Main Repository README:** Keep the main README.md of your repository updated with your overall progress, links to daily challenges, and a summary of what you've achieved.
6. **Milestones/Branches (Optional):** For larger projects, consider using branches to manage features or milestones.

Important Considerations:

- **Flexibility:** This is a guide, not a rigid schedule. If a topic takes longer, adjust. If you feel confident, accelerate.
- **Breaks:** Don't burn out. If you miss a day, just pick up where you left off. The goal is consistent effort, not perfection.
- **Active Learning:** Don't just copy-paste. Type out the code, debug it, and try to modify it slightly to test your understanding.
- **Documentation:** Refer constantly to the official JAX documentation.
- **Community:** If you get stuck, don't hesitate to search for answers or ask questions on forums like Stack Overflow, Reddit (r/MachineLearning, r/deeplearning), or JAX's GitHub issues.
- **Focus on Understanding:** JAX is about understanding transformations and the functional paradigm. Don't just get code to run; understand *why* it works that way.
- **Celebrate Small Wins:** Each successful concept learned and each commit pushed is a win!

This 90-day challenge will give you a robust foundation in JAX and a fantastic portfolio to showcase your learning journey. Good luck, and have fun with it!
This is an excellent, detailed plan! The 90-day challenge with a GitHub focus is a perfect way to internalize JAX concepts and build a tangible portfolio. Here are some additions and refinements to make your challenge even more effective:

Enhancements to Your 90-Day Challenge

1. **Intermediate Checkpoints/Review Days:**
   - **After each phase (Days 30, 60, 90):** Dedicate a day or two to reviewing all the code you've written, refactoring where necessary, and writing a comprehensive summary in a _README.md file for that phase. This helps consolidate learning and ensures your code is clean and understandable for others (and future you!).
   - **Mini-Project Reviews:** For the mini-project days, include time for testing, basic evaluation metrics, and thorough documentation.

2. **Debugging in JAX:**
    - **Early Introduction:** Introduce jax.debug.print and jax.lax.mark_array early on (e.g., around Day 10-15). These are crucial for understanding what's happening inside JIT-compiled functions. Python's print() often won't work as expected with JIT.
    - **Error Handling:** Discuss common JAX errors (e.g., shape mismatches, immutable array errors, issues with non-pure functions) and how to interpret them.
3. **Performance Profiling:**
    - **Basic Timing:** Beyond timeit, introduce jax.block_until_ready() when timing JAX operations, as JAX operations are asynchronous.
    - **Advanced Profiling (Later Stage):** Briefly mention JAX's profiling tools (jax.profiler) for deeper performance analysis, especially when working on the final project.
4. **Beyond jax.grad:**
    - **jax.jit, jax.vmap, jax.grad Composition:** Emphasize how these transformations compose. Show examples like jax.jit(jax.vmap(jax.grad(f))). This is where JAX truly shines.
    - **Jacobians, Hessians (Revisit):** After vmap is covered, revisit jax.jacfwd, jax.jacrev, and jax.hessian to show how to compute full Jacobians and Hessians efficiently.
5. **PyTrees and Custom Data Structures:**
    - **Formal Introduction:** Dedicate a specific day to PyTrees (e.g., around Day 40-45). Explain how JAX handles arbitrary nested Python structures and how jax.tree_util functions work (tree_map, tree_flatten, tree_unflatten). This is fundamental for managing model parameters in JAX.
    - **Benefit:** Explain why PyTrees are so powerful for functional programming and transformations.
6. **"Sharp Bits" and Best Practices:**
    - **Dedicated Section/Day:** Have a day (maybe mid-challenge, Day 45-50) to discuss JAX's "sharp bits" – common pitfalls and design patterns:
        - **Immutability:** Reiterate why x = x.at[idx].set(val) is used instead of x[idx] = val.
        - **Pure Functions:** Stress the importance of pure functions for JAX transformations.
        - **Static vs. Traced Arguments:** How static_argnums works and its implications for recompilation.
        - **Side Effects:** Explain why side effects (like print without jax.debug.print) inside JIT'd functions don't behave as expected.
    - **Randomness Review:** Reinforce the PRNG key management, especially in loops and with vmap.
7. **Data Handling:**

    - **Dataset Integration:** For your ML projects, briefly touch on integrating with common data loading libraries (e.g., torch.utils.data.DataLoader for PyTorch users, tf.data.Dataset for TensorFlow users). This helps bridge the gap between JAX and existing data pipelines.
8. **Project Ideas Expansion:**
    - **More Diverse Projects:** Suggest a wider range of project ideas for the later phases to keep things interesting:
        - **Simple Reinforcement Learning:** Implement a basic policy gradient method for a small Gym environment.
        - **Physics Simulations:** A simple N-body simulation, a heat equation solver, or a cellular automaton (e.g., Conway's Game of Life) using JAX's array operations.
        - **Optimization Algorithms:** Implement different optimization algorithms (e.g., L-BFGS, conjugate gradient) from scratch using JAX's autodiff.
        - **Basic Generative Models:** A simple Variational Autoencoder (VAE) or Generative Adversarial Network (GAN).

Refined GitHub Strategy:

- **Detailed READMEs:** Ensure each day's README.md (or the README.md for a larger module) clearly states:
    - **Goal for the Day:** What concept was explored.
    - **Key Learnings:** 1-3 bullet points summarizing takeaways.
    - **Code Explanation:** Brief overview of the accompanying Python script.
    - **Challenges/Notes:** Any difficulties encountered or interesting observations.
- **Version Control Best Practices:**
    - **Branching:** Encourage creating a new branch for each week or major project, then merging back to main. This is good practice for larger projects.
    - **Commit Frequency:** Reiterate that smaller, more frequent commits are better than large, infrequent ones. Even committing "Day X: Read tutorial on JIT" with just a markdown file is valuable.
- **Showcase Branch (Optional):** Towards the end, create a showcase branch or a final_projects directory in main where your best work is presented clearly for a portfolio.

Motivational Tips:

- **Share Your Progress:** Post on social media (LinkedIn, Twitter/X) about your daily progress using a consistent hashtag (e.g., #JAX90DaysChallenge). This builds public accountability and connects you with others learning JAX.
- **Find a Study Buddy:** If possible, find someone else interested in JAX to learn alongside you. You can review each other's code and discuss concepts.
- **Set Rewards:** Small, personal rewards for hitting milestones (e.g., after 30, 60, 90 days).
- **Visualize Progress:** GitHub's contribution graph is a great visual motivator. Seeing that green grid fill up is satisfying!

By incorporating these points, your 90-day JAX challenge will be incredibly structured, comprehensive, and effective for both learning and building a valuable portfolio. Good luck!