Research Protocols & Standard Operating Procedures 🛡️

"Slow is Smooth, Smooth is Fast."

This document defines the engineering standards required to contribute to this project. Violations of these protocols lead to "Code Rot" and are unacceptable.

---

1. The "Experiment-First" Architecture 🧪

We do not use a monolithic `src/` folder. We use an **Experiment-First** structure.

The Standard Layout

Every experiment MUST live in `experiments/v{VERSION}_{DESCRIPTION}/`.
Inside that folder, it MUST be self-contained:

- `models/`: A COPY of the model file used. Do not import from previous versions.
- `logs/`: Training logs.
- `audits/`: JSON results and validation reports.
- `tests/`: Unit tests and diagnostic scripts specific to this version.
- `journal.md`: The narrative log of the experiment (The "Captain's Log").
- `train.py`: The entry point script.

> [!IMPORTANT]
> **The Journal is the Single Source of Truth**
> Everything about an experiment should be traceable from `journal.md`:
>
> - What was tried (task breakdown)
> - Why it was tried (hypothesis, methods)
> - What happened (results, observations)
> - What was learned (walkthroughs, proof of work)
> - What's next (recommendations)
>
> When you finish an experiment, merge `task.md` and `walkthrough.md` into journal sections. Future agents should only need to read `journal.md` to understand the complete story.

Why?
This allows us to delete or archive any old experiment without breaking the current SOTA. It decouples progress.

B. The Global Root Strategy 🌍

While `experiments/` is for *change*, the Root is for *permanence*.

- **`scripts/`**: **V-Agnostic Utilities**. Data preparation, global plotting, or repo maintenance tools. (If it imports a specific model, it belongs in `experiments/`).

- `docs/`: **Deep Knowledge**. Mathematical proofs, architectural whitepapers, and guides that outlive any single version.
- `data/`: **Immutable Source**. Raw datasets. Experiments treat this as Read-Only.
- `configs/`: **Shared Baselines**. Global configuration constants (if needed).
- `final_report/`: **The Publication**. When a milestone is reached, polished figures and summaries are copied here for external consumption (e.g. arXiv/GitHub pages).

C. Strategic Continuity (The Roadmap) 🗺️

While `journal.md` is for the **Experiment**, `ROADMAP.md` is for the **Project**.

- **Requirement**: Every project MUST have a `ROADMAP.md` in the root (initialized from `ROADMAP_TEMPLATE.md`).
- **The Protocol**:
    1. **Planning**: Before starting a new version, check the active Phase in the roadmap.
    2. **Execution**: If an experiment triggers a strategic pivot, update the roadmap immediately.
    3. **Handoff**: Before closing a session, update the Roadmap's Phase table with your findings.

Why? To prevent "Short-Sighted Agent" syndrome. The Roadmap ensures results aren't just logged, but are integrated into the project's long-term trajectory.

---

2. The Agentic Workflow 🤖

A. Matrix Mode (Deep Work)

When entering a complex task:

1. **Plan**: Create `implementation_plan.md`.
2. **Execute**: Write code, strictly following the plan.
3. **Verify**: Run `audit.py` or `tests/`.
4. **Document**: Update `journal.md` and `walkthrough.md`.

B. The "Mini-Train" Rule

Before launching a full run:

1. Run the training script for **200 steps** (or 1 epoch on small data).
2. **Verify Convergence**: Does loss go down?
3. **Verify Artifacts**: Are checkpoints saving? Are logs writing?
   Never launch a 2-day job without a 2-minute test.

C. The "Mini-Audit" Rule

Before claiming success or running a benchmark:

1. **Constraint**: Must run in **< 5 minutes**.
2. **Task**: A simplified, synthetic version of the Hard Task (e.g. "Toy Recall" vs "Book Recall").
3. **Threshold**: Must hit **100% Accuracy** (or equivalent 0.0 Loss).
   If it fails the Toy Task, it will fail the Real Task. Fail fast.

---

3. Code Hygiene 🧹

- **Explicit Imports**: Avoid `from model import *`.
- **Config Separation**: Hyperparameters should be at the top of the script or in a config object, not buried in loops.
- **Assertion Guardrails**: Use `assert` statements aggressively to validate tensor shapes at the start of `forward()`.

4. Final Report Protocol 📝

When an experiment concludes:

1. **Success**: Mark as SOTA in `LEADERBOARD.md`. Merge findings into `README.md`.
2. **Failure**: Document *why* in `journal.md`. Move folder to `archive/` if it is clutter.
3. **Handoff**: Write a summary in `journal.md` for the next agent.

---

5. Git Hygiene & Version Control 🐙

A. The "Heavy File" Ban

Never commit:

- Checkpoints (`.pt`, `.ckpt`)
- Large Logs (`.log`, `.txt` > 1MB)
- Datasets (`.bin`, `.jsonl`)
  Why?: Bloats the repo size forever. Use `.gitignore` religiously.

B. The "folder-as-branch" Strategy

In this research repo, we prefer **New Folders (`experiments/vXX`)** over **Git Branches** for experimental variations.

- **Benefit**: You can diff `v1/models/model.py` vs `v2/models/model.py` directly in the editor.
- **Benefit**: Multiple experiments can run simultaneously on the same machine without switching git branches (which would break running jobs).

C. The "Clean Start" Rule

Before launching a training run:

1. **Commit your code**.
2. Log the **Git Hash** in your `journal.md` or training log.
   This ensures that if the run is a success 3 days later, you know EXACTLY what code produced it.

---

6. Agent Cleanup Protocol 🧹

When: After completing a discrete task (e.g., reorganization, audit, migration).

A. Definition: "Stub Files"

Files created **only** to perform a one-time operation:

- `cleanup_*.py` (e.g., `cleanup_experiments.py`)
- `audit_structure.py` (after generating the report)
- `migrate_*.py`
- `temp_*.py`

NOT stub files:

- `train.py`, `audit.py` (persistent entry points)
- Scripts in `experiments/vXX/tests/` (diagnostic tools)
- Scripts in `scripts/` (reusable utilities)

B. The Rule

Before calling `notify_user` to complete a task:

1. Identify stub files created during this session.
2. Delete them: `rm cleanup_experiments.py audit_structure.py`
3. Commit the clean state.

C. Rationale

- **Avoid Clutter**: The root should contain **only** persistent infrastructure.
- **Prevent Confusion**: Future agents won't mistakenly re-run stale cleanup scripts.
- **Signal Completion**: A clean directory signals "this task is done."

D. Artifact Migration

At the end of an experiment, migrate planning/verification artifacts from `/artifacts/` to the experiment folder:

Step 1: Move Artifacts

```
# If you created task.md or walkthrough.md in artifacts directory
mv /path/to/artifacts/task.md experiments/v68/task.md
mv /path/to/artifacts/walkthrough.md experiments/v68/walkthrough.md
```

Step 2: Merge into journal.md (Recommended)
Instead of keeping separate files, merge content into `journal.md` sections:

```
# experiments/v68/journal.md

## Task Breakdown
[Paste content from task.md here]

## Walkthrough
[Paste content from walkthrough.md here - proof of work, validation]
```

Then delete standalone files: `rm experiments/v68/task.md experiments/v68/walkthrough.md`

Alternative: Keep Separate + Symlink (If you need them accessible from root)

```
# Keep originals in experiment folder
# Create symlinks in artifacts for convenience
ln -s $(pwd)/experiments/v68/task.md /path/to/artifacts/task_v68.md
```

Why merge? Single source of truth, no sync issues, complete narrative in one file.

---

7. Research Methodology Protocols 🔬

A. The Smoke Test Mandate

When: Creating any script with **moderate or higher dependency complexity**.

Complexity Indicators:

- Inherits from multiple classes
- Heavily parametrized (> 5 config variables)
- Interacts with GPU/CUDA
- Loads external models or data

The Rule: Create `tests/smoke_test_[FEATURE].py` alongside the main script.

Example:

```
# tests/smoke_test_v68_lane_unification.py
import torch
from models.hybrid_transformer_v68 import HybridTransformerV68

def test_initialization():
    model = HybridTransformerV68(vocab_size=100, hidden_size=64)
    assert model is not None

def test_forward_pass():
    model = HybridTransformerV68(vocab_size=100, hidden_size=64)
    x = torch.randint(0, 100, (2, 10))  # Batch=2, Seq=10
    logits, _ = model(x)
    assert logits.shape == (2, 10, 100)

if __name__ == "__main__":
    test_initialization()
    test_forward_pass()
    print("✅ All smoke tests passed.")
```

Why: Catches import errors, shape bugs, and device mismatches **before** launching a 2-day training run.

B. The Metrics Constitution

Every project must define its **North Star Metrics** early and evaluate them **consistently**.

Step 1: Define
Create `METRICS_METHODOLOGY.md` specifying:

- **Core Metrics** (e.g., Recall, Gate Density, Throughput)
- **Success Thresholds** (e.g., Recall > 90%, Density < 15%)
- **Evaluation Protocol** (e.g., batch size, context length, hardware)

Step 2: Enforce via Class Inheritance
Build a base audit class that all experiments inherit:

```python
# experiments/base_audit.py
class BaseAudit:
    def __init__(self, model, dataset):
        self.model = model
        self.dataset = dataset

    def compute_recall(self):
        raise NotImplementedError

    def compute_density(self):
        raise NotImplementedError

    def run_full_audit(self):
        results = {
            "recall": self.compute_recall(),
            "density": self.compute_density(),
            # ... other metrics
        }
        return results

# experiments/v68_lane_unification/audit_v68.py
from experiments.base_audit import BaseAudit

class AuditV68(BaseAudit):
    def compute_recall(self):
        # V68-specific recall logic
        pass
```

Why: Ensures every experiment reports the **same metrics** in the **same format**, enabling apples-to-apples comparison in `LEADERBOARD.md`.

C. The Principled Problem-Solving Framework

When you hit a major architectural or algorithmic impasse:

1. Establish Canonical Wisdom

- **Survey the Literature**: Find 3-5 foundational papers addressing similar problems.
- **Document Best Practices**: In a `docs/[TOPIC].md` file (e.g., `docs/gradient_flow.md`, `docs/memory_gating.md`).
- **Extract Heuristics**: E.g., "Weight Update Ratio should be ~10^-3" (from DEBUGGING_GUIDE.md).

2. Ground Your Approach

- **Cite Inspiration**: "Our regret gating is inspired by Hindsight Experience Replay (Andrychowicz et al., 2017)."
- **Reference Implementations**: "We adapted the attention masking from Transformer-XL's official repo."
- **Benchmark Against Empirical Results**: "Paper X reports 85% recall at 100k context. Our baseline must exceed this."

3. Document in the Journal

Every experiment's `journal.md` **must** include a **Methods** section:

```
## Methods

### Architectural Decisions
1.  **Lane Unification**: Inspired by [Paper X], we merge Lexical/Semantic
lanes using Centroid-based addressing.
2.  **Gating Logic**: Following [Paper Y]'s finding that surprisal-based
gates reduce redundancy, we implement...

### Key References
-   [Andrychowicz et al., 2017] Hindsight Experience Replay
-   [Wu et al., 2022] Memorizing Transformers
-   [Bulatov et al., 2023] Recurrent Memory Transformer

### Deviations from Literature
We use Z-score normalization instead of raw entropy because [REASON].
```

Why:

- **Reproducibility**: Future agents (or humans) can understand your reasoning.
- **Credibility**: Reviewers see you're not reinventing the wheel blindly.
- **Velocity**: You avoid dead-ends that prior work already explored.

D. The First-Stop-Shop Rule

Before implementing a new feature:

1. **Check the paper's official repo** (if available).
2. **Run their test suite** to understand expected behavior.
3. **Copy their hyperparameters** as your baseline.
4. **Only diverge** if you have a hypothesis for why your approach is better.

Anti-Pattern: "I'll just implement it from scratch based on the math in the paper."
Result: 3 days of debugging shape errors that the official repo already solved.

Correct Pattern: "I cloned their repo, confirmed it works, adapted their `forward()` to my use case."
Result: Working implementation in 2 hours.

---

7. Multi-Environment & Hardware Layout 🖥️☁️

We operate in a **Hybrid Compute** environment.

A. The Agent Environment (GCP Compute Engine / L4)

- **Role**: **Home Base**. Agents live here.
- **Capabilities**: Coding, Logic, Mini-Trains (L4 GPU, **22GB vRAM**).
- **Rule**: ALL code is written and versioned here.

B. The external "Heavy Lift" (Google Colab / A100 Cluster)

- **Role**: **Muscle**. User manually runs massive jobs here.
- **Capabilities**: High-RAM Training (A100 GPU, **80GB vRAM**).
- **Process**:
  1. **Export**: User zips the code from Agent Env.
  2. **Train**: User runs job on Colab.
  3. **Import**: User uploads logs/checkpoints back to Agent Env via `gdown`.

C. The "Airlock" Protocol

When handling external results:

1. **Never** manually edit code on Colab and forget to update the Agent.
2. **Always** bring the `log.txt` back to `experiments/vXX/logs/` so the Agent can analyze it.