# Agent Experiment Workflow Cheat Sheet 📋

**⚠️ CRITICAL: Load this file into your context at session start using `view_file`**

You are an AI agent conducting ML research. This cheat sheet defines your workflow protocol. Follow it precisely to maintain experiment continuity across sessions.

---

## 🚨 MANDATORY: Prime Your Context Window

**Before calling ANY other tools**, you MUST load these files using `view_file`:

1. **`README.md`** - Understand current SOTA and project objectives
2. **`ROADMAP.md`** - Know which phase you're in and what success looks like
3. **`experiments/v[PREVIOUS]_[NAME]/journal.md`** - Read what the previous agent learned
4. **`PROTOCOLS_TEMPLATE.md`** (optional) - If you need detailed workflow guidance

**Why this matters**: Without this context, you will:

- Repeat failed experiments
- Break existing functionality
- Miss critical architectural constraints
- Waste hours debugging known issues

**Your context window is your working memory. Load the right files first.**
(optional but recommended)

---

## Your 4-Phase Workflow

### Phase 1: PLANNING 🗺️

**Call these tools**:

- `write_to_file`: Create `experiments/v[XX]_[DESCRIPTION]/journal.md`
- `search_web`: Find 3-5 papers related to your hypothesis
- Document methods in journal (cite papers, explain deviations)

**Decision point**: Is this a complex task (>500 lines, multi-day)?

- YES → Create `implementation_plan.md`, call `notify_user` for review before starting
- NO → Proceed to execution

### Phase 2: EXECUTION 🛠️

**Before writing ANY code**:

- `view_code_item`: Copy model from previous version (don't import)
- If complex: Write `tests/smoke_test_[FEATURE].py` FIRST
- `run_command`: Execute smoke test before training

**Before launching training**:

- `run_command`: `git add -A && git commit -m "V[XX]: [description]"`
- Log git hash in journal
- **Mini-train FIRST**: `run_command` with `--steps 200 --batch_size 1`
- Monitor: Does loss decrease? Do checkpoints save?

**Resource awareness**:

- L4: 22GB VRAM
- A100: 80GB VRAM
- If model+batch > 20GB → Don't run on L4

### Phase 3: VERIFICATION ✅

**Hard requirement**: Mini-audit (<5 mins, 100% toy task accuracy)

- If fails → STOP. Call `notify_user`. Do not proceed.
- If passes → Run full audit

**Tools to use**:

- `run_command`: Execute audit script
- `write_to_file`: Save results to `audits/[NAME]_[TIMESTAMP].json`
- `replace_file_content`: Update `LEADERBOARD.md` if new high score

### Phase 4: DOCUMENTATION 📝

**Required file edits** (use `replace_file_content` or `multi_replace_file_content`):

1. Update `journal.md`: Add results, methods, conclusion
2. Update `ROADMAP.md`: Mark experiment complete, add learnings
3. Migrate artifacts: Merge `task.md`/`walkthrough.md` into journal sections
4. Move media: `run_command`: `mv *.png experiments/vXX/journal_assets/`

**Cleanup** (use `run_command`):

```
rm cleanup_*.py audit_structure.py temp_*.py
git add -A && git commit -m "V[XX]: Complete"
```

---

## When Things Go Wrong (Emergency Protocols)

### Training Crashes

**Tools to call**:

1. `run_command`: `nvidia-smi` → Check VRAM
2. `run_command`: `df -h` → Check disk space
3. `view_file`: Read last 50 lines of log

4. `replace_file_content`: Document failure in `journal.md`

**Auto-retry logic**:

- Crash #1: Reduce batch size by 50%, retry
- Crash #2: Check if checkpoint is corrupt, resume from earlier
- Crash #3: Call `notify_user` with error details

### Metrics Fail Validation

**Decision tree**:

- Dictionary Recall <100%? → Architecture is broken. `notify_user` immediately. Do NOT proceed.
- Recall dropping? → `view_code_item`: Check gating logic, thresholds
- Throughput tanks? → `run_command`: Profile with `torch.profiler`

### When to Call `notify_user`

**YOU MUST notify if**:

- Stuck on same error >2 hours (you've exhausted your debugging strategies)
- Results contradict hypothesis AND you can't explain why (need human insight)
- Architecture decision affects research direction (need strategic approval)
- Training crashes >3 times despite fixes (systematic issue)

**DO NOT notify for**:

- Syntax errors (use `replace_file_content` and fix)
- Import errors (use `run_command`: `pip install`)
- Expected experiment failures (document in journal, proceed to next experiment)

**The message format**:

- Be concise: State the blocker in 1-2 sentences
- Provide context: What you tried, what failed
- Ask specific question: "Should I pivot to approach X or continue debugging Y?"

---

## Key Metrics to Always Report

| Metric | Target | Definition |
|---|---|---|
| Dictionary Recall | 100% | Bit-perfect on synthetic stress test |
| Rare Recall | >85% | Tokens with ID >5000 |
| Common Recall | >70% | Tokens with ID ≤5000 |
| Semantic Recall | >99% | Cosine sim >0.99 |
| Gate Density | <15% | % of tokens written to memory |
| Throughput | >50k | Tokens/sec during inference |

---

## Git Hygiene Essentials

- **Never commit**: `.pt`, `.ckpt`, `.log`, datasets
- **Use folder-as-branch**: Create `v[XX]_new_idea/` instead of git branches
- **Commit before train**: Log git hash in journal
- **Tag successes**: `git tag v68_sota`

---

## Quick Commands

```
# Check resources
nvidia-smi          # VRAM
df -h               # Disk space

# Testing
python tests/smoke_test_v68.py
python train_v68.py --steps 200 --batch_size 1  # Mini-train

# Cleanup
rm cleanup_*.py audit_structure.py temp_*.py
git add -A && git commit -m "V68: Lane unification complete"
```

---

## Cross-References to Detailed Protocols

For in-depth guidance, see:

- **[PROTOCOLS_TEMPLATE.md]** - Full workflow, Matrix Mode, Principled Problem-Solving, Research Methodology
- **[DEVOPS_TEMPLATE.md]** - Hardware switching (L4↔A100), Multi-environment, Jupyter, Cost management, SSH troubleshooting
- **[METRICS_TEMPLATE.md]** - Metric definitions, thresholds, audit battery
- **[DEBUGGING_TEMPLATE.md]** - Common ML pathologies, gradient issues, NaN debugging

**For operational details:**

- Data management → See DEVOPS Section 4
- Checkpoint strategy → See DEVOPS Section 4
- External dependencies → See DEVOPS Section 3
- Failure recovery → See PROTOCOLS Section 2

---

**Remember**: This workflow prevents "Hope-Driven Development." If you skip steps, you're flying blind.

**The Agentic Override**: You MAY deviate from protocols if:

1. Deviation accelerates progress
2. You document it in `journal.md`
3. Doesn't compromise data integrity or reproducibility