Надёжный “склад результатов” для AI в Cursor: как перестать терять лучшие модели и логи

Главный вывод: организуйте единый протокол экспериментов и файловую структуру с автоматической версификацией, immutable-логами и “best-checkpoint” правилами. Заставьте ИИ строго следовать этим правилам через явные системные инструкции и автоматические pre/post hooks (скрипты), а критические операции (сохранение артефактов) дублируйте и проверяйте. Ниже — готовый план и шаблоны.

1) Базовая файловая структура проекта

Предложение (адаптируйте пути под ваш репозиторий):

- /experiments
  - /configs
    - exp-YYYYMMDD-HHMM.json|yaml — полная спецификация запуска
  - /runs
    - /exp-YYYYMMDD-HHMM-<slug>
      - logs/ (все логи процесса + stdout/stderr)
      - metrics.csv (строка/эпоха → метрики)
      - artifacts/
        - checkpoints/
          - step-000010.pt
          - step-000050.pt
          - best.pt (симлинк/копия на лучший)
        - reports/
          - summary.md (краткий отчёт)
          - charts/ (графики метрик)
      - config.lock.json (фактическая конфигурация запуска)
      - manifest.json (список артефактов, их хэши и пути)
- /best
  - taskA-latest/
    - best.pt
    - metrics.json

        - README.md (критерии выбора, дата)
- /registry
    - models.csv (реестр всех моделей и метрик, версионирование)
    - artifacts.csv (артефакты, хэши, размеры, путь)

Ключевые принципы:

- Каждый запуск неизменяем (immutable): новая папка /runs/exp-...; ничего не затирается.
- Глобальная “витрина” лучших результатов: /best/<task>-latest/ — всегда обновляется только если новый результат лучше по заданному критерию.

2) Строгий протокол экспериментов

Задайте в проекте единый spec формата YAML/JSON (для Spec-Kit) и закрепите правила в README + system prompts для Cursor:

- Идентификатор эксперимента: timestamp + краткий slug (автоматически).
- Конфиг должен включать:
    - dataset, seed, hyperparams, архитектура, критерий “лучше” (например: val_loss ↓, val_acc ↑, PSNR ↑).
    - политика чекпоинтов: save_every_n, keep_last_k, keep_best.
    - пути: output_root, experiment_dir, registry_dir, best_dir.
- Обязательные артефакты на выходе:
    - metrics.csv (включая колонку is_best)
    - manifest.json (перечень артефактов с SHA256)
    - summary.md (что запускалось, лучшие значения, где артефакты)
- Правило обновления /best: обновлять только если метрика улучшилась, иначе игнор.

Пример минимального конфига (yaml):

```
experiment:
  name: "denoise-fft"
  id: "auto"            # автогенерация YYYYMMDD-HHMM-<slug>
  seed: 42
data:
  dataset: "mydata/v1"
model:
  arch: "unet-fft-v3"
train:
  epochs: 50
  batch_size: 32
  lr: 3e-4
  save_every_n: 5
  keep_last_k: 3
  keep_best: true
metrics:
```

```
  primary: "val_psnr"
  mode: "max"           # max = лучше, min = лучше
paths:
  output_root: "./experiments"
  registry_dir: "./registry"
  best_dir: "./best/task-denoise"
logging:
  level: "INFO"
  csv_interval: 1
  tensorboard: true
```

3) Авто-скрипты pre-run/post-run (hooks)

Сделайте два служебных скрипта, которые ИИ обязан вызывать:

- scripts/pre_run.py
  - генерирует ID запуска, создаёт папки, пишет config.lock.json;
  - открывает metrics.csv и пишет заголовки колонок;
  - инициирует manifest.json.
- scripts/post_run.py
  - финализирует manifest (скан артефактов с SHA256);
  - вычисляет лучшую метрику по primary;
  - обновляет /registry/*.csv (добавляет строку с результатами);
  - при необходимости обновляет /best/<task>-latest (только если улучшение);
  - пишет summary.md.

Пример команды запуска (в Makefile или npm scripts):

```
python scripts/pre_run.py --config configs/exp-...yaml
python train.py --config configs/exp-...yaml --out_dir $(EXPERIMENT_DIR)
python scripts/post_run.py --exp_dir $(EXPERIMENT_DIR)
```

Важно: никаких “свободных” путей внутри train.py — только те, что переданы через config.lock.json или переменные окружения.

4) Жёсткие инструкции для ИИ в Cursor (system/developer prompts)

Добавьте в Cursor следующие правила как закреплённые “инструкции проекта” (в Spec-Kit/MemoryBank и в .cursor/rules.md):

- Никогда не перезаписывать артефакты в /runs/ — только создавать новую директорию запуска.
- Всегда вызывать scripts/pre_run.py перед любыми вычислениями.
- Всегда логировать метрики в CSV и STDOUT. Запрещено выводить только “финальный результат” без промежуточных логов.

- При сохранении модели:
  - сохранять шаговые checkpoint’ы в artifacts/checkpoints/step-XXXXX.pt согласно save_every_n;
  - при улучшении primary-метрики сохранять artifacts/checkpoints/best.pt (копия, не симлинк — на случай перемещений).
- Запрещено удалять или перезаписывать best.pt напрямую; его обновляет только post_run.py по критерию.
- Любые “лучшие результаты” всегда зеркалируются в /best/...; если нет улучшения — /best остаётся без изменений.
- Любая модификация политики сохранений — только через конфиг и только перед запуском.
- Любые “очистки” (cleanup) — только в /runs/ конкретного запуска и только через scripts/cleanup.py с dry-run отчётом.

Пример короткого system prompt для Cursor:
“Всегда выполняй эксперименты через pre_run → training → post_run. Никогда не затирай файлы в /runs/. Лучшие модели записывай как artifacts/checkpoints/best.pt внутри запуска. Глобальный best обновляет только post_run по метрике metrics.primary. Весь вывод логируй в runs/<id>/logs и metrics.csv.”

5) MemoryBank и Spec-Kit: как закрепить память

- В MemoryBank сохрани:
  - шаблон конфига;
  - шаблон prompts с правилами неизменяемости артефактов;
  - список “критических путей” (output_root, best_dir, registry_dir);
  - “канонические” функции/скрипты (pre_run, post_run).
- В Spec-Kit опиши:
  - schema конфигов (валидируем через jsonschema/yamale);
  - требование, что train.py читает только config.lock.json (признак воспроизводимости);
  - правила версионирования (semver для модели/кода).

6) Технические детали против “забывания”

- Зафиксируйте конфигурацию в файле config.lock.json, который генерирует pre_run.py (копия исходного конфига + автоматически вычисленные поля: experiment_id, таймстемп, пути).
- Всегда пишите артефакты строго по путям из lock-файла (не из окружения редактора).
- Логи:
  - файлный логгер + stdout; ротация логов; отдельный errors.log.
  - дублирование ключевых метрик в metrics.csv.

- Метрики:
  - CSV-формат: step, epoch, train_loss, val_loss, val_acc, primary, is_best, timestamp.
- Реестр:
  - registry/models.csv: experiment_id, model_path, primary_value, commit_hash, config_hash, data_version.
  - Периодически валидируйте “битые” артефакты (скрипт verify_artifacts.py по SHA256).

7) Как “заставить ИИ выполнять”

- В корень проекта добавьте .cursor/guardrails.md с проверочным чек-листом, который ИИ обязан проходить перед генерацией/изменением кода:
  - Создан ли pre_run? Запишет ли он config.lock.json?
  - Пишутся ли метрики в CSV?
  - Не перезаписываются ли файлы в /runs?
  - Есть ли обновление /best только через post_run?
- Используйте “instruction headers” в файлах, например:

```
# RULES:
# - Do not write outside EXPERIMENT_DIR
# - Save checkpoints only in artifacts/checkpoints
# - Update best only via post_run.py
```

- Добавьте unit-тесты на:
  - корректность путей сохранения,
  - поведение при улучшении/неулучшении метрики,
  - формат metrics.csv и manifest.json.

8) Минимальные кодовые сниппеты

Шаблон логгера:

```
import logging, sys, os
def setup_logger(log_dir):
    os.makedirs(log_dir, exist_ok=True)
    logger = logging.getLogger("exp")
    logger.setLevel(logging.INFO)
    fh = logging.FileHandler(os.path.join(log_dir, "run.log"))
    sh = logging.StreamHandler(sys.stdout)
    for h in (fh, sh):
        h.setFormatter(logging.Formatter("%(asctime)s %(levelname)s %(message)s"))
        logger.addHandler(h)
    return logger
```

Запись метрик:

```
import csv, os, time
class MetricsWriter:
    def __init__(self, path):
        self.path = path
        write_header = not os.path.exists(path)
        self.f = open(path, "a", newline="")
        self.w = csv.DictWriter(self.f, fieldnames=["step","epoch","primary","is_best","t
        if write_header: self.w.writeheader()
    def log(self, **kw):
        kw.setdefault("timestamp", int(time.time()))
        self.w.writerow(kw); self.f.flush()
```

Обновление best (в post_run.py):

```
def update_best_if_improved(exp_dir, best_dir, primary_name, mode):
    # читаем metrics.csv, находим лучший primary и сравниваем с best_dir/metrics.json
    # копируем best.pt и пишем свежие метаданные только при улучшении
```

9) Дополнительно: защита от перезаписи

- Установите **umask** только на создание, запрет на overwrite (или проверка существования).
- Включите “fail-fast”: если путь для запуска существует — останавливать процесс.
- Добавьте дублирующее сохранение best в облако/другой диск (например, ./mirror/best/...).

10) Короткий чек-лист внедрения

- Создать структуру каталогов и скрипты pre_run/post_run.
- Зафиксировать правила в .cursor/rules.md и MemoryBank.
- Настроить Spec-Kit schema + валидацию конфигов.
- Подключить logger и MetricsWriter в train.py.
- Обновить CI: тесты на сохранение артефактов и best-логику.
- Вставить guardrails в system prompt Cursor: “никаких перезаписей, только новые /runs; best только через post_run; всё логировать”.

Если покажете текущий layout и фрагменты train.py, можно дать точные сниппеты под ваши пути и интегрировать с вашей системой логирования/метрик (включая TensorBoard, CSV и summary.md), а также шаблоны rule-prompts для Cursor/MemoryBank на русском.