# TrackTempo ML Pipeline: Tooling and Automation Summary

## 1. Preprocessing Tooling

Goal: Automate the transformation from raw JSON racecards to model-ready inference data (e.g., `model_ready_infer.pkl`).

Requirements:

- Flatten and normalize JSON files (by date and batch)
- Clean and validate key fields (e.g., time, course, horse names)
- Embed categorical fields (e.g., trainer, jockey, track type)
- Extract and vectorize NLP commentary (optional: use spaCy, sklearn, or transformer embeddings)
- Save intermediate outputs for inspection
- Output: consistent `.pkl` and `.csv` versions of model-ready data

Current State:

- Sequence of Jupyter notebook cells executed manually
- Robust and reproducible, but not modular or batchable
- Candidate for a `preprocess.py` or `make_dataset.py` script

Next Step:

- Refactor notebook into modular functions
- Chain functions in one CLI-accessible preprocessing runner

## 2. Training Tooling

Goal: Train the transformer model from preprocessed inference data + post-race results.

Requirements:

- Load model-ready training dataset from .pkl
- Handle dynamic batching of race fields
- Apply race masking and positional-insensitive attention
- Use BCEWithLogitsLoss for winner prediction
- Log loss curves, Top-1 / Top-3 prediction accuracy
- Support model checkpointing and manual seeding
- Optional: YAML/JSON config file for hyperparams

Current State:

- Training logic exists in notebooks
- Checkpointing is functional
- Evaluation is manual via inline plots

Next Step:

- Create `train.py` with CLI args (epochs, batch size, seed, model path)
- Standardize evaluation metrics output and model checkpoint saving
- Optional: Add experiment logging (CSV/JSON/MLFlow/W&B)

## 3. Inference Tooling

Goal: Run predictions on new racecards using a trained model.

Requirements:

- Load trained model from .pt or TorchScript/ONNX
- Run inference on one or more races in JSON format
- Apply same preprocessing steps as training
- Return top-N predictions and win probabilities
- Optionally format for human-readable reporting or downstream systems

Current State:

- Inference logic is performed ad-hoc in notebooks
- Race-by-race execution
- Output is available for analysis but not deployable

Next Step:

- Wrap inference into `predict.py`
- Standardize input/output format
- Add Streamlit wrapper or REST API for easy access

## 4. Next Steps and Deliverables

- [ ] Refactor preprocessing logic into pipeline script
- [ ] Create `train.py` with logging and CLI control
- [ ] Wrap inference into script with optional Streamlit interface

- [ ] Expand audit logging and tagging (voided races, fuzzy hit reasons)
- [ ] Tag and version datasets monthly (March 2025, April 2025, etc.)

You?re now in a position to move from a notebook-driven workflow to a modular, production-ready racing ML pipeline.