Category	Answers	Typical capabilities	Non-goals
Code versioning (Git)	* Which code snapshot did we use?	* history, diff, review, branching, tags * code provenance for papers and experiments	* does not version large data well * does not record “runs” or produced artifacts by itself
Data versioning and lineage (DVC, lakeFS, etc.)	* Which dataset snapshot did we use? * Which derived artifacts depend on which inputs?	* data snapshots / versions (often stored outside git) * lineage via declared deps/outs * reproducible data pipelines in terms of file dependencies * stage skip via cache and “nothing changed” detection (DVC-style)	* not a full experiment tracker UI * not a scheduler/queue/retry system (beyond simple stage execution)
Environment capture (conda/mamba, lockfiles)	* Which exact dependencies made this run work?	* pinned packages / versions * exportable environment spec (env.yml / lockfile) * reduces environment drift across machines	* does not track runs, metrics, or artifacts
Training frameworks (Lightning, HF Trainer, etc.)	* How do we write training loops faster and more consistently?	* training loop abstractions * standard callbacks, logging hooks * multi-GPU / mixed precision conveniences (depending on framework)	* not experiment tracking by itself * not orchestration by itself * not data versioning
Experiment tracking (MLflow, W&B, etc.)	* Which run produced this result? * What parameters, metrics, and artifacts did it generate?	* run records: params, metrics, tags * artifact logging: plots, tables, models, reports * comparison: search, filter, compare runs * provenance links (to code/config/data references) if you log them	* not HPO search engine (though it can integrate with one) * not orchestration/scheduling (though it can be triggered from one)
HPO optimization (Optuna, Ray Tune, etc.)	* How do we search the hyperparameter and design space systematically?	* define objective functions * search algorithms (samplers), pruning, early stopping * study analysis and visualizations * parallel trial execution (varies by tool and setup)	* does not guarantee provenance unless paired with tracking/versioning * does not replace broader evaluation surface (metrics beyond the objective)
Orchestration (Prefect, Dagster, Airflow, Argo/KFP)	* How do we run the workflow reliably and repeatedly, at scale?	* scheduling and triggers * retries, backoff, timeouts * concurrency / queues / distributed execution primitives * task caching / stage skip (orchestrator-style), idempotency patterns * observability of workflow runs (logs, states)	* not a tracker unless you log to a tracker * not a data versioning system (though it can call one)