AELP / AELP2 — Technical Overview

Generated 2025-09-29 05:03 UTC

1) What problem we solve

We predict and assemble winning ad creatives and portfolios before spending, using offline data from Meta and vendor sources. The system ingests historical performance, scores creative blueprints with a trained new■ad ranker, calibrates CPM/CTR/CVR baselines by placement, simulates CAC/volume via Monte■Carlo, then packages a day■by■day launch plan with per■ad setup.

2) High■level architecture

```
Data & Vendors → Ingestion → Feature/Model → Forecasts/RL → Planner UI → Launch
[Meta Ads API] [Vendor CSV/SearchAPI] \ / \ / v v +-----------------------+ |
AELP2/pipelines & | | AELP2/tools importers | +-----------------------+ | v
+-----------------------+ | Creative objects & | | features (reports/) |
+-----------------------+ | v +-----------------------+ | New■ad ranker + | |
baselines/forecasts | +-----------------------+ | | | v | Offline bandit sim v
+-----------------------+ | Creative Planner UI | | (Next API + Vite UI) |
+-----------------------+ | v +-----------------------+ | Setup Playbooks / | |
Launch & Monitoring | +-----------------------+
```

Repository inventory

Total files scanned: 61832 (Python: 343)

Top extensions:

- .js: 26367
- .map: 10174
- .ts: 9060
- .json: 4903
- .md: 1548
- .mjs: 1505
- noext: 1428
- .mts: 1298
- .png: 1249
- .proto: 943
- .svg: 554
- .cjs: 455
- .jpg: 427
- .py: 343
- .cts: 335

Key environment (masked)

```
export GOOGLE_CLOUD_PROJECT=aura-t…
export BIGQUERY_USERS_DATASET=***
export BIGQUERY_TRAINING_DATASET=gaelp_…
export BIGQUERY_DATASET=gaelp_…
export GA4_PROPERTY_ID=proper…
export NODE_ENV=***
export GCS_BUCKET=gaelp-…
export CREATIVE_GCS_BUCKET=gaelp-…
export REDIS_HOST=***
export REDIS_PORT=***
export REDIS_URL=redis:…
export OPENAI_API_KEY=sk-svc…
export ANTHROPIC_API_KEY=sk-ant…
export META_ACCESS_TOKEN=EAAZBi…
export META_ADLIBRARY_ACCESS_TOKEN=EAAZBi…
export META_API_VERSION=***
export META_ADLIBRARY_COUNTRIES=***
export META_APP_ID=440073…
export META_APP_SECRET=***
export META_BUSINESS_ID=477239…
export META_ACCOUNT_ID=act_19…
export META_PRIMARY_ACTION=***
export GOOGLE_ADS_DEVELOPER_TOKEN=uikJ5k…
export GOOGLE_ADS_CLIENT_ID=637142…
export GOOGLE_ADS_CLIENT_SECRET=GOCSPX…
export GOOGLE_ADS_REFRESH_TOKEN=1//04T…
export GOOGLE_ADS_LOGIN_CUSTOMER_ID=***
export GOOGLE_ADS_CUSTOMER_ID=***
export TIKTOK_ACCESS_TOKEN=your_t…
export TIKTOK_APP_ID=your_t…
```

3) Pipelines & Tools

```
1) Meta → BigQuery (by placement): meta_to_bq.py --by_placement →
meta_ad_performance_by_place 2) Vendor/Ad Library imports: fetch_searchapi_meta.py
→ vendor_imports/*.csv → import_vendor_meta_creatives.py 3) Features + Scoring:
build_features_from_creative_objects.py → creative_features.jsonl
score_vendor_creatives.py → vendor_scores.json (p_win, lcb) 4) Baselines &
```

Forecasts: compute_us_paid_baselines_by_place.py → us_meta_baselines_by_place.json forecast_us_cac_volume.py → us_cac_volume_forecasts.json (+ Balance) 5) RL Prep and Simulation: add_novelty_and_export_rl_pack.py → rl_test_pack.json, asset_briefs.json simulate_bandit_from_forecasts.py → rl_offline_simulation.json 6) UI/API (Planner): Next app routes → /api/planner/* (forecasts, RL, vendor scores) Vite UI → /creative-planner (Top■K, packages, setup)

Key scripts

AELP2/pipelines/ads_common.py

AELP2/pipelines/ads_mcc_coordinator.py

AELP2/pipelines/attribution_engine_stub.py

AELP2/pipelines/audience_expansion.py

AELP2/pipelines/auto_recalibration.py

AELP2/pipelines/bandits_writer.py

AELP2/pipelines/bid_edit_proposals.py

AELP2/pipelines/bid_landscape_modeling.py

AELP2/pipelines/bing_to_bq.py

AELP2/pipelines/build_affiliate_triggered.py

AELP2/pipelines/build_ga_aligned_daily.py

AELP2/pipelines/build_impact_partner_domains.py

AELP2/pipelines/build_triggered_series.py

AELP2/pipelines/calibration_parallel_stub.py

AELP2/pipelines/calibration_stratified_views.py

AELP2/pipelines/canary_monitoring.py

AELP2/pipelines/canary_timeline_writer.py

AELP2/pipelines/channel_attribution_r.py

AELP2/pipelines/channel_attribution_runner.py

AELP2/pipelines/check_data_quality.py

AELP2/pipelines/competitive_intel_ingest.py

AELP2/pipelines/copy_optimizer_stub.py

AELP2/pipelines/cost_monitoring_stub.py

AELP2/pipelines/create_bq_views.py

AELP2/pipelines/create_channel_views.py

AELP2/pipelines/creative_ab_planner.py

AELP2/pipelines/creative_bandit_head.py

AELP2/pipelines/creative_embeddings_stub.py

AELP2/pipelines/creative_fatigue_alerts.py

AELP2/pipelines/cross_platform_kpi_daily.py

AELP2/pipelines/dayparting_optimizer.py

AELP2/pipelines/delayed_conversions_stub.py

AELP2/pipelines/fidelity_evaluation.py

AELP2/pipelines/fidelity_parallel_stub.py

AELP2/pipelines/ga4_backfill_audience_breakdown.py

AELP2/pipelines/ga4_build_attribution.py

AELP2/pipelines/ga4_build_audience_breakdown.py

AELP2/pipelines/ga4_build_audiences.py

AELP2/pipelines/ga4_build_derived.py

AELP2/pipelines/ga4_extract_affiliate_clickids.py

AELP2/pipelines/ga4_lagged_attribution.py

AELP2/pipelines/ga4_paths_attribution.py

AELP2/pipelines/ga4_permissions_check.py

AELP2/pipelines/ga4_to_bq.py

AELP2/pipelines/generate_qs_alerts.py

AELP2/pipelines/generate_qs_fix_tickets.py

AELP2/pipelines/google_ads_ad_performance_to_bq.py

AELP2/pipelines/google_ads_adgroups_to_bq.py

AELP2/pipelines/google_ads_assets_to_bq.py

AELP2/pipelines/google_ads_conversion_actions_to_bq.py

AELP2/pipelines/google_ads_conversion_stats_by_action_to_bq.py

AELP2/pipelines/google_ads_discover_accounts.py

AELP2/pipelines/google_ads_geo_device_to_bq.py

AELP2/pipelines/google_ads_keywords_to_bq.py

AELP2/pipelines/google_ads_mcc_to_bq.py

AELP2/pipelines/google_ads_search_terms_to_bq.py

AELP2/pipelines/google_ads_to_bq.py

AELP2/pipelines/google_recommendations_scanner.py

AELP2/pipelines/gsc_to_bq.py

AELP2/pipelines/hints_to_proposals.py

AELP2/pipelines/impact_backfill_performance.py

AELP2/pipelines/impact_clicks_to_bq.py

AELP2/pipelines/impact_entities_to_bq.py

AELP2/pipelines/impact_run_report_to_bq.py

AELP2/pipelines/impact_to_bq.py

AELP2/pipelines/journey_path_summary.py

AELP2/pipelines/journeys_populate.py

AELP2/pipelines/kpi_consistency_check.py

AELP2/pipelines/kpi_crosscheck.py

AELP2/pipelines/linkedin_to_bq.py

AELP2/pipelines/load_affiliate_ach_costs.py

AELP2/pipelines/load_promo_calendar.py

AELP2/pipelines/lp_ab_hooks_stub.py

AELP2/pipelines/ltv_priors.py

AELP2/pipelines/meta_to_bq.py

AELP2/pipelines/mmm_attributed_service.py

AELP2/pipelines/mmm_lightweightmmm.py

AELP2/pipelines/mmm_service.py

AELP2/pipelines/model_registry_stub.py

AELP2/pipelines/module_runner.py

AELP2/pipelines/namespace_refactor_report.py

AELP2/pipelines/offpolicy_eval.py

AELP2/pipelines/opportunity_outcomes.py

AELP2/pipelines/opportunity_scanner.py

AELP2/pipelines/ops_alerts_stub.py

AELP2/pipelines/parity_report.py

AELP2/pipelines/permissions_check.py

AELP2/pipelines/platform_skeleton_log.py

AELP2/pipelines/policy_hints_writer.py

AELP2/pipelines/portfolio_optimizer.py

AELP2/pipelines/privacy_audit_stub.py

AELP2/pipelines/propensity_uplift.py

AELP2/pipelines/quality_signal_daily.py

AELP2/pipelines/quality_softgate_stub.py

AELP2/pipelines/realtime_budget_pacer.py

AELP2/pipelines/reconcile_posthoc.py

AELP2/pipelines/rl_policy_hints_writer.py

AELP2/pipelines/robyn_runner.py

AELP2/pipelines/robyn_validator.py

AELP2/pipelines/rule_engine.py

AELP2/pipelines/security_audit.py

AELP2/pipelines/segments_to_audiences.py

AELP2/pipelines/slo_watch_stub.py

AELP2/pipelines/tiktok_to_bq.py

AELP2/pipelines/training_posthoc_reconciliation.py

AELP2/pipelines/trust_gates_evaluator.py

AELP2/pipelines/uplift_eval.py

AELP2/pipelines/upload_google_offline_conversions.py

AELP2/pipelines/upload_meta_capi_conversions.py

AELP2/pipelines/users_db_stub.py

AELP2/pipelines/value_bridge.py

AELP2/pipelines/youtube_reach_planner.py

Tools

AELP2/tools/ad_level_accuracy.py

AELP2/tools/ad_level_calibration_v22.py

AELP2/tools/ad_level_calibration_v23.py

AELP2/tools/ad_level_ranker_v24.py

AELP2/tools/add_novelty_and_export_rl_pack.py

AELP2/tools/add_phone_look.py

AELP2/tools/annotate_weekly_with_policy.py

AELP2/tools/assemble_boring_good.py

AELP2/tools/assemble_enforced.py

AELP2/tools/assemble_from_assets.py

AELP2/tools/assemble_original_batch.py

AELP2/tools/assemble_pattern.py

AELP2/tools/assemble_real_ad.py

AELP2/tools/assemble_spot_the_tell.py

AELP2/tools/assemble_two_screens.py

AELP2/tools/assemble_with_vo.py

AELP2/tools/attention_heuristics.py

AELP2/tools/backfill_ad_daily_insights.py

AELP2/tools/bigspy_auto_export.py

AELP2/tools/build_brand_pack.py

AELP2/tools/build_cool_variant.py

AELP2/tools/build_features_from_creative_objects.py

AELP2/tools/build_labels_weekly.py

AELP2/tools/build_placement_calibrators.py

AELP2/tools/build_proof_assets.py

AELP2/tools/build_system_overview_pdf.py

AELP2/tools/build_topk_from_scores.py

AELP2/tools/build_weekly_predictions.py

AELP2/tools/cascade_dr_topk.py

AELP2/tools/check_dual_gate_thresholds.py

AELP2/tools/collect_kb_overlays.py

AELP2/tools/compute_aesthetic_metrics.py

AELP2/tools/compute_locked_targets.py

AELP2/tools/compute_mmm_bands.py

AELP2/tools/compute_motion_features.py

AELP2/tools/compute_target_cac_frontier.py

AELP2/tools/compute_us_meta_baselines.py

AELP2/tools/compute_us_paid_baselines_by_place.py

AELP2/tools/conformal_topk.py

AELP2/tools/conformal_topk_weekly.py

AELP2/tools/crawl_youtube_titles.py

AELP2/tools/demo_storyboard.py

AELP2/tools/detect_objects_yolo.py

AELP2/tools/download_runway_results.py

AELP2/tools/dual_gate_weekly.py

AELP2/tools/eleven_design_voice.py

AELP2/tools/enrich_creatives_with_objects.py

AELP2/tools/enrich_weekly_with_cpc.py

AELP2/tools/estimate_policy_uplift.py

AELP2/tools/eval_ablation.py

AELP2/tools/eval_wbua.py

AELP2/tools/eval_wbua_cluster.py

AELP2/tools/eval_wbua_forward.py

AELP2/tools/eval_wbua_novel.py

AELP2/tools/export_meta_ad_configs.py

AELP2/tools/export_meta_creative_outcomes.py

AELP2/tools/export_ui_overlays_stub.py

AELP2/tools/extract_audio_features.py

AELP2/tools/extract_audio_lufs.py

AELP2/tools/extract_keyframes.py

AELP2/tools/extract_legibility_features.py

AELP2/tools/extract_visual_embeddings.py

AELP2/tools/fetch_campaign_placement_conversions.py

AELP2/tools/fetch_meta_adlibrary.py

AELP2/tools/fetch_meta_creatives.py

AELP2/tools/fetch_searchapi_meta.py

AELP2/tools/fidelity_eval_roll.py

AELP2/tools/finalize_gate.py

AELP2/tools/fit_cac_calibrator.py

AELP2/tools/forecast_us_cac_volume.py

AELP2/tools/forward_forecast.py

AELP2/tools/gen_eleven_vo.py

AELP2/tools/gen_eleven_vo_pack.py

AELP2/tools/gen_runway_hooks.py

AELP2/tools/gen_veo_hooks.py

AELP2/tools/generate_advantage_manifest.py

AELP2/tools/generate_balance_blueprints.py

AELP2/tools/generate_blueprints.py

AELP2/tools/generate_score_loop.py

AELP2/tools/hourly_multipliers.py

AELP2/tools/import_proof_clips.py

AELP2/tools/import_vendor_meta_creatives.py

AELP2/tools/join_dna_to_creatives.py

AELP2/tools/join_finals_features.py

AELP2/tools/journey_states_from_bq.py

AELP2/tools/log_creative_dna.py

AELP2/tools/make_meta_account_audit.py

AELP2/tools/merge_copy_banks.py

AELP2/tools/meta_campaign_bayes.py

AELP2/tools/mine_meta_copy.py

AELP2/tools/mj_ingest.py

AELP2/tools/normalize_audio.py

AELP2/tools/offline_creative_search.py

AELP2/tools/offline_rl_stub.py

AELP2/tools/ope_config_weighting.py

AELP2/tools/ope_topk.py

AELP2/tools/ope_upgrades.py

AELP2/tools/ope_uplift_topk.py

AELP2/tools/parse_brand_guide.py

AELP2/tools/policy_audit_live.py

AELP2/tools/portfolio_selector.py

AELP2/tools/propose_kb_non_claims.py

AELP2/tools/pull_google_ads_copy.py

AELP2/tools/pull_google_ads_copy_from_bq.py

AELP2/tools/pull_google_ads_copy_rest.py

AELP2/tools/pull_impact_copy.py

AELP2/tools/pull_impact_copy_from_bq.py

AELP2/tools/qc_gates.py

AELP2/tools/render_bayes_report.py

AELP2/tools/render_quiz_overlay.py

AELP2/tools/render_sim_onepager.py

AELP2/tools/report_policy_vs_mixed.py

AELP2/tools/rl_shadow_score.py

AELP2/tools/score_ad_items.py

AELP2/tools/score_new_ads.py

AELP2/tools/score_vendor_creatives.py

AELP2/tools/selector_utility_baseline.py

AELP2/tools/self_judge.py

AELP2/tools/serve_previews.py

AELP2/tools/sim_fidelity_campaigns.py

AELP2/tools/sim_fidelity_campaigns_journey.py

AELP2/tools/sim_fidelity_campaigns_temporal.py

AELP2/tools/sim_fidelity_campaigns_temporal_v2.py

AELP2/tools/sim_fidelity_campaigns_temporal_v3.py

AELP2/tools/sim_fidelity_eval.py

AELP2/tools/sim_fidelity_eval_empirical.py

AELP2/tools/sim_fidelity_journey_criteo.py

AELP2/tools/simulate_bandit_from_forecasts.py

AELP2/tools/sort_uploaded_assets.py

AELP2/tools/still_to_motion.py

AELP2/tools/tag_ad_patterns.py

AELP2/tools/train_new_ad_ranker.py

AELP2/tools/tune_generator_priors.py

AELP2/tools/uplift_baseline_test.py

AELP2/tools/validate_candidates.py

AELP2/tools/validate_kb.py

AELP2/tools/voice_library_add_prompt.py

AELP2/tools/weekly_fidelity_from_v3.py

AELP2/tools/weekly_relabel.py

AELP2/tools/weekly_topN_dual.py

Apps & APIs

AELP2/apps/dashboard (Next.js API + UI)

AELP2/external/growth-compass-77 (Vite React UI)

API routes (Next.js):

/api/ab/exposure

/api/ads/creative

/api/ads/google/create

/api/auth/[...nextauth]

/api/bq/ab-experiments

/api/bq/ab-exposures

/api/bq/ads

/api/bq/ads/qs-is

/api/bq/alerts

/api/bq/approvals/queue

/api/bq/approvals/reject

/api/bq/arms/posteriors

/api/bq/auctions/minutely

/api/bq/bid-edits

/api/bq/bid-landscape

/api/bq/bidding

/api/bq/bidding_recent

/api/bq/bidding_replay

/api/bq/canaries

/api/bq/channel-attribution

/api/bq/copy-suggestions

/api/bq/creative-variants

/api/bq/creatives

/api/bq/cross-kpi

/api/bq/dayparting

/api/bq/episodes

/api/bq/explore/cells

/api/bq/fidelity

/api/bq/freshness

/api/bq/ga4/channels

/api/bq/halo

/api/bq/headroom

/api/bq/impact/partner-opportunities

/api/bq/impact/partner-seed-harvest

/api/bq/impact/seed-harvest

/api/bq/impact/top-partners

/api/bq/impact/triggered-by-partner

/api/bq/interference

/api/bq/journeys/sankey

/api/bq/kpi

/api/bq/kpi/daily

/api/bq/kpi/summary

/api/bq/kpi/yesterday

/api/bq/lp-ab

/api/bq/lp/tests

/api/bq/ltv/summary

/api/bq/mmm/allocations

/api/bq/mmm/channels

/api/bq/mmm/curves

/api/bq/mmm/whatif

/api/bq/offpolicy

/api/bq/opportunities

/api/bq/ops-alerts

/api/bq/policy-enforcement

/api/bq/portfolio-allocations

/api/bq/qa/cac/pacer

/api/bq/qa/enrollments/aligned

/api/bq/qa/enrollments/daily

/api/bq/qa/enrollments/monthly

/api/bq/reach-estimates

/api/bq/safety

/api/bq/subagents

/api/bq/value-uploads

/api/canvas/list

/api/canvas/pin

/api/canvas/unpin

/api/chat

/api/chat/stream

/api/connections/health

/api/control/ab-approve

/api/control/ads-ingest

/api/control/apply-canary

/api/control/apply-creative

/api/control/audience/sync

/api/control/backfill/bid

/api/control/backfill/ltv

/api/control/backfill/policy

/api/control/backfill/rl

/api/control/bandit-apply

/api/control/bandit-approve

/api/control/canary-rollback

/api/control/creative/enqueue

/api/control/creative/publish

/api/control/emergency-stop

/api/control/evaluate

/api/control/fidelity-kpi

/api/control/ga4-attribution

/api/control/ga4-ingest

/api/control/kpi-lock

/api/control/lp/publish

/api/control/onboarding/backfill

/api/control/onboarding/create

/api/control/opportunity-approve

/api/control/reach-planner

/api/control/status

/api/control/switch-dataset

/api/control/test-account-ingest

/api/control/training-run

/api/control/value-upload/google

/api/control/value-upload/meta

/api/creative/generate

/api/creative/publish

/api/dataset

/api/kpi-source

/api/media/generate

/api/module/[slug]/result

/api/module/[slug]/start

/api/module/[slug]/status

/api/ops/flows

/api/planner/assets/briefs

/api/planner/forecasts

/api/planner/packages

/api/planner/rl

/api/planner/setup/[id]

/api/planner/vendor-scores

/api/queue/processor

/api/reports/executive

/api/research/channels

/api/scenarios/model

/api/training/start

External UI pages (Vite):

/Affiliates, /Approvals, /AuctionsMonitor, /Audiences, /Backstage, /Canvas, /Channels, /CreativeCenter, /CreativePlanner, /ExecutiveDashboard, /Experiments, /Finance, /Index, /LandingPages, /NotFound, /OpsChat, /QS, /RLInsights, /SpendPlanner, /TrainingCenter

Data lineage (producer → artifact → consumer)

vendor_imports/*.csv → import_vendor_meta_creatives.py → reports/creative_objects/

creative_objects/ → build_features_from_creative_objects.py → reports/creative_features/creative_features.jsonl

creative_features.jsonl + models/new_ad_ranker → score_vendor_creatives.py → reports/vendor_scores.json

BigQuery meta_ad_performance_by_place → compute_us_paid_baselines_by_place.py → reports/us_meta_baselines_by_place.json

reports/us_meta_baselines.json + ad_blueprints_top20.json → forecast_us_cac_volume.py → reports/us_cac_volume_forecasts.json

reports/us_balance_forecasts.json (Balance forecasts generator) → reports/rl_balance_pack.json

reports/us_cac_volume_forecasts.json → simulate_bandit_from_forecasts.py → reports/rl_offline_simulation.json

reports/* (forecasts, RL, vendor scores) → Next API /api/planner/* → Vite UI /creative-planner

4) Forecasting & Accuracy

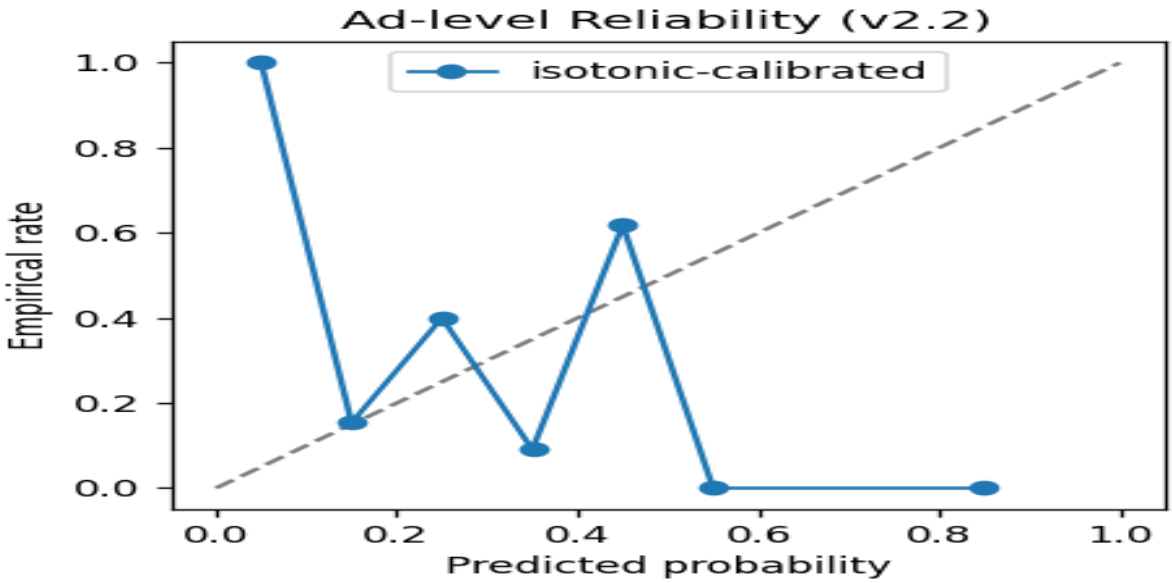
Vendor creatives scored: 1650

Forecast items — Security: 20, Balance: 20

Baseline keys (by placement): 23

Ranker precision@5: 0.2666666666666667, precision@10: 0.3

Calibration (reliability curve):



5) Creative Planner & Launch

The Planner exposes forecasts, RL packs, and setup checklists. It provides package builders for $30k/$50k daily budgets, per■ad instructions (Campaign/Ad Set/Ad), and exports (PDF/ZIP/JSON).

Endpoints

/api/planner/forecasts — security/balance forecasts

/api/planner/vendor-scores — scored vendor creatives

/api/planner/rl — rl packs + offline sim

/api/planner/setup/[creative_id] — per■ad setup checklist

6) Ops & How to Run

Servers: Next API (3000), Vite UI (8080).

Start API: cd AELP2/apps/dashboard && npm run build && PORT=3000 NODE_ENV=production npm run start

Start UI: cd AELP2/external/growth-compass-77 && npm run build && npm run preview -- --host 127.0.0.1 --port 8080

Preview finals: python3 AELP2/tools/serve_previews.py and tunnel 8080

Data refresh (typical sequence)

1) Meta ingest (by placement) → 2) Baselines → 3) Vendor import → 4) Features → 5) Score → 6) Forecasts → 7) RL sim → 8) Planner

7) Reports inventory snapshot

ad_balance_blueprints_top20.json: items=20

ad_blueprints_top20.json: items=20

ad_level_accuracy.json: keys=7

ad_level_accuracy_v22.json: keys=6

ad_level_accuracy_v22_details.json:

ad_level_accuracy_v23.json: keys=5

ad_level_accuracy_v23_details.json:

asset_briefs.json: items=20

attention_scores.json: items=6

audio_features.json: items=6

cac_calibrator.json: keys=4

candidates_validation.json: keys=2

cascade_dr_topk.json: keys=1

conformal_topk_v24.json: keys=2

conformal_topk_weekly.json: keys=5

copy_bank.json: keys=2

copy_bank_merged.json: items=171

creative_leaderboard.json: items=50

creative_render_log.json: items=10

dual_gate_status.json: keys=5

dual_gate_weekly.json: keys=1

eval_protocol.json: keys=8

eval_protocol_weekly.json: keys=6

google_ads_copy.json: keys=1

hourly_multipliers.json: keys=3

impact_copy.json: keys=1

journey_states_probe.json: keys=5

kb_non_claim_suggestions.json: keys=1

kb_validation.json: keys=2

new_ad_scores.json: items=14

ope_config_weighted.json: keys=1

ope_upgrades_topk.json: keys=1

policy_audit.json: keys=2

policy_uplift.json: keys=2

policy_vs_mixed_summary.json: keys=2

rl_balance_pack.json: items=20

rl_offline_simulation.json: keys=4

rl_shadow_score.json: keys=2

rl_test_pack.json: items=20

runway_tasks.json: keys=1

selector_choice.json: keys=2

sim_fidelity.json: keys=3

sim_fidelity_campaigns.json: keys=4

sim_fidelity_campaigns_journey.json: keys=2

sim_fidelity_campaigns_temporal.json: keys=2

sim_fidelity_campaigns_temporal_v2.json: keys=3

sim_fidelity_campaigns_temporal_v3.json: keys=2

sim_fidelity_roll.json: keys=1

sim_forward_forecast.json: keys=4

sort_uploads_log.json: keys=4

target_cac_locked.json: keys=2

topk_slate.json: items=3

us_balance_forecasts.json: items=20

us_cac_volume_forecasts.json: items=20

us_meta_baselines.json: keys=19

us_meta_baselines_by_place.json: items=23

vendor_scores.json: items=1650

vendor_top20.json: items=20

veo_tasks.json: keys=1

wbua_cluster_summary.json: keys=5

wbua_forward.json: keys=4

wbua_novel_summary.json: keys=4

wbua_summary.json: keys=4

weekly_portfolio.json: keys=1

weekly_topN_dual.json: keys=1

youtube_copy.json: items=14

Report type hints

ad_balance_blueprints_top20.json — count:number, items:array[object]

ad_blueprints_top20.json — count:number, items:array[object]

ad_level_accuracy.json — precision_at_5:number, precision_at_10:number, pairwise_win_rate:number, kendall_tau:number, n_campaigns:number, n_creatives:number, notes:array[unknown]

ad_level_accuracy_v22.json — precision_at_5:number, precision_at_10:number, pairwise_win_rate:number, stability_top10:number, n_campaigns:number, n_samples:number

ad_level_accuracy_v22_details.json — list[object] keys=campaign_file,precision_at_5,precision_at_10,pairwise_win_rate

ad_level_accuracy_v23.json — precision_at_5:number, precision_at_10:number, pairwise_win_rate:null, n_campaigns:number, n_samples:number

ad_level_accuracy_v23_details.json — list[object] keys=campaign_file,precision_at_5,precision_at_10,pairwise_win_rate

asset_briefs.json — items:array[object]

attention_scores.json — items:array[object]

audio_features.json — items:array[object]

cac_calibrator.json — x:array[number], y:array[number], note:string, n:number

candidates_validation.json — checked:number, issues:array[unknown]

cascade_dr_topk.json — results:array[object]

conformal_topk_v24.json — selector:string, curves:object

conformal_topk_weekly.json — K:number, alpha:number, coverage:number, n_eval:number, per_campaign:array[object]

copy_bank.json — titles:array[object], bodies:array[object]

copy_bank_merged.json — items:array[object], summary:object

creative_leaderboard.json — count:number, items:array[object]

creative_render_log.json — items:array[object]

dual_gate_status.json — ok:array[object], fail:array[object], min_precision:number, min_yield:number, ts:number

dual_gate_weekly.json — grid:object

eval_protocol.json — window_days:number, train_days:number, test_days:number, split:string, cross_validation:string, labels:object, objectives:object, gates:object

eval_protocol_weekly.json — window_days:number, aggregation:string, split:string, cross_validation:string, labels:object, gates:object

google_ads_copy.json — error:string

hourly_multipliers.json — since:string, until:string, hourly:object

impact_copy.json — error:string

journey_states_probe.json — project:null, dataset:null, tables:object, note:string, error:string

kb_non_claim_suggestions.json — suggestions:array[object]

kb_validation.json — validated:number, errors:array[unknown]

new_ad_scores.json — items:array[object]

ope_config_weighted.json — results:array[object]

ope_upgrades_topk.json — results:array[object]

policy_audit.json — counts:object, non_compliant:array[object]

policy_uplift.json — cvr_uplift_ratio:object, cac_gain_ratio:object

policy_vs_mixed_summary.json — mixed:object, policy_only:object

rl_balance_pack.json — items:array[object]

rl_offline_simulation.json — days:number, daily_budget:number, ranking:array[object], history:array[object]

rl_shadow_score.json — generated_at:string, results:array[object]

rl_test_pack.json — items:array[object]

runway_tasks.json — results:array[object]

selector_choice.json — selector:string, candidates:array[object]

sim_fidelity.json — days:array[object], mape_purchases_median:number, mape_cac_median:null

sim_fidelity_campaigns.json — window:object, summary:object, daily:array[object], per_campaign:object

sim_fidelity_campaigns_journey.json — summary:object, daily:array[object]

sim_fidelity_campaigns_temporal.json — summary:object, daily:array[object]

sim_fidelity_campaigns_temporal_v2.json — summary:object, daily:array[object], note:string

sim_fidelity_campaigns_temporal_v3.json — summary:object, daily:array[object]

sim_fidelity_roll.json — splits:array[object]

sim_forward_forecast.json — train_days:array[string], hold_days:array[string], summary:object, daily:array[object]

sort_uploads_log.json — src:string, moved_count:number, moved:array[unknown], skipped:array[string]

target_cac_locked.json — targets:object, weeks:number

topk_slate.json — items:array[object], note:string

us_balance_forecasts.json — items:array[object], budgets:array[number]

us_cac_volume_forecasts.json — budgets:array[number], items:array[object]

us_meta_baselines.json — start_date:string, end_date:string, days:number, cpm_avg:number, cpm_p10:number, cpm_p50:number, cpm_p90:number, ctr_avg:number, ctr_p10:number, ctr_p50:number, ctr_p90:number, cvr_avg:number, cvr_p10:number, cvr_p50:number, cvr_p90:number, imps:number, clicks:number, cost:number, conversions:number

us_meta_baselines_by_place.json — start_date:string, end_date:string, items:object

vendor_scores.json — items:array[object], count:number

vendor_top20.json — count:number, items:array[object]

veo_tasks.json — results:array[object]

wbua_cluster_summary.json — wbua_cluster_holdout:number, pairs:number, ci95_groups:array[number], groups:number, weeks_counted:number

wbua_forward.json — wbua_forward:number, pairs:number, ci95_groups:array[number], groups:number

wbua_novel_summary.json — wbua_novel:number, pairs_novel:number, ci95_groups:array[number], groups:number

wbua_summary.json — wbua:number, pairs:number, ci95:array[number], groups:number

weekly_portfolio.json — portfolio:array[object]

weekly_topN_dual.json — topN:array[object]

youtube_copy.json — tried:array[string], count:number, items:array[object]

BigQuery tables (from meta_to_bq.py)

meta_ad_performance

```
date: DATE campaign_id: STRING adset_id: STRING ad_id: STRING impressions: INT64 clicks: INT64 cost: FLOAT64 conversions: FLOAT64 revenue: FLOAT64 ctr: FLOAT64 cvr: FLOAT64 avg_cpc: FLOAT64 name_hash: STRING date: DATE campaign_id: STRING adset_id: STRING ad_id: STRING publisher_platform: STRING placement: STRING device: STRING impressions: INT64 clicks: INT64 cost: FLOAT64 conversions: FLOAT64 revenue: FLOAT64 ctr: FLOAT64 cvr: FLOAT64 avg_cpc: FLOAT64
```

meta_ad_performance_by_place

```
date: DATE campaign_id: STRING adset_id: STRING ad_id: STRING publisher_platform: STRING placement: STRING device: STRING impressions: INT64 clicks: INT64 cost: FLOAT64 conversions: FLOAT64 revenue: FLOAT64 ctr: FLOAT64 cvr: FLOAT64 avg_cpc: FLOAT64
```

8) Risks & Mitigations

• API rate limits → backoff + window slicing in meta_to_bq.py

• US Ad Library gaps → supplement with SearchAPI/vendor CSVs; rely on our own Meta performance for baselines

• Forecast drift → recalibrate baselines weekly; placement-aware CVR clamps; conformal lower bounds

9) Script docstrings (headnotes)

tools/ad_level_accuracy.py: Compute ad-level offline accuracy metrics from historical data and simulator predictions.

tools/ad_level_calibration_v22.py: no docstring

tools/ad_level_calibration_v23.py: no docstring

tools/ad_level_ranker_v24.py: no docstring

tools/add_novelty_and_export_rl_pack.py: no docstring

tools/add_phone_look.py: Apply a subtle smartphone look: grain, slight vignette, gentle motion jitter.

tools/annotate_weekly_with_policy.py: Join adset/campaign config into weekly creatives and mark policy compliance.

tools/assemble_boring_good.py: Assemble 3 "boringly good" Aura variants:

tools/assemble_enforced.py: Assemble a 9:16 ad with enforced slot grammar using a manifest + recipe.

tools/assemble_from_assets.py: no docstring

tools/assemble_original_batch.py: Assemble fully original ads from Runway clips and VO packs.

tools/assemble_pattern.py: Assemble a pattern-driven ad using:

tools/assemble_real_ad.py: Assemble a "real" ad using Runway hooks, a proof clip, a relief clip, captions, and ElevenLabs VO.

tools/assemble_spot_the_tell.py: Assemble a 9:16 "Spot The Tell" video:

tools/assemble_two_screens.py: Assemble a 9:16 split-screen "Two Screens, Two Outcomes" video.

tools/assemble_with_vo.py: Assemble a Veo/Runway hook with ElevenLabs VO, add end-card and disclaimer overlay, normalize audio.

tools/attention_heuristics.py: First-3s attention heuristics (rough):

tools/backfill_ad_daily_insights.py: Backfill ad-level daily insights in small date chunks with checkpointing.

tools/bigspy_auto_export.py: Headless BigSpy exporter (best-effort, cookie-based login reuse)

tools/build_brand_pack.py: Build a minimal brand pack from existing ads + site:

tools/build_cool_variant.py: Build a polished preview ad from your brand + MJ assets without waiting on remote video models.

tools/build_features_from_creative_objects.py: Build lightweight numeric features for creatives using the cached Meta

tools/build_labels_weekly.py: Construct pairwise training data (diff features, label win vs baseline) from

tools/build_placement_calibrators.py: no docstring

tools/build_proof_assets.py: Build simple proof overlay PNGs and an end-card image from brand pack.

tools/build_system_overview_pdf.py: Build a comprehensive PDF (for non-coders) that explains AELP/AELP2:

tools/build_topk_from_scores.py: no docstring

tools/build_weekly_predictions.py: Build weekly (calendar) predictions and actuals per creative from ad-level daily insights.

tools/cascade_dr_topk.py: Cascade-DR style estimator for a Top-K slate policy.

tools/check_dual_gate_thresholds.py: Check dual-gate precision and yield against thresholds and write a status file.

tools/collect_kb_overlays.py: Collect claim overlays (text + mandatory disclaimer) and a default CTA from KB for use in end-cards.

tools/compute_aesthetic_metrics.py: Compute lightweight aesthetic/quality proxies (no-ref):

tools/compute_locked_targets.py: Compute locked target_CAC per campaign from recent weekly files to stabilize gates

tools/compute_mmm_bands.py: Compute simple MMM-like daily spend bands (min/base/max) from recent spend and returns using a concave response proxy.

tools/compute_motion_features.py: Compute simple motion features from finals using OpenCV optical flow.

tools/compute_target_cac_frontier.py: Compute per-campaign target_CAC by scanning historical CAC percentiles and choosing the point that maximizes

tools/compute_us_meta_baselines.py: Compute US Meta baselines from BigQuery and write to reports/us_meta_baselines.json.

tools/compute_us_paid_baselines_by_place.py: Compute US Meta paid-event baselines by publisher_platform and placement from

tools/conformal_topk.py: no docstring

tools/conformal_topk_weekly.py: Selection-conditional conformal bounds for weekly creatives.

tools/crawl_youtube_titles.py: Lightweight crawler to fetch video titles/descriptions from public YouTube channel pages without API keys.

tools/demo_storyboard.py: Generate a 4-frame 9:16 storyboard from the Identity KB:

tools/detect_objects_yolo.py: Detect objects on mid-frames using YOLOv8n (CPU). Reports counts of person and cell phone.

tools/download_runway_results.py: no docstring

tools/dual_gate_weekly.py: Dual-gate weekly evaluator.

tools/eleven_design_voice.py: Design and create an ElevenLabs voice from a natural-language prompt, then store it in our library.

tools/enrich_creatives_with_objects.py: no docstring

tools/enrich_weekly_with_cpc.py: Enrich weekly creatives with a CPC-mix based predicted CAC.

tools/estimate_policy_uplift.py: Estimate coarse uplift scalars for key policy switches by comparing compliant-like vs noncompliant items within campaign-weeks.

tools/eval_ablation.py: no docstring

tools/eval_wbua.py: Evaluate Weekly Baseline Uplift Accuracy (WBUA) on weekly_creatives/* files.

tools/eval_wbua_cluster.py: Cluster-holdout WBUA: treat creative "families" as clusters using ad name

tools/eval_wbua_forward.py: Forward-holdout WBUA: freeze decisions using only history up to week t-1 and

tools/eval_wbua_novel.py: Compute WBUA (Weekly Baseline Uplift Accuracy) for novel-only creatives.

tools/export_meta_ad_configs.py: Export current campaign and adset configuration from Meta Graph API.

tools/export_meta_creative_outcomes.py: Export historical per-ad outcomes from Meta (read-only) and build files for ad-level accuracy.

tools/export_ui_overlays_stub.py: Stub exporter for UI overlays: create timing JSON entries for proof inserts.

tools/extract_audio_features.py: Extract simple audio features via ffprobe:

tools/extract_audio_lufs.py: Extract LUFS using ffmpeg loudnorm analyze mode for finals.

tools/extract_keyframes.py: Extract simple keyframes for local MP4s (finals) to AELP2/reports/keyframes//tXXXX.jpg.

tools/extract_legibility_features.py: Compute legibility proxy per final by reusing self_judge (contrast of top band).

tools/extract_visual_embeddings.py: Compute OpenCLIP embeddings for keyframes under AELP2/reports/keyframes/* and

tools/fetch_campaign_placement_conversions.py: no docstring

tools/fetch_meta_adlibrary.py: Meta Ad Library fetcher

tools/fetch_meta_creatives.py: Fetch Meta Ad and AdCreative objects for historical ads listed in AELP2/reports/creative/*.json.

tools/fetch_searchapi_meta.py: Fetch Meta Ad Library results via SearchAPI.io (self-serve) and write CSVs that our

tools/fidelity_eval_roll.py: Rolling-origin evaluation for the temporal (Phase 2) simulator.

tools/finalize_gate.py: Finalize a candidate video with hard gates so low-quality outputs never publish.

tools/fit_cac_calibrator.py: Fit a monotonic calibrator mapping predicted CAC to actual CAC using isotonic regression.

tools/forecast_us_cac_volume.py: Monte Carlo CAC/volume forecast for Top-20 blueprints at $30k and $50k budgets.

tools/forward_forecast.py: Forward forecast check using Phase 2 (temporal) model.

tools/gen_eleven_vo.py: Generate VO lines with ElevenLabs v3 and save MP3s.

tools/gen_eleven_vo_pack.py: Generate a pack of VO lines from AELP2/prompts/vo_scripts.json using ElevenLabs.

tools/gen_runway_hooks.py: Generate 9:16 hook clips on Runway (Gen■4 Turbo) using image_to_video.

tools/gen_veo_hooks.py: Generate 9:16 hook clips on Vertex AI (Veo 3 Fast) and write to GCS.

tools/generate_advantage_manifest.py: Create a simple Advantage+ Creative manifest from finals.

tools/generate_balance_blueprints.py: Generate Balance-focused creative blueprints from Aura parental-controls concepts

tools/generate_blueprints.py: Generate synthetic creative blueprints (visual + copy/offer/CTA/placement) as

tools/generate_score_loop.py: Orchestrate generate -> feature -> score -> filter loop (stub).

tools/hourly_multipliers.py: Fetch account-level hourly CVR multipliers (normalized to 1.0 mean).

tools/import_proof_clips.py: Import real screen recordings and cut proof clips for slot B.

tools/import_vendor_meta_creatives.py: Normalize vendor exports (BigSpy/PowerAdSpy/SocialPeta) into GAELP creative_objects.

tools/join_dna_to_creatives.py: Join heuristic CreativeDNA-style features to historical ad items using ad_name and placement mix.

tools/join_finals_features.py: Join per-final features into one JSONL for analysis.

tools/journey_states_from_bq.py: Optional: Pull journey states from BigQuery if available.

tools/log_creative_dna.py: Write CreativeDNA + cost placeholders for produced finals; load to BigQuery if configured.

tools/make_meta_account_audit.py: Generate a Macro Account Audit checklist for Meta (offline).

tools/merge_copy_banks.py: Merge copy from Meta (copy_bank.json), Google Ads (google_ads_copy.json), Impact (impact_copy.json), and YouTube (youtube_copy.json).

tools/meta_campaign_bayes.py: Bayesian per-campaign MMM (Poisson with geometric adstock) for Meta campaigns.

tools/mine_meta_copy.py: no docstring

tools/mj_ingest.py: Ingest Midjourney backplates from a folder, normalize to 9:16, lightly denoise,

tools/normalize_audio.py: Normalize audio to -14 LUFS using ffmpeg loudnorm. If no audio, adds a silent track.

tools/offline_creative_search.py: Offline creative generation + simulator scoring scaffold.

tools/offline_rl_stub.py: Offline RL stub (bandit/IQL-lite): trains a logistic policy to predict P(win) from features and applies a conservative penalty.

tools/ope_config_weighting.py: OPE with config-based propensity reweighting (diagnostic).

tools/ope_topk.py: no docstring

tools/ope_upgrades.py: OPE upgrades: DM, IPS, SNIPS, DR, SWITCH, CAB; plus simple bandit-FQE.

tools/ope_uplift_topk.py: no docstring

tools/parse_brand_guide.py: Parse brand guide PDFs to extract a usable brand_config.json for the creative engine.

tools/policy_audit_live.py: Policy audit of current live adsets/campaigns vs META_POLICY_SETUP.md

tools/portfolio_selector.py: Portfolio selector: choose a per-campaign slate that maximizes predicted purchases under CAC caps and test budget share.

tools/propose_kb_non_claims.py: Propose non-claim benefit lines for each product KB from merged copy bank (Meta/Google/Impact/YouTube).

tools/pull_google_ads_copy.py: Pull Google Ads headlines/descriptions via the Google Ads API (read-only) using env credentials.

tools/pull_google_ads_copy_from_bq.py: Pull Google Ads ad names/headlines from BigQuery (preferred over API on this box).

tools/pull_google_ads_copy_rest.py: Pull Google Ads headlines/descriptions via REST (searchStream) using OAuth refresh token.

tools/pull_impact_copy.py: Pull top affiliate (Impact) copy snippets via Impact API.

tools/pull_impact_copy_from_bq.py: Pull affiliate (Impact) copy-like fields from BigQuery tables populated by AELP2 pipelines.

tools/qc_gates.py: Hard QC gates for candidate clips and finals.

tools/render_bayes_report.py: no docstring

tools/render_quiz_overlay.py: Render timed quiz-style overlay cards on a 9:16 video using ffmpeg drawbox/drawtext.

tools/render_sim_onepager.py: Render a simple, marketer-friendly one-pager (Markdown) summarizing simulator fidelity.

tools/report_policy_vs_mixed.py: Compare WBUA and dual-gate metrics on mixed vs policy-annotated data.

tools/rl_shadow_score.py: RL shadow scorer using temporal v2 simulator parameters.

tools/score_ad_items.py: Score ad items using success_config proxies (best-effort on available fields).

tools/score_new_ads.py: Score local final videos with the Meta-only new-ad ranker (light features).

tools/score_vendor_creatives.py: Score all creative_objects (including vendor-imported) with the trained new-ad ranker.

tools/selector_utility_baseline.py: Baseline utility selector: rank creatives by expected utility proxy and export Top-N per campaign (offline only).

tools/self_judge.py: Self-judging evaluator for creative quality.

tools/serve_previews.py: Serve AELP2/outputs/finals on http://127.0.0.1:8080 for local/SSH-tunnel preview.

tools/sim_fidelity_campaigns.py: Phase 1 fidelity: per-campaign heterogeneity

tools/sim_fidelity_campaigns_journey.py: Phase 3 fidelity: journey stages + recency proxy

tools/sim_fidelity_campaigns_temporal.py: Phase 2 fidelity: temporal patterns (weekday seasonality + frequency fatigue)

tools/sim_fidelity_campaigns_temporal_v2.py: Temporal simulator v2: time-decayed CVR priors + auto window (per-campaign)

tools/sim_fidelity_campaigns_temporal_v3.py: Temporal simulator v3: per-campaign hourly effect + fast-drift window rule

tools/sim_fidelity_eval.py: Simulation Fidelity Check (AELP ↔ AELP2)

tools/sim_fidelity_eval_empirical.py: no docstring

tools/sim_fidelity_journey_criteo.py: Journey + Criteo fidelity evaluation (no BigQuery writes).

tools/simulate_bandit_from_forecasts.py: Offline bandit simulation using the forecast distributions for Top-20 blueprints.

tools/sort_uploaded_assets.py: Sort a big "drop" folder of mixed digital assets into the repo structure.

tools/still_to_motion.py: Turn a still image (1080x1920 recommended) into a subtle-motion MP4.

tools/tag_ad_patterns.py: Tag ads with hook_type, emotion, proof_device, captions_present (heuristics on text),

tools/train_new_ad_ranker.py: Train a lightweight pairwise ranker on weekly pairs (feature diffs).

tools/tune_generator_priors.py: no docstring

tools/uplift_baseline_test.py: Historic baseline uplift test.

tools/validate_candidates.py: no docstring

tools/validate_kb.py: no docstring

tools/voice_library_add_prompt.py: no docstring

tools/weekly_fidelity_from_v3.py: Compute weekly (7-day aggregated) fidelity from sim_fidelity_campaigns_temporal_v3.json.

tools/weekly_relabel.py: Recompute weekly labels from creative_enriched/*.json and write to creative_weekly/*.json

tools/weekly_topN_dual.py: Export weekly Top-N creatives that pass the dual gate with a conservative lower-bound utility.

pipelines/ads_common.py: Shared utilities for Google Ads → BigQuery loaders.

pipelines/ads_mcc_coordinator.py: MCC Coordinator: Orchestrate Ads data loads across all child accounts.

pipelines/attribution_engine_stub.py: Real Attribution Engine Implementation for AELP2

pipelines/audience_expansion.py: Audience Expansion Tooling (shadow-only).

pipelines/auto_recalibration.py: Auto-recalibration (shadow-only): detect drift and log proposals.

pipelines/bandits_writer.py: Bandits Posteriors Writer (heuristic P0)

pipelines/bid_edit_proposals.py: Bid Edit Proposals (shadow-only, stub).

pipelines/bid_landscape_modeling.py: Bid Landscape Modeling (stub): derive CPC↔volume curves per campaign.

pipelines/bing_to_bq.py: Stub: Bing Ads to BigQuery adapter (pilot)

pipelines/build_affiliate_triggered.py: Affiliate-triggered (delayed-reward) series from GA4 export.

pipelines/build_ga_aligned_daily.py: Build GA-aligned KPI daily table that keeps GA as the majority signal and fills only

pipelines/build_impact_partner_domains.py: Build partner→domain mapping from Impact MediaPartners table.

pipelines/build_triggered_series.py: Build touch-aligned (delayed-reward) daily series from GA4 export.

pipelines/calibration_parallel_stub.py: Parallel calibration probes (stub): demonstrates multiprocessing fanout across probe params.

pipelines/calibration_stratified_views.py: Create calibration stratified views (by channel/device) for RL vs Ads.

pipelines/canary_monitoring.py: Canary Monitoring (shadow): detect anomalies and write ops_alerts.

pipelines/canary_timeline_writer.py: Writes a simple canary timeline to BQ for dashboard consumption.

pipelines/channel_attribution_r.py: ChannelAttribution (R) weekly job — containerized stub with BQ writer.

pipelines/channel_attribution_runner.py: Channel Attribution Runner (skeleton)

pipelines/check_data_quality.py: Basic data quality and freshness checks for core BigQuery tables.

pipelines/competitive_intel_ingest.py: Competitive Intelligence ingest (stub): ensure auction insights table exists.

pipelines/copy_optimizer_stub.py: Copy Optimization Loop (policy-safe, stub).

pipelines/cost_monitoring_stub.py: Cost monitoring (stub): compute daily cost and emit ops_alerts if exceeding cap.

pipelines/create_bq_views.py: Create standard BigQuery views for dashboards/subagents.

pipelines/create_channel_views.py: Create channel-specific daily views from ads_campaign_performance using advertising_channel_type.

pipelines/creative_ab_planner.py: Creative AB Planner (shadow-only): proposes a test and writes to BQ.

pipelines/creative_bandit_head.py: Creative Bandit Head (stub): logs decision proposals to ab_experiments (shadow-only).

pipelines/creative_embeddings_stub.py: Production Creative Embeddings System for AELP2

pipelines/creative_fatigue_alerts.py: Creative Fatigue Detection (stub): flags CTR/CVR decay over recent days.

pipelines/cross_platform_kpi_daily.py: Cross-Platform KPI Daily (stub): aggregates KPI metrics across platforms.

pipelines/dayparting_optimizer.py: Dayparting Optimizer (stub): propose hour/day schedule caps.

pipelines/delayed_conversions_stub.py: Production Delayed Conversions Processing for AELP2

pipelines/fidelity_evaluation.py: Fidelity Evaluation: Compare simulation/RL telemetry vs GA4/Ads (MAPE/RMSE/KS) and write results to BigQuery.

pipelines/fidelity_parallel_stub.py: Parallel fidelity evaluation (stub): shards dates and would call fidelity_evaluation per shard.

pipelines/ga4_backfill_audience_breakdown.py: WITH events AS (

pipelines/ga4_build_attribution.py: Build GA attribution daily tables from export using purchase-level cookies (first pass LND proxy):

pipelines/ga4_build_audience_breakdown.py: Build high-intent breakdown by channel (source/medium) from GA4 export.

pipelines/ga4_build_audiences.py: Build high-intent audiences from GA4 export:

pipelines/ga4_build_derived.py: Build GA-derived daily/monthly KPI tables and cohort summaries from GA4 export (events_*).

pipelines/ga4_extract_affiliate_clickids.py: Extract affiliate click IDs from GA4 export into a joinable table.

pipelines/ga4_lagged_attribution.py: GA4 Lagged Attribution Importer

pipelines/ga4_paths_attribution.py: Build multi-touch attribution from GA4 export:

pipelines/ga4_permissions_check.py: GA4 permissions check (dry-run friendly): verifies Data API access or provides guidance.

pipelines/ga4_to_bq.py: Ingest GA4 aggregates into BigQuery (ga4_aggregates).

pipelines/generate_qs_alerts.py: SELECT DATE(date) AS d, campaign_id, ANY_VALUE(campaign_name) AS name,

pipelines/generate_qs_fix_tickets.py: WITH base AS (

pipelines/google_ads_ad_performance_to_bq.py: Load Google Ads ad performance into BigQuery (ads_ad_performance).

pipelines/google_ads_adgroups_to_bq.py: Load Google Ads ad group performance into BigQuery (ads_ad_group_performance).

pipelines/google_ads_assets_to_bq.py: Google Ads Assets ingestion.

pipelines/google_ads_conversion_actions_to_bq.py: Load Google Ads conversion actions into BigQuery (ads_conversion_actions).

pipelines/google_ads_conversion_stats_by_action_to_bq.py: Load Google Ads conversion stats by conversion_action into BigQuery

pipelines/google_ads_discover_accounts.py: Discover accessible Google Ads accounts (including MCC/manager accounts).

pipelines/google_ads_geo_device_to_bq.py: Load Google Ads device-level performance into BigQuery (ads_geo_device_performance).

pipelines/google_ads_keywords_to_bq.py: Load Google Ads keyword performance into BigQuery (ads_keyword_performance).

pipelines/google_ads_mcc_to_bq.py: Enumerate all child accounts under an MCC and load Ads performance into BigQuery.

pipelines/google_ads_search_terms_to_bq.py: Load Google Ads search terms into BigQuery (ads_search_terms).

pipelines/google_ads_to_bq.py: Ingest Google Ads performance into BigQuery (ads_campaign_performance).

pipelines/google_recommendations_scanner.py: Google Ads Recommendations Scanner (safe, shadow-only).

pipelines/gsc_to_bq.py: Google Search Console → BigQuery (brand vs non-brand trend).

pipelines/hints_to_proposals.py: Promote policy hints to shadow proposals (HITL path, stub).

pipelines/impact_backfill_performance.py: Impact.com daily performance backfill (auto-discovery).

pipelines/impact_clicks_to_bq.py: no docstring

pipelines/impact_entities_to_bq.py: Impact.com entities → BigQuery (MediaPartners, Ads, Deals, Invoices, TrackingValueRequests).

pipelines/impact_run_report_to_bq.py: Try the official Run endpoint first; if forbidden, fall back to ReportExport.

pipelines/impact_to_bq.py: Impact.com (Impact Radius) → BigQuery loader.

pipelines/journey_path_summary.py: Summarize user journey paths and transition probabilities.

pipelines/journeys_populate.py: Journeys Populate (bootstrap)

pipelines/kpi_consistency_check.py: KPI Consistency Check: compares KPI-only view vs training episodes daily metrics.

pipelines/kpi_crosscheck.py: KPI Cross-check: compare ads_kpi_daily view vs aggregated ads_campaign_performance.

pipelines/linkedin_to_bq.py: no docstring

pipelines/load_affiliate_ach_costs.py: Load external affiliate (ACH) payouts into BigQuery to complete partner costs.

pipelines/load_promo_calendar.py: no docstring

pipelines/lp_ab_hooks_stub.py: Landing-Page A/B Hooks (stub): propose UTM cohorts and GA4 goal names.

pipelines/ltv_priors.py: LTV Priors Daily: computes simple 30/90-day LTV priors per segment from uplift scores.

pipelines/meta_to_bq.py: Meta → BigQuery loader (schema ensure + ingestion).

pipelines/mmm_attributed_service.py: MMM with Proper Delayed Reward Attribution (3-14 day windows)

pipelines/mmm_lightweightmmm.py: LightweightMMM runner with safe fallbacks and BQ logging.

pipelines/mmm_service.py: MMM v1 (Lightweight, dependency-free):

pipelines/model_registry_stub.py: Production Model Registry for AELP2

pipelines/module_runner.py: LP Module Runner (background job)

pipelines/namespace_refactor_report.py: Namespace Refactor Report (stub): scans repo for non-AELP2 imports and prints summary.

pipelines/offpolicy_eval.py: Off-policy evaluation (stub): compares hints vs realized outcomes.

pipelines/opportunity_outcomes.py: Opportunity Outcomes (stub): summarize approvals and subsequent performance.

pipelines/opportunity_scanner.py: Opportunity Scanner v1 (Google-first, shadow)

pipelines/ops_alerts_stub.py: Ops Alerts (stub): writes a placeholder alert row.

pipelines/parity_report.py: Parity Report (stub): compares ads_kpi_daily vs training_episodes_daily.

pipelines/permissions_check.py: Permissions & Accounts Checker (writes to BQ).

pipelines/platform_skeleton_log.py: no docstring

pipelines/policy_hints_writer.py: Policy Hints writer (stub): writes exploration/budget tilt hints to BQ.

pipelines/portfolio_optimizer.py: Portfolio Optimizer (stub): propose daily cross-campaign allocations under CAC cap.

pipelines/privacy_audit_stub.py: Privacy audit (stub): ensures free-text fields are not stored; writes summary row.

pipelines/propensity_uplift.py: Propensity/Uplift Bootstrap: writes `segment_scores_daily` using simple exposed vs unexposed deltas.

pipelines/quality_signal_daily.py: Quality Signal Daily (stub): computes a simple quality proxy (trial→paid, retention).

pipelines/quality_softgate_stub.py: Quality soft-gate (stub): reads quality_signal_daily and emits safety_events when below threshold.

pipelines/realtime_budget_pacer.py: Real-time budget pacer (stub): emits pacing proposals at minute cadence.

pipelines/reconcile_posthoc.py: Post-hoc reconciliation of RL vs Ads/GA4 metrics by date.

pipelines/rl_policy_hints_writer.py: RL Policy Hints Writer (bootstrap)

pipelines/robyn_runner.py: Robyn Validator Runner (skeleton)

pipelines/robyn_validator.py: Robyn weekly validator (containerized R) — runner stub with BQ summary write.

pipelines/rule_engine.py: Rule Engine (stub): evaluates safe rules and records actions (HITL required).

pipelines/security_audit.py: Security Audit (enhanced stub): records IAM/audit status notes and ADC context.

pipelines/segments_to_audiences.py: Segments → Audiences mapping (shadow-only).

pipelines/slo_watch_stub.py: SLO/Alerting stub: scans ops_flow_runs and safety_events for failures; emits ops_alerts.

pipelines/tiktok_to_bq.py: no docstring

pipelines/training_posthoc_reconciliation.py: Post-hoc reconciliation (lag-aware KPIs).

pipelines/trust_gates_evaluator.py: Trust Gates Evaluator: computes pass/fail for pilot gates and writes to BQ.

pipelines/uplift_eval.py: Uplift v1: Bootstrap segment uplift evaluation from journey tables (if present).

pipelines/upload_google_offline_conversions.py: Google Offline Conversions Upload (HITL-gated):

pipelines/upload_meta_capi_conversions.py: Meta CAPI Conversions Upload (HITL-gated): builds hashed payload and logs intent.

pipelines/users_db_stub.py: Production User Database Management for AELP2

pipelines/value_bridge.py: Value-Based Bidding Bridge (stubs)

pipelines/youtube_reach_planner.py: YouTube Reach Planner — real API when available.

10) What to change safely

• AOV assumptions in forecasting scripts

• Chip sets/filters in AELP2/config/*.yaml for vendor imports

• Number of creatives per package in Planner UI

Appendix A — API route index

/api/ab/exposure — exposure

/api/ads/creative — creative

/api/ads/google/create — google → create

/api/auth/[...nextauth] — [...nextauth]

/api/bq/ab-experiments — ab experiments

/api/bq/ab-exposures — ab exposures

/api/bq/ads — ads

/api/bq/ads/qs-is — ads → qs is

/api/bq/alerts — alerts

/api/bq/approvals/queue — approvals → queue

/api/bq/approvals/reject — approvals → reject

/api/bq/arms/posteriors — arms → posteriors

/api/bq/auctions/minutely — auctions → minutely

/api/bq/bid-edits — bid edits

/api/bq/bid-landscape — bid landscape

/api/bq/bidding — bidding

/api/bq/bidding_recent — bidding recent

/api/bq/bidding_replay — bidding replay

/api/bq/canaries — canaries

/api/bq/channel-attribution — channel attribution

/api/bq/copy-suggestions — copy suggestions

/api/bq/creative-variants — creative variants

/api/bq/creatives — creatives

/api/bq/cross-kpi — cross kpi

/api/bq/dayparting — dayparting

/api/bq/episodes — episodes

/api/bq/explore/cells — explore → cells

/api/bq/fidelity — fidelity

/api/bq/freshness — freshness

/api/bq/ga4/channels — ga4 → channels

/api/bq/halo — halo

/api/bq/headroom — headroom

/api/bq/impact/partner-opportunities — impact → partner opportunities

/api/bq/impact/partner-seed-harvest — impact → partner seed harvest

/api/bq/impact/seed-harvest — impact → seed harvest

/api/bq/impact/top-partners — impact → top partners

/api/bq/impact/triggered-by-partner — impact → triggered by partner

/api/bq/interference — interference

/api/bq/journeys/sankey — journeys → sankey

/api/bq/kpi — kpi

/api/bq/kpi/daily — kpi → daily

/api/bq/kpi/summary — kpi → summary

/api/bq/kpi/yesterday — kpi → yesterday

/api/bq/lp-ab — lp ab

/api/bq/lp/tests — lp → tests

/api/bq/ltv/summary — ltv → summary

/api/bq/mmm/allocations — mmm → allocations

/api/bq/mmm/channels — mmm → channels

/api/bq/mmm/curves — mmm → curves

/api/bq/mmm/whatif — mmm → whatif

/api/bq/offpolicy — offpolicy

/api/bq/opportunities — opportunities

/api/bq/ops-alerts — ops alerts

/api/bq/policy-enforcement — policy enforcement

/api/bq/portfolio-allocations — portfolio allocations

/api/bq/qa/cac/pacer — qa → cac → pacer

/api/bq/qa/enrollments/aligned — qa → enrollments → aligned

/api/bq/qa/enrollments/daily — qa → enrollments → daily

/api/bq/qa/enrollments/monthly — qa → enrollments → monthly

/api/bq/reach-estimates — reach estimates

/api/bq/safety — safety

/api/bq/subagents — subagents

/api/bq/value-uploads — value uploads

/api/canvas/list — list

/api/canvas/pin — pin

/api/canvas/unpin — unpin

/api/chat — chat

/api/chat/stream — stream

/api/connections/health — health

/api/control/ab-approve — ab approve

/api/control/ads-ingest — ads ingest

/api/control/apply-canary — apply canary

/api/control/apply-creative — apply creative

/api/control/audience/sync — audience → sync

/api/control/backfill/bid — backfill → bid

/api/control/backfill/ltv — backfill → ltv

/api/control/backfill/policy — backfill → policy

/api/control/backfill/rl — backfill → rl

/api/control/bandit-apply — bandit apply

/api/control/bandit-approve — bandit approve

/api/control/canary-rollback — canary rollback

/api/control/creative/enqueue — creative → enqueue

/api/control/creative/publish — creative → publish

/api/control/emergency-stop — emergency stop

/api/control/evaluate — evaluate

/api/control/fidelity-kpi — fidelity kpi

/api/control/ga4-attribution — ga4 attribution

/api/control/ga4-ingest — ga4 ingest

/api/control/kpi-lock — kpi lock

/api/control/lp/publish — lp → publish

/api/control/onboarding/backfill — onboarding → backfill

/api/control/onboarding/create — onboarding → create

/api/control/opportunity-approve — opportunity approve

/api/control/reach-planner — reach planner

/api/control/status — status

/api/control/switch-dataset — switch dataset

/api/control/test-account-ingest — test account ingest

/api/control/training-run — training run

/api/control/value-upload/google — value upload → google

/api/control/value-upload/meta — value upload → meta

/api/creative/generate — generate

/api/creative/publish — publish

/api/dataset — dataset

/api/kpi-source — kpi source

/api/media/generate — generate

/api/module/[slug]/result — [slug] → result

/api/module/[slug]/start — [slug] → start

/api/module/[slug]/status — [slug] → status

/api/ops/flows — flows

/api/planner/assets/briefs — assets → briefs

/api/planner/forecasts — forecasts

/api/planner/packages — packages

/api/planner/rl — rl

/api/planner/setup/[id] — setup → [id]

/api/planner/vendor-scores — vendor scores

/api/queue/processor — processor

/api/reports/executive — executive

/api/research/channels — channels

/api/scenarios/model — model

/api/training/start — start

Appendix B — Report schema samples

ad_balance_blueprints_top20.json: top_keys=['count', 'items'], items[0]_keys=['creative_id', 'p_win', 'lcb']

ad_blueprints_top20.json: top_keys=['count', 'items'], items[0]_keys=['creative_id', 'p_win', 'lcb', 'blueprint']

ad_level_accuracy.json: top_keys=['precision_at_5', 'precision_at_10', 'pairwise_win_rate', 'kendall_tau', 'n_campaigns', 'n_creatives', 'notes']

ad_level_accuracy_v22.json: top_keys=['precision_at_5', 'precision_at_10', 'pairwise_win_rate', 'stability_top10', 'n_campaigns', 'n_samples']

ad_level_accuracy_v22_details.json: list[0]_keys=['campaign_file', 'precision_at_5', 'precision_at_10', 'pairwise_win_rate']

ad_level_accuracy_v23.json: top_keys=['precision_at_5', 'precision_at_10', 'pairwise_win_rate', 'n_campaigns', 'n_samples']

ad_level_accuracy_v23_details.json: list[0]_keys=['campaign_file', 'precision_at_5', 'precision_at_10', 'pairwise_win_rate']

asset_briefs.json: top_keys=['items'], items[0]_keys=['creative_id', 'headline', 'overlay', 'motion', 'cta', 'placements', 'notes']

attention_scores.json: top_keys=['items'], items[0]_keys=['file', 'kf_gap_s', 'first3s_score']

audio_features.json: top_keys=['items'], items[0]_keys=['file', 'duration_s', 'channels', 'bit_rate', 'lufs']

cac_calibrator.json: top_keys=['x', 'y', 'note', 'n']

candidates_validation.json: top_keys=['checked', 'issues']

cascade_dr_topk.json: top_keys=['results']

conformal_topk_v24.json: top_keys=['selector', 'curves']

conformal_topk_weekly.json: top_keys=['K', 'alpha', 'coverage', 'n_eval', 'per_campaign']

copy_bank.json: top_keys=['titles', 'bodies']

copy_bank_merged.json: top_keys=['items', 'summary'], items[0]_keys=['text', 'sources', 'count']

creative_leaderboard.json: top_keys=['count', 'items'], items[0]_keys=['id', 'product', 'product_id', 'format', 'message', 'insight', 'hooks', 'copy_lines', 'cta', 'visuals', 'policy', 'sim_score']

creative_render_log.json: top_keys=['items'], items[0]_keys=['path', 'created_at', 'features', 'costs']

dual_gate_status.json: top_keys=['ok', 'fail', 'min_precision', 'min_yield', 'ts']

dual_gate_weekly.json: top_keys=['grid']

eval_protocol.json: top_keys=['window_days', 'train_days', 'test_days', 'split', 'cross_validation', 'labels', 'objectives', 'gates']

eval_protocol_weekly.json: top_keys=['window_days', 'aggregation', 'split', 'cross_validation', 'labels', 'gates']

google_ads_copy.json: top_keys=['error']

hourly_multipliers.json: top_keys=['since', 'until', 'hourly']

impact_copy.json: top_keys=['error']

journey_states_probe.json: top_keys=['project', 'dataset', 'tables', 'note', 'error']

kb_non_claim_suggestions.json: top_keys=['suggestions']

kb_validation.json: top_keys=['validated', 'errors']

new_ad_scores.json: top_keys=['items'], items[0]_keys=['file', 'p_win', 'lcb']

ope_config_weighted.json: top_keys=['results']

ope_upgrades_topk.json: top_keys=['results']

policy_audit.json: top_keys=['counts', 'non_compliant']

policy_uplift.json: top_keys=['cvr_uplift_ratio', 'cac_gain_ratio']

policy_vs_mixed_summary.json: top_keys=['mixed', 'policy_only']

rl_balance_pack.json: top_keys=['items'], items[0]_keys=['creative_id']

rl_offline_simulation.json: top_keys=['days', 'daily_budget', 'ranking', 'history']

rl_shadow_score.json: top_keys=['generated_at', 'results']

rl_test_pack.json: top_keys=['items'], items[0]_keys=['creative_id', 'motif', 'subject', 'palette', 'format', 'cta', 'placements', 'priors', 'forecasts']

runway_tasks.json: top_keys=['results']

selector_choice.json: top_keys=['selector', 'candidates']

sim_fidelity.json: top_keys=['days', 'mape_purchases_median', 'mape_cac_median']

sim_fidelity_campaigns.json: top_keys=['window', 'summary', 'daily', 'per_campaign']

sim_fidelity_campaigns_journey.json: top_keys=['summary', 'daily']

sim_fidelity_campaigns_temporal.json: top_keys=['summary', 'daily']

sim_fidelity_campaigns_temporal_v2.json: top_keys=['summary', 'daily', 'note']

sim_fidelity_campaigns_temporal_v3.json: top_keys=['summary', 'daily']

sim_fidelity_roll.json: top_keys=['splits']

sim_forward_forecast.json: top_keys=['train_days', 'hold_days', 'summary', 'daily']

sort_uploads_log.json: top_keys=['src', 'moved_count', 'moved', 'skipped']

target_cac_locked.json: top_keys=['targets', 'weeks']

topk_slate.json: top_keys=['items', 'note'], items[0]_keys=['file', 'p_win', 'lcb']

us_balance_forecasts.json: top_keys=['items', 'budgets'], items[0]_keys=['creative_id', 'p_win', 'lcb', 'budgets']

us_cac_volume_forecasts.json: top_keys=['budgets', 'items'], items[0]_keys=['creative_id', 'p_win', 'lcb', 'budgets']

us_meta_baselines.json: top_keys=['start_date', 'end_date', 'days', 'cpm_avg', 'cpm_p10', 'cpm_p50', 'cpm_p90', 'ctr_avg', 'ctr_p10', 'ctr_p50', 'ctr_p90', 'cvr_avg']

us_meta_baselines_by_place.json: top_keys=['start_date', 'end_date', 'items'], items_keys=['audience_network/an_classic', 'audience_network/rewarded_video', 'facebook/facebook_profile_feed', 'facebook/facebook_reels', 'facebook/facebook_reels_overlay', 'facebook/facebook_stories', 'facebook/feed', 'facebook/instream_video', 'facebook/marketplace', 'facebook/video_feeds', 'facebook/search', 'instagram/feed']

vendor_scores.json: top_keys=['items', 'count'], items[0]_keys=['creative_id', 'p_win', 'lcb']

vendor_top20.json: top_keys=['count', 'items'], items[0]_keys=['creative_id', 'p_win', 'lcb', 'title', 'ad_text', 'destination_url', 'page_id']

veo_tasks.json: top_keys=['results']

wbua_cluster_summary.json: top_keys=['wbua_cluster_holdout', 'pairs', 'ci95_groups', 'groups', 'weeks_counted']

wbua_forward.json: top_keys=['wbua_forward', 'pairs', 'ci95_groups', 'groups']

wbua_novel_summary.json: top_keys=['wbua_novel', 'pairs_novel', 'ci95_groups', 'groups']

wbua_summary.json: top_keys=['wbua', 'pairs', 'ci95', 'groups']

weekly_portfolio.json: top_keys=['portfolio']

weekly_topN_dual.json: top_keys=['topN']

youtube_copy.json: top_keys=['tried', 'count', 'items'], items[0]_keys=['channel', 'title']

Appendix C — Source file excerpts (first 10 lines each)

AELP2/__init__.py

```
"""AELP2 package root (scaffold)."""
```

AELP2/adapters/google_enhanced_conversions.py

```
import hashlib import json from typing import Dict, Any, List def _sha256(s: str)
-> str: return hashlib.sha256(s.encode('utf-8')).hexdigest() def
normalize_email(email: str) -> str:
```

AELP2/adapters/linkedin_adapter_stub.py

```
""" LinkedIn adapter stub: placeholder for future platform integration
(shadow-only). """ def apply_budget_change(*args, **kwargs): return {'ok': False,
'error': 'LinkedIn adapter not implemented (stub)'}
```

AELP2/adapters/meta_adapter_stub.py

```
""" Production Meta Ads API Adapter for AELP2 Real Meta Marketing API integration
with: - Campaign and ad set management - Budget optimization - Creative management
and publishing - Real-time performance data - No fallbacks or stub implementations
```

AELP2/adapters/meta_capi.py

```
import hashlib from typing import Dict, Any, List def _sha256(s: str) -> str:
return hashlib.sha256(s.encode('utf-8')).hexdigest() def normalize_email(email:
str) -> str: return (email or '').strip().lower()
```

AELP2/adapters/tiktok_adapter_stub.py

```
""" TikTok adapter stub: placeholder for future platform integration (shadow-only).
""" def apply_budget_change(*args, **kwargs): return {'ok': False, 'error': 'TikTok
adapter not implemented (stub)'}
```

AELP2/apps/dashboard/middleware.ts

```
import { withAuth } from 'next-auth/middleware' export default withAuth({
callbacks: { authorized: ({ token }) => !!token, }, }) const enabled =
process.env.ENABLE_AUTH === '1' export const config = {
```

AELP2/apps/dashboard/next-env.d.ts

```
/// <reference types="next" /> /// <reference types="next/image-types/global" /> //
NOTE: This file should not be edited // see
https://nextjs.org/docs/basic-features/typescript for more information.
```

AELP2/apps/dashboard/src/app/api/ab/exposure/route.ts

```
import { NextResponse } from 'next/server' import { BigQuery } from
'@google-cloud/bigquery' import { resolveDatasetForAction } from
'../../../../lib/dataset' export async function POST(req: Request) { try { const
body = await req.json().catch(() => ({})) as any const projectId =
process.env.GOOGLE_CLOUD_PROJECT as string const { dataset, allowed, mode, reason }
= resolveDatasetForAction('write') if (!allowed) return NextResponse.json({ ok:
false, error: reason, mode, dataset }, { status: 403 })
```

AELP2/apps/dashboard/src/app/api/ads/creative/route.ts

```
import { NextResponse } from 'next/server' import { BigQuery } from
'@google-cloud/bigquery' import { getDatasetFromCookie } from
'../../../../lib/dataset' export const dynamic = 'force-dynamic' export const
runtime = 'nodejs' async function lookupCustomerId(projectId: string, dataset:
string, campaignId: string): Promise<string | null> { try { const bq = new
BigQuery({ projectId })
```

AELP2/apps/dashboard/src/app/api/ads/google/create/route.ts

```
import { NextRequest, NextResponse } from 'next/server' import { BigQuery } from
'@google-cloud/bigquery' import { resolveDatasetForAction } from
'../../../../../lib/dataset' export const dynamic = 'force-dynamic' // Minimal
create: enqueue an RSA creation request to approvals queue (HITL) export async
function POST(req: NextRequest) { try { const { allowed, reason, dataset, mode } =
resolveDatasetForAction('write')
```

AELP2/apps/dashboard/src/app/api/auth/[...nextauth]/route.ts

```
import NextAuth from 'next-auth' import GoogleProvider from
'next-auth/providers/google' import Credentials from
'next-auth/providers/credentials' export const dynamic = 'force-dynamic' const
handler = NextAuth({ providers: [ ...(process.env.GOOGLE_CLIENT_ID &&
process.env.GOOGLE_CLIENT_SECRET ? [
```

AELP2/apps/dashboard/src/app/api/bq/ab-experiments/route.ts

```
import { NextResponse } from 'next/server' import { BigQuery } from
'@google-cloud/bigquery' import { getDatasetFromCookie } from
'../../../../lib/dataset' export const dynamic = 'force-dynamic' export async
function GET() { try { const projectId = process.env.GOOGLE_CLOUD_PROJECT! const {
dataset } = getDatasetFromCookie()
```

AELP2/apps/dashboard/src/app/api/bq/ab-exposures/route.ts

```
import { NextResponse } from 'next/server' import { BigQuery } from
'@google-cloud/bigquery' import { getDatasetFromCookie } from
'../../../../lib/dataset' export async function GET() { try { const projectId =
process.env.GOOGLE_CLOUD_PROJECT as string const { dataset } =
getDatasetFromCookie() const bq = new BigQuery({ projectId }) const sql = `
```

AELP2/apps/dashboard/src/app/api/bq/ads/qs-is/route.ts

```
import { NextResponse } from 'next/server' import { cookies } from 'next/headers'
import { DATASET_COOKIE, PROD_DATASET, SANDBOX_DATASET } from
'../../../../../lib/dataset' import { createBigQueryClient } from
'../../../../../lib/bigquery-client' export const dynamic = 'force-dynamic'
function classify(name: string | null): {brand: boolean} { // Brand only if
explicit token 'Brand' or known naming pattern (avoid matching 'Aura') const s =
String(name||'')
```

AELP2/apps/dashboard/src/app/api/bq/ads/route.ts

```
import { NextResponse } from 'next/server' import { BigQuery } from
'@google-cloud/bigquery' import { getDatasetFromCookie } from
'../../../../lib/dataset' export const dynamic = 'force-dynamic' export async
function GET() { const projectId = process.env.GOOGLE_CLOUD_PROJECT! const {
dataset } = getDatasetFromCookie() const bq = new BigQuery({ projectId })
```

AELP2/apps/dashboard/src/app/api/bq/alerts/route.ts

```
import { NextResponse } from 'next/server' import { BigQuery } from
'@google-cloud/bigquery' import { getDatasetFromCookie } from
'../../../../lib/dataset' export const dynamic = 'force-dynamic' export async
function GET() { try { const projectId = process.env.GOOGLE_CLOUD_PROJECT as string
const { dataset } = getDatasetFromCookie()
```

AELP2/apps/dashboard/src/app/api/bq/approvals/queue/route.ts

```
import { NextResponse } from 'next/server' import { BigQuery } from
'@google-cloud/bigquery' import { getDatasetFromCookie } from
'../../../../../lib/dataset' export const dynamic = 'force-dynamic' export async
function GET(req: Request) { try { const projectId =
process.env.GOOGLE_CLOUD_PROJECT as string const { dataset } =
getDatasetFromCookie()
```

AELP2/apps/dashboard/src/app/api/bq/approvals/reject/route.ts

```
import { NextResponse } from 'next/server' import { BigQuery } from
'@google-cloud/bigquery' import { getDatasetFromCookie } from
'../../../../../lib/dataset' export const dynamic = 'force-dynamic' export async
function POST(req: Request) { try { const body = await req.json().catch(()=>({}))
as any const run_id = String(body?.run_id || '')
```

AELP2/apps/dashboard/src/app/api/bq/arms/posteriors/route.ts

```
import { NextResponse } from 'next/server' import { BigQuery } from
'@google-cloud/bigquery' import { getDatasetFromCookie } from
'../../../../../lib/dataset' export const dynamic = 'force-dynamic' export async
function GET() { try { const projectId = process.env.GOOGLE_CLOUD_PROJECT as string
const { dataset } = getDatasetFromCookie()
```

AELP2/apps/dashboard/src/app/api/bq/auctions/minutely/route.ts

```
import { NextResponse } from 'next/server' import { BigQuery } from
'@google-cloud/bigquery' import { getDatasetFromCookie } from
'../../../../../lib/dataset' export const dynamic = 'force-dynamic' export async
function GET(req: Request) { try { const projectId =
process.env.GOOGLE_CLOUD_PROJECT! const { dataset } = getDatasetFromCookie()
```

AELP2/apps/dashboard/src/app/api/bq/bid-edits/route.ts

```
import { NextResponse } from 'next/server' import { BigQuery } from
'@google-cloud/bigquery' import { getDatasetFromCookie } from
'../../../../lib/dataset' export async function GET() { try { const projectId =
process.env.GOOGLE_CLOUD_PROJECT as string const { dataset } =
getDatasetFromCookie() const bq = new BigQuery({ projectId }) const [rows] = await
bq.query({ query: `
```

AELP2/apps/dashboard/src/app/api/bq/bid-landscape/route.ts

```
import { NextResponse } from 'next/server' import { BigQuery } from
'@google-cloud/bigquery' import { getDatasetFromCookie } from
'../../../../lib/dataset' export const dynamic = 'force-dynamic' export async
function GET() { try { const projectId = process.env.GOOGLE_CLOUD_PROJECT! const {
dataset } = getDatasetFromCookie()
```

AELP2/apps/dashboard/src/app/api/bq/bidding/route.ts

```
import { NextResponse } from 'next/server' import { BigQuery } from
'@google-cloud/bigquery' import { getDatasetFromCookie } from
'../../../../lib/dataset' export const dynamic = 'force-dynamic' export async
function GET() { const projectId = process.env.GOOGLE_CLOUD_PROJECT! const {
dataset } = getDatasetFromCookie() const bq = new BigQuery({ projectId })
```

AELP2/apps/dashboard/src/app/api/bq/bidding_recent/route.ts

```
import { NextResponse } from 'next/server' import { BigQuery } from
'@google-cloud/bigquery' import { getDatasetFromCookie } from
'../../../../lib/dataset' export const dynamic = 'force-dynamic' export async
function GET(request: Request) { const { searchParams } = new URL(request.url)
const limit = Math.min(parseInt(searchParams.get('limit') || '1000', 10), 5000)
const projectId = process.env.GOOGLE_CLOUD_PROJECT!
```

AELP2/apps/dashboard/src/app/api/bq/bidding_replay/route.ts

```
import { NextResponse } from 'next/server' import { BigQuery } from
'@google-cloud/bigquery' import { getDatasetFromCookie } from
'../../../../lib/dataset' export const dynamic = 'force-dynamic' export async
function GET(request: Request) { const { searchParams } = new URL(request.url)
const episodeId = searchParams.get('episode_id') const step =
searchParams.get('step')
```

AELP2/apps/dashboard/src/app/api/bq/canaries/route.ts

```
import { NextResponse } from 'next/server' import { BigQuery } from
'@google-cloud/bigquery' import { getDatasetFromCookie } from
'../../../../lib/dataset' export async function GET() { try { const projectId =
process.env.GOOGLE_CLOUD_PROJECT as string const { dataset } =
getDatasetFromCookie() const bq = new BigQuery({ projectId }) const sql = `
```

AELP2/apps/dashboard/src/app/api/bq/channel-attribution/route.ts

```
import { NextResponse } from 'next/server' import { BigQuery } from
'@google-cloud/bigquery' import { getDatasetFromCookie } from
'../../../../lib/dataset' export async function GET() { try { const projectId =
process.env.GOOGLE_CLOUD_PROJECT as string const { dataset } =
getDatasetFromCookie() const bq = new BigQuery({ projectId }) const sql = `
```

AELP2/apps/dashboard/src/app/api/bq/copy-suggestions/route.ts

```
import { NextResponse } from 'next/server' import { BigQuery } from
'@google-cloud/bigquery' import { getDatasetFromCookie } from
'../../../../lib/dataset' export async function GET() { try { const projectId =
process.env.GOOGLE_CLOUD_PROJECT as string const { dataset } =
getDatasetFromCookie() const bq = new BigQuery({ projectId }) const [rows] = await
bq.query({ query: `
```

AELP2/apps/dashboard/src/app/api/bq/creative-variants/route.ts

```
import { NextResponse } from 'next/server' import { BigQuery } from
'@google-cloud/bigquery' import { getDatasetFromCookie } from
'../../../../lib/dataset' export const dynamic = 'force-dynamic' export async
function GET() { try { const projectId = process.env.GOOGLE_CLOUD_PROJECT as string
const { dataset } = getDatasetFromCookie()
```

AELP2/apps/dashboard/src/app/api/bq/creatives/route.ts

```
import { NextResponse } from 'next/server' import { BigQuery } from
'@google-cloud/bigquery' import { getDatasetFromCookie } from
'../../../../lib/dataset' export const dynamic = 'force-dynamic' export async
function GET() { const projectId = process.env.GOOGLE_CLOUD_PROJECT! const {
dataset } = getDatasetFromCookie() const bq = new BigQuery({ projectId })
```

AELP2/apps/dashboard/src/app/api/bq/cross-kpi/route.ts

```
import { NextResponse } from 'next/server' import { BigQuery } from
'@google-cloud/bigquery' import { getDatasetFromCookie } from
'../../../../lib/dataset' export async function GET() { try { const projectId =
process.env.GOOGLE_CLOUD_PROJECT as string const { dataset } =
getDatasetFromCookie() const bq = new BigQuery({ projectId }) const [rows] = await
bq.query({ query: `
```

AELP2/apps/dashboard/src/app/api/bq/dayparting/route.ts

```
import { NextResponse } from 'next/server' import { BigQuery } from
'@google-cloud/bigquery' import { getDatasetFromCookie } from
'../../../../lib/dataset' export const dynamic = 'force-dynamic' export async
function GET() { try { const projectId = process.env.GOOGLE_CLOUD_PROJECT! const {
dataset } = getDatasetFromCookie()
```

AELP2/apps/dashboard/src/app/api/bq/episodes/route.ts

```
import { NextResponse } from 'next/server' import { BigQuery } from
'@google-cloud/bigquery' import { getDatasetFromCookie } from
'../../../../lib/dataset' export const dynamic = 'force-dynamic' export async
function GET() { const projectId = process.env.GOOGLE_CLOUD_PROJECT! const {
dataset } = getDatasetFromCookie() const bq = new BigQuery({ projectId })
```

AELP2/apps/dashboard/src/app/api/bq/explore/cells/route.ts

```
import { NextRequest, NextResponse } from 'next/server' import {
createBigQueryClient } from '../../../../../lib/bigquery-client' import {
resolveDatasetForAction, getDatasetFromCookie } from '../../../../../lib/dataset'
export const dynamic = 'force-dynamic' export async function GET() { try { const
projectId = process.env.GOOGLE_CLOUD_PROJECT as string const { dataset } =
getDatasetFromCookie()
```

AELP2/apps/dashboard/src/app/api/bq/fidelity/route.ts

```
import { NextResponse } from 'next/server' import { BigQuery } from
'@google-cloud/bigquery' import { getDatasetFromCookie } from
'../../../../lib/dataset' export const dynamic = 'force-dynamic' export async
function GET() { const projectId = process.env.GOOGLE_CLOUD_PROJECT! const {
dataset } = getDatasetFromCookie() const bq = new BigQuery({ projectId })
```

AELP2/apps/dashboard/src/app/api/bq/freshness/route.ts

```
import { NextResponse } from 'next/server' import { cookies } from 'next/headers'
import { DATASET_COOKIE, SANDBOX_DATASET, PROD_DATASET } from
'../../../../lib/dataset' import { createBigQueryClient } from
'../../../../lib/bigquery-client' export const dynamic = 'force-dynamic' export
async function GET() { try { const projectId = process.env.GOOGLE_CLOUD_PROJECT as
string
```

AELP2/apps/dashboard/src/app/api/bq/ga4/channels/route.ts

```
import { NextResponse } from 'next/server' import { BigQuery } from
'@google-cloud/bigquery' import { cookies } from 'next/headers' import {
DATASET_COOKIE, PROD_DATASET, SANDBOX_DATASET } from '../../../../../lib/dataset'
export const dynamic = 'force-dynamic' export async function GET(req: Request) {
try { const url = new URL(req.url)
```

AELP2/apps/dashboard/src/app/api/bq/halo/route.ts

```
import { NextResponse } from 'next/server' import { createBigQueryClient } from
'../../../../lib/bigquery-client' import { getDatasetFromCookie } from
'../../../../lib/dataset' export async function GET() { try { const projectId =
process.env.GOOGLE_CLOUD_PROJECT as string const { dataset } =
getDatasetFromCookie() const bq = createBigQueryClient(projectId) const [rows] =
await bq.query({ query: `SELECT date, exp_id, brand_lift, ci_low, ci_high, method
FROM \`${projectId}.${dataset}.halo_reads_daily\` ORDER BY date DESC LIMIT 28` })
```

AELP2/apps/dashboard/src/app/api/bq/headroom/route.ts

```
import { NextResponse } from 'next/server' import { createBigQueryClient } from
'../../../../lib/bigquery-client' import { getDatasetFromCookie } from
'../../../../lib/dataset' export async function GET() { try { const projectId =
process.env.GOOGLE_CLOUD_PROJECT as string const { dataset } =
getDatasetFromCookie() const bq = createBigQueryClient(projectId) // Use latest
mmm_allocations; optionally include impression share cap if available
```

AELP2/apps/dashboard/src/app/api/bq/impact/partner-opportunities/route.ts

```
import { NextRequest, NextResponse } from 'next/server' import { BigQuery } from
'@google-cloud/bigquery' export async function GET(req: NextRequest) { try { const
{ searchParams } = new URL(req.url) const days = parseInt(searchParams.get('days')
|| '60', 10) const minPayout = parseFloat(searchParams.get('min_payout') || '1000')
const minDays = parseInt(searchParams.get('min_days') || '14', 10) const corrThresh
= parseFloat(searchParams.get('corr') || '0.25')
```

AELP2/apps/dashboard/src/app/api/bq/impact/partner-seed-harvest/route.ts

```
import { NextRequest, NextResponse } from 'next/server' import { BigQuery } from
'@google-cloud/bigquery' export async function GET(req: NextRequest) { try { const
{ searchParams } = new URL(req.url) const partnerId =
searchParams.get('partner_id') const days = parseInt(searchParams.get('days') ||
'60', 10) if (!partnerId) return NextResponse.json({ error: 'partner_id required'
}, { status: 400 })
```

AELP2/apps/dashboard/src/app/api/bq/impact/seed-harvest/route.ts

```
import { NextResponse } from 'next/server' import { BigQuery } from
'@google-cloud/bigquery' export async function GET() { try { const projectId =
process.env.GOOGLE_CLOUD_PROJECT as string const dataset =
process.env.BIGQUERY_TRAINING_DATASET as string if (!projectId || !dataset) throw
new Error('Missing GOOGLE_CLOUD_PROJECT or BIGQUERY_TRAINING_DATASET') const bq =
new BigQuery({ projectId }) const [rows] = await bq.query({
```

AELP2/apps/dashboard/src/app/api/bq/impact/top-partners/route.ts

```
import { NextRequest, NextResponse } from 'next/server' import { BigQuery } from
'@google-cloud/bigquery' export async function GET(req: NextRequest) { try { const
{ searchParams } = new URL(req.url) const days = parseInt(searchParams.get('days')
```

```
|| '28', 10) const minPayout = parseFloat(searchParams.get('min_payout') || '0')
const limit = parseInt(searchParams.get('limit') || '20', 10)
```

AELP2/apps/dashboard/src/app/api/bq/impact/triggered-by-partner/route.ts

```
import { NextRequest, NextResponse } from 'next/server' import { BigQuery } from
'@google-cloud/bigquery' export async function GET(req: NextRequest) { try { const
{ searchParams } = new URL(req.url) const days = parseInt(searchParams.get('days')
|| '28', 10) const limit = parseInt(searchParams.get('limit') || '20', 10) const
projectId = process.env.GOOGLE_CLOUD_PROJECT as string const dataset =
process.env.BIGQUERY_TRAINING_DATASET as string
```

AELP2/apps/dashboard/src/app/api/bq/interference/route.ts

```
import { NextResponse } from 'next/server' import { createBigQueryClient } from
'../../../../lib/bigquery-client' import { getDatasetFromCookie } from
'../../../../lib/dataset' export async function GET() { try { const projectId =
process.env.GOOGLE_CLOUD_PROJECT as string const { dataset } =
getDatasetFromCookie() const bq = createBigQueryClient(projectId) const [rows] =
await bq.query({ query: `SELECT date, from_channel, to_channel, cannibalization,
lift FROM \`${projectId}.${dataset}.channel_interference_scores\` ORDER BY date
DESC LIMIT 28` })
```

AELP2/apps/dashboard/src/app/api/bq/journeys/sankey/route.ts

```
import { NextResponse } from 'next/server' import { BigQuery } from
'@google-cloud/bigquery' import { getDatasetFromCookie } from
'../../../../../lib/dataset' export const dynamic = 'force-dynamic' export async
function GET() { try { const projectId = process.env.GOOGLE_CLOUD_PROJECT! const {
dataset } = getDatasetFromCookie()
```

AELP2/apps/dashboard/src/app/api/bq/kpi/daily/route.ts

```
import { NextResponse } from 'next/server' import { BigQuery } from
'@google-cloud/bigquery' import { cookies } from 'next/headers' import {
DATASET_COOKIE, PROD_DATASET, SANDBOX_DATASET } from '../../../../../lib/dataset'
import { getKpiSourceFromCookie } from '../../../../lib/kpi' export const dynamic =
'force-dynamic' export async function GET(req: Request) { try {
```

AELP2/apps/dashboard/src/app/api/bq/kpi/route.ts

```
import { NextResponse } from 'next/server' import { BigQuery } from
'@google-cloud/bigquery' import { getDatasetFromCookie } from
'../../../../lib/dataset' export const dynamic = 'force-dynamic' export async
function GET() { const projectId = process.env.GOOGLE_CLOUD_PROJECT! const {
dataset } = getDatasetFromCookie() const bq = new BigQuery({ projectId })
```

AELP2/apps/dashboard/src/app/api/bq/kpi/summary/route.ts

```
import { NextResponse } from 'next/server' import { createBigQueryClient } from
'../../../../../lib/bigquery-client' import { cookies } from 'next/headers' import
{ DATASET_COOKIE, PROD_DATASET, SANDBOX_DATASET } from '../../../../../lib/dataset'
import { getKpiSourceFromCookie } from '../../../../lib/kpi' export const dynamic =
'force-dynamic' export async function GET() { try {
```

AELP2/apps/dashboard/src/app/api/bq/kpi/yesterday/route.ts

```
import { NextResponse } from 'next/server' import { createBigQueryClient } from
'../../../../../lib/bigquery-client' import { cookies } from 'next/headers' import
```

```
{ DATASET_COOKIE, PROD_DATASET, SANDBOX_DATASET } from '../../../../../lib/dataset'
import { getKpiSourceFromCookie } from '../../../../lib/kpi' export const dynamic =
'force-dynamic' export async function GET() { try {
```

AELP2/apps/dashboard/src/app/api/bq/lp/tests/route.ts

```
import { NextResponse } from 'next/server' import { getDatasetFromCookie } from
'../../../../../lib/dataset' import { createBigQueryClient } from
'../../../../../lib/bigquery-client' export const dynamic = 'force-dynamic' export
async function GET() { try { const projectId = process.env.GOOGLE_CLOUD_PROJECT as
string const { dataset } = getDatasetFromCookie()
```

AELP2/apps/dashboard/src/app/api/bq/lp-ab/route.ts

```
import { NextResponse } from 'next/server' import { BigQuery } from
'@google-cloud/bigquery' import { getDatasetFromCookie } from
'../../../../lib/dataset' export async function GET() { try { const projectId =
process.env.GOOGLE_CLOUD_PROJECT as string const { dataset } =
getDatasetFromCookie() const bq = new BigQuery({ projectId }) const [rows] = await
bq.query({ query: `
```

AELP2/apps/dashboard/src/app/api/bq/ltv/summary/route.ts

```
import { NextResponse } from 'next/server' import { BigQuery } from
'@google-cloud/bigquery' import { cookies } from 'next/headers' import {
DATASET_COOKIE, SANDBOX_DATASET, PROD_DATASET } from '../../../../../lib/dataset'
export const dynamic = 'force-dynamic' export async function GET() { try { const
projectId = process.env.GOOGLE_CLOUD_PROJECT as string
```

AELP2/apps/dashboard/src/app/api/bq/mmm/allocations/route.ts

```
import { NextResponse } from 'next/server' import { BigQuery } from
'@google-cloud/bigquery' import { getDatasetFromCookie } from
'../../../../../lib/dataset' export async function GET() { try { const projectId =
process.env.GOOGLE_CLOUD_PROJECT as string const { dataset } =
getDatasetFromCookie() const bq = new BigQuery({ projectId }) const sql = `
```

AELP2/apps/dashboard/src/app/api/bq/mmm/channels/route.ts

```
import { NextResponse } from 'next/server' import { BigQuery } from
'@google-cloud/bigquery' import { getDatasetFromCookie } from
'../../../../../lib/dataset' export const dynamic = 'force-dynamic' export async
function GET() { try { const projectId = process.env.GOOGLE_CLOUD_PROJECT! const {
dataset } = getDatasetFromCookie()
```

AELP2/apps/dashboard/src/app/api/bq/mmm/curves/route.ts

```
import { NextResponse } from 'next/server' import { BigQuery } from
'@google-cloud/bigquery' import { getDatasetFromCookie } from
'../../../../../lib/dataset' export const dynamic = 'force-dynamic' export async
function GET(req: Request) { try { const url = new URL(req.url) const channel =
url.searchParams.get('channel') || 'google_ads'
```

AELP2/apps/dashboard/src/app/api/bq/mmm/whatif/route.ts

```
import { NextRequest, NextResponse } from 'next/server' import { BigQuery } from
'@google-cloud/bigquery' import { getDatasetFromCookie } from
'../../../../../lib/dataset' export const dynamic = 'force-dynamic' export async
function GET(req: NextRequest) { try { const url = new URL(req.url) const channel =
```

```
url.searchParams.get('channel') || 'google_ads'
```

AELP2/apps/dashboard/src/app/api/bq/offpolicy/route.ts

```
import { NextResponse } from 'next/server' import { BigQuery } from
'@google-cloud/bigquery' import { getDatasetFromCookie } from
'../../../../lib/dataset' export const dynamic = 'force-dynamic' export async
function GET() { try { const projectId = process.env.GOOGLE_CLOUD_PROJECT! const {
dataset } = getDatasetFromCookie()
```

AELP2/apps/dashboard/src/app/api/bq/opportunities/route.ts

```
import { NextResponse } from 'next/server' import { BigQuery } from
'@google-cloud/bigquery' import { getDatasetFromCookie } from
'../../../../lib/dataset' export async function GET() { try { const projectId =
process.env.GOOGLE_CLOUD_PROJECT as string const { dataset } =
getDatasetFromCookie() const bq = new BigQuery({ projectId }) const sql = `
```

AELP2/apps/dashboard/src/app/api/bq/ops-alerts/route.ts

```
import { NextResponse } from 'next/server' import { BigQuery } from
'@google-cloud/bigquery' import { getDatasetFromCookie } from
'../../../../lib/dataset' export const dynamic = 'force-dynamic' export async
function GET() { try { const projectId = process.env.GOOGLE_CLOUD_PROJECT! const {
dataset } = getDatasetFromCookie()
```

AELP2/apps/dashboard/src/app/api/bq/policy-enforcement/route.ts

```
import { NextResponse } from 'next/server' import { BigQuery } from
'@google-cloud/bigquery' import { getDatasetFromCookie } from
'../../../../lib/dataset' export async function GET() { try { const projectId =
process.env.GOOGLE_CLOUD_PROJECT as string const { dataset } =
getDatasetFromCookie() const bq = new BigQuery({ projectId }) const [rows] = await
bq.query({ query: `
```

AELP2/apps/dashboard/src/app/api/bq/portfolio-allocations/route.ts

```
import { NextResponse } from 'next/server' import { BigQuery } from
'@google-cloud/bigquery' import { cookies } from 'next/headers' import {
DATASET_COOKIE, SANDBOX_DATASET, PROD_DATASET } from '../../../../lib/dataset'
export const dynamic = 'force-dynamic' export async function GET() { try { const
projectId = process.env.GOOGLE_CLOUD_PROJECT!
```

AELP2/apps/dashboard/src/app/api/bq/qa/cac/pacer/route.ts

```
import { NextResponse } from 'next/server' import { cookies } from 'next/headers'
import { DATASET_COOKIE, PROD_DATASET, SANDBOX_DATASET } from
'../../../../../../lib/dataset' import { createBigQueryClient } from
'../../../../../../lib/bigquery-client' export const dynamic = 'force-dynamic'
export async function GET(req: Request) { try { const url = new URL(req.url)
```

AELP2/apps/dashboard/src/app/api/bq/qa/enrollments/aligned/route.ts

```
import { NextResponse } from 'next/server' import { cookies } from 'next/headers'
import { DATASET_COOKIE, PROD_DATASET, SANDBOX_DATASET } from
'../../../../../../lib/dataset' import { createBigQueryClient } from
'../../../../../../lib/bigquery-client' export const dynamic = 'force-dynamic'
export async function GET(req: Request) { try { const url = new URL(req.url)
```

AELP2/apps/dashboard/src/app/api/bq/qa/enrollments/daily/route.ts

```
import { NextResponse } from 'next/server' import { cookies } from 'next/headers'
import { DATASET_COOKIE, PROD_DATASET, SANDBOX_DATASET } from
'../../../../../../lib/dataset' import { createBigQueryClient } from
'../../../../../../lib/bigquery-client' export const dynamic = 'force-dynamic'
export async function GET(req: Request) { try { const url = new URL(req.url)
```

AELP2/apps/dashboard/src/app/api/bq/qa/enrollments/monthly/route.ts

```
import { NextResponse } from 'next/server' import { cookies } from 'next/headers'
import { DATASET_COOKIE, PROD_DATASET, SANDBOX_DATASET } from
'../../../../../../lib/dataset' import { createBigQueryClient } from
'../../../../../../lib/bigquery-client' export const dynamic = 'force-dynamic'
export async function GET() { try { const projectId =
process.env.GOOGLE_CLOUD_PROJECT as string
```

AELP2/apps/dashboard/src/app/api/bq/reach-estimates/route.ts

```
import { NextResponse } from 'next/server' import { BigQuery } from
'@google-cloud/bigquery' import { getDatasetFromCookie } from
'../../../../lib/dataset' export async function GET() { try { const projectId =
process.env.GOOGLE_CLOUD_PROJECT as string const { dataset } =
getDatasetFromCookie() const bq = new BigQuery({ projectId }) const [rows] = await
bq.query({ query: `
```

AELP2/apps/dashboard/src/app/api/bq/safety/route.ts

```
import { NextResponse } from 'next/server' import { BigQuery } from
'@google-cloud/bigquery' import { getDatasetFromCookie } from
'../../../../lib/dataset' export const dynamic = 'force-dynamic' export async
function GET() { const projectId = process.env.GOOGLE_CLOUD_PROJECT! const {
dataset } = getDatasetFromCookie() const bq = new BigQuery({ projectId })
```

AELP2/apps/dashboard/src/app/api/bq/subagents/route.ts

```
import { NextResponse } from 'next/server' import { BigQuery } from
'@google-cloud/bigquery' import { getDatasetFromCookie } from
'../../../../lib/dataset' export const dynamic = 'force-dynamic' export async
function GET() { const projectId = process.env.GOOGLE_CLOUD_PROJECT! const {
dataset } = getDatasetFromCookie() const bq = new BigQuery({ projectId })
```

AELP2/apps/dashboard/src/app/api/bq/value-uploads/route.ts

```
import { NextRequest, NextResponse } from 'next/server' import { BigQuery } from
'@google-cloud/bigquery' import { getDatasetFromCookie, resolveDatasetForAction }
from '../../../../lib/dataset' export const dynamic = 'force-dynamic' function bq()
{ const projectId = process.env.GOOGLE_CLOUD_PROJECT! return new BigQuery({
projectId }) }
```

AELP2/apps/dashboard/src/app/api/canvas/list/route.ts

```
import { NextResponse } from 'next/server' import { BigQuery } from
'@google-cloud/bigquery' export const dynamic = 'force-dynamic' export async
function GET() { try { const projectId = process.env.GOOGLE_CLOUD_PROJECT as string
const usersDs = process.env.BIGQUERY_USERS_DATASET as string const bq = new
BigQuery({ projectId })
```

AELP2/apps/dashboard/src/app/api/canvas/pin/route.ts

```
import { NextRequest, NextResponse } from 'next/server' import { BigQuery } from
'@google-cloud/bigquery' export const dynamic = 'force-dynamic' export async
function POST(req: NextRequest) { try { const form = await req.formData() const
title = String(form.get('title') || 'Chart') const vizStr = String(form.get('viz')
|| '')
```

AELP2/apps/dashboard/src/app/api/canvas/unpin/route.ts

```
import { NextRequest, NextResponse } from 'next/server' import { BigQuery } from
'@google-cloud/bigquery' export const dynamic = 'force-dynamic' export async
function POST(req: NextRequest) { try { const form = await req.formData() const id
= String(form.get('id') || '') if (!id) return NextResponse.json({ ok: false,
error: 'id required' }, { status: 400 })
```

AELP2/apps/dashboard/src/app/api/chat/route.ts

```
import { NextRequest, NextResponse } from 'next/server' import { BigQuery } from
'@google-cloud/bigquery' import { serializeBigQueryRows } from
'../../../lib/bigquery-serializer' import { DATASET_COOKIE, PROD_DATASET,
SANDBOX_DATASET } from '../../../lib/dataset' import { cookies } from
'next/headers' export const dynamic = 'force-dynamic' async function buildContext()
{ const projectId = process.env.GOOGLE_CLOUD_PROJECT as string
```

AELP2/apps/dashboard/src/app/api/chat/stream/route.ts

```
import { NextRequest } from 'next/server' export const dynamic = 'force-dynamic'
export async function POST(req: NextRequest) { const { absoluteUrl } = await
import('../../../lib/url') try { const body = await req.json().catch(()=>({}))
const messages = Array.isArray(body?.messages) ? body.messages : [] // call
existing /api/chat for a full reply, then stream it in chunks
```

AELP2/apps/dashboard/src/app/api/connections/health/route.ts

```
import { NextResponse } from 'next/server' import { getDatasetFromCookie } from
'../../../../lib/dataset' export async function GET() { const envDataset =
process.env.BIGQUERY_TRAINING_DATASET || null const { dataset, mode } =
getDatasetFromCookie() const checks = { project: process.env.GOOGLE_CLOUD_PROJECT
|| null, dataset_selected: dataset, dataset_mode: mode,
```

AELP2/apps/dashboard/src/app/api/control/ab-approve/route.ts

```
import { NextResponse } from 'next/server' import { BigQuery } from
'@google-cloud/bigquery' import { resolveDatasetForAction } from
'../../../../lib/dataset' export async function POST(req: Request) { try { let
body: any = {} const ct = req.headers.get('content-type') || '' if
(ct.includes('application/json')) body = await req.json().catch(()=>({})) else if
(ct.includes('application/x-www-form-urlencoded') ||
ct.includes('multipart/form-data')) {
```

AELP2/apps/dashboard/src/app/api/control/ads-ingest/route.ts

```
import { NextResponse } from 'next/server' import { resolveDatasetForAction } from
'../../../../lib/dataset' import { spawn } from 'node:child_process' import path
from 'node:path' export const dynamic = 'force-dynamic' export async function
POST(req: Request) { // Safety: block writes on prod const { dataset, mode,
allowed, reason } = resolveDatasetForAction('write')
```

AELP2/apps/dashboard/src/app/api/control/apply-canary/route.ts

```
import { NextResponse } from 'next/server' import { spawn } from 'child_process'
export async function POST(req: Request) { try { const body = await
req.json().catch(() => ({})) as any const ids = String(body?.campaign_ids ||
process.env.AELP2_GOOGLE_CANARY_CAMPAIGN_IDS || '') const direction =
String(body?.direction || process.env.AELP2_CANARY_BUDGET_DIRECTION || 'down') if
(!ids) return NextResponse.json({ ok: false, error: 'campaign_ids required' }, {
status: 400 }) if ((process.env.GATES_ENABLED || '1') === '1' &&
(process.env.AELP2_ALLOW_GOOGLE_MUTATIONS || '0') !== '1') {
```

AELP2/apps/dashboard/src/app/api/control/apply-creative/route.ts

```
import { NextResponse } from 'next/server' import { spawn } from 'child_process'
export async function POST(req: Request) { try { const body = await
req.json().catch(()=> ({})) as any const change = JSON.stringify(body?.change ||
{}) if ((process.env.GATES_ENABLED || '1') === '1' &&
(process.env.AELP2_ALLOW_BANDIT_MUTATIONS || '0') !== '1') { return
NextResponse.json({ ok: false, error: 'flag_denied: AELP2_ALLOW_BANDIT_MUTATIONS=0'
}, { status: 200 }) }
```

AELP2/apps/dashboard/src/app/api/control/audience/sync/route.ts

```
import { NextResponse } from 'next/server' import { resolveDatasetForAction } from
'../../../../../lib/dataset' export const dynamic = 'force-dynamic' export async
function POST() { const { allowed, reason, mode, dataset } =
resolveDatasetForAction('write') if (!allowed) return NextResponse.json({ ok:
false, error: reason, mode, dataset }, { status: 403 }) // Stub: enqueue an
audience sync job (no-op here) return NextResponse.json({ ok: true, queued: true,
dataset })
```

AELP2/apps/dashboard/src/app/api/control/backfill/bid/route.ts

```
import { NextResponse } from 'next/server' import { resolveDatasetForAction } from
'../../../../../lib/dataset' import { BigQuery } from '@google-cloud/bigquery'
export const dynamic = 'force-dynamic' export async function POST() { try { const {
allowed, reason, dataset, mode } = resolveDatasetForAction('write') if (!allowed)
return NextResponse.json({ ok:false, error: reason, mode, dataset }, { status: 403
})
```

AELP2/apps/dashboard/src/app/api/control/backfill/ltv/route.ts

```
import { NextResponse } from 'next/server' import { resolveDatasetForAction } from
'../../../../../lib/dataset' import { BigQuery } from '@google-cloud/bigquery'
export const dynamic = 'force-dynamic' export async function POST() { try { const {
allowed, reason, dataset, mode } = resolveDatasetForAction('write') if (!allowed)
return NextResponse.json({ ok:false, error: reason, mode, dataset }, { status: 403
})
```

AELP2/apps/dashboard/src/app/api/control/backfill/policy/route.ts

```
import { NextResponse } from 'next/server' import { resolveDatasetForAction } from
'../../../../../lib/dataset' import { BigQuery } from '@google-cloud/bigquery'
export const dynamic = 'force-dynamic' export async function POST() { try { const {
allowed, reason, dataset, mode } = resolveDatasetForAction('write') if (!allowed)
return NextResponse.json({ ok:false, error: reason, mode, dataset }, { status: 403
})
```

AELP2/apps/dashboard/src/app/api/control/backfill/rl/route.ts

```
import { NextResponse } from 'next/server' import { resolveDatasetForAction } from
'../../../../../lib/dataset' import { BigQuery } from '@google-cloud/bigquery'
export const dynamic = 'force-dynamic' export async function POST() { try { const {
allowed, reason, dataset, mode } = resolveDatasetForAction('write') if (!allowed)
return NextResponse.json({ ok:false, error: reason, mode, dataset }, { status: 403
})
```

AELP2/apps/dashboard/src/app/api/control/bandit-apply/route.ts

```
import { NextResponse } from 'next/server' import { spawn } from 'child_process'
export async function POST(req: Request) { try { const body = await
req.json().catch(() => ({})) as any const lookback = String(body?.lookback || '30')
if ((process.env.GATES_ENABLED || '1') === '1' &&
(process.env.AELP2_ALLOW_BANDIT_MUTATIONS || '0') !== '1') { return
NextResponse.json({ ok: false, error: 'flag_denied: AELP2_ALLOW_BANDIT_MUTATIONS=0'
}, { status: 200 }) }
```

AELP2/apps/dashboard/src/app/api/control/bandit-approve/route.ts

```
import { NextResponse } from 'next/server' import { BigQuery } from
'@google-cloud/bigquery' import { resolveDatasetForAction } from
'../../../../lib/dataset' export async function POST(req: Request) { try { const
body = await req.json().catch(() => ({})) as any const projectId =
process.env.GOOGLE_CLOUD_PROJECT as string const { dataset, allowed, mode, reason }
= resolveDatasetForAction('write') if (!allowed) return NextResponse.json({ ok:
false, error: reason, mode, dataset }, { status: 403 })
```

AELP2/apps/dashboard/src/app/api/control/canary-rollback/route.ts

```
import { NextResponse } from 'next/server' import { spawn } from 'child_process'
export async function POST() { try { const py = process.env.PYTHON_BIN || 'python3'
const child = spawn(py, ['AELP2/scripts/canary_rollback.py'], { env: {
...process.env }}) let out = ''; let err = '' child.stdout.on('data', (d)=> out +=
d.toString()) child.stderr.on('data', (d)=> err += d.toString())
```

AELP2/apps/dashboard/src/app/api/control/creative/enqueue/route.ts

```
import { NextRequest, NextResponse } from 'next/server' import { BigQuery } from
'@google-cloud/bigquery' import { resolveDatasetForAction } from
'../../../../../lib/dataset' export async function POST(req: NextRequest) { try {
const { dataset, allowed, reason } = resolveDatasetForAction('write') if (!allowed)
return NextResponse.json({ ok:false, error: reason }, { status: 403 }) const body =
await req.json().catch(()=>({})) as any const platform =
String(body?.platform||'google_ads')
```

AELP2/apps/dashboard/src/app/api/control/creative/publish/route.ts

```
import { NextRequest, NextResponse } from 'next/server' import { BigQuery } from
'@google-cloud/bigquery' import { resolveDatasetForAction } from
'../../../../../lib/dataset' export const dynamic = 'force-dynamic' async function
getAccessToken(client_id: string, client_secret: string, refresh_token: string):
Promise<string> { const params = new URLSearchParams({ client_id, client_secret,
refresh_token, grant_type: 'refresh_token' }) const r = await
fetch('https://oauth2.googleapis.com/token', { method:'POST',
headers:{'content-type':'application/x-www-form-urlencoded'}, body:
params.toString() }) const j = await r.json(); if(!r.ok) throw new
```

```
Error(j?.error_description||'oauth token error'); return j.access_token
```

AELP2/apps/dashboard/src/app/api/control/emergency-stop/route.ts

```
import { NextResponse } from 'next/server' import { spawn } from 'child_process'
export async function POST(req: Request) { const body = await
req.json().catch(()=>({})) as any const reason = String(body?.reason ||
'manual_stop') const note = String(body?.note || 'dashboard') const py =
process.env.PYTHON_BIN || 'python3' const args =
['AELP2/scripts/emergency_stop.py', '--reason', reason, '--note', note] const child
= spawn(py, args, { env: { ...process.env }})
```

AELP2/apps/dashboard/src/app/api/control/evaluate/route.ts

```
import { NextRequest, NextResponse } from 'next/server' export async function
POST(req: NextRequest) { try { const body = await req.json().catch(()=>({})) as any
const changePct = Number(body?.change_pct||0) const todayPct =
Number(body?.today_pct||0) const perChangeCap =
Number(process.env.AELP2_PER_CHANGE_CAP || '5') const dailyCap =
Number(process.env.AELP2_DAILY_CAP || '10') const ok = (changePct <= perChangeCap)
&& ((todayPct + changePct) <= dailyCap)
```

AELP2/apps/dashboard/src/app/api/control/fidelity-kpi/route.ts

```
import { NextResponse } from 'next/server' import { BigQuery } from
'@google-cloud/bigquery' import { resolveDatasetForAction } from
'../../../../lib/dataset' export const dynamic = 'force-dynamic' export async
function POST() { const { dataset, mode, allowed, reason } =
resolveDatasetForAction('write') if (!allowed) return NextResponse.json({ ok:
false, error: reason, mode, dataset }, { status: 403 })
```

AELP2/apps/dashboard/src/app/api/control/ga4-attribution/route.ts

```
import { NextResponse } from 'next/server' import { resolveDatasetForAction } from
'../../../../lib/dataset' import { spawn } from 'node:child_process' export const
dynamic = 'force-dynamic' export async function POST() { const { dataset, mode,
allowed, reason } = resolveDatasetForAction('write') if (!allowed) return
NextResponse.json({ ok: false, error: reason, mode, dataset }, { status: 403 })
```

AELP2/apps/dashboard/src/app/api/control/ga4-ingest/route.ts

```
import { NextResponse } from 'next/server' import { resolveDatasetForAction } from
'../../../../lib/dataset' import { spawn } from 'node:child_process' import path
from 'node:path' export const dynamic = 'force-dynamic' export async function
POST() { const { dataset, mode, allowed, reason } =
resolveDatasetForAction('write') if (!allowed) return NextResponse.json({ ok:
false, error: reason, mode, dataset }, { status: 403 })
```

AELP2/apps/dashboard/src/app/api/control/kpi-lock/route.ts

```
import { NextResponse } from 'next/server' import { BigQuery } from
'@google-cloud/bigquery' import { resolveDatasetForAction } from
'../../../../lib/dataset' export const dynamic = 'force-dynamic' async function
createOrReplaceView(bq: BigQuery, projectId: string, viewFqtn: string, sql: string)
{ // Prefer DDL to avoid table APIs across permissions const ddl = `CREATE OR
REPLACE VIEW \`${viewFqtn}\` AS\n${sql}` await bq.query({ query: ddl })
```

AELP2/apps/dashboard/src/app/api/control/lp/publish/route.ts

```
import { NextRequest, NextResponse } from 'next/server' import { BigQuery } from
'@google-cloud/bigquery' import { resolveDatasetForAction } from
'../../../../../lib/dataset' export const dynamic = 'force-dynamic' function
uid(prefix='lp_') { return `${prefix}${Math.random().toString(36).slice(2,10)}` }
```

AELP2/apps/dashboard/src/app/api/control/onboarding/backfill/route.ts

```
import { NextResponse } from 'next/server' import { spawn } from 'child_process'
const MAP: Record<string, string> = { meta: 'AELP2.pipelines.meta_to_bq', linkedin:
'AELP2.pipelines.linkedin_to_bq', tiktok: 'AELP2.pipelines.tiktok_to_bq', } export
async function POST(req: Request) {
```

AELP2/apps/dashboard/src/app/api/control/onboarding/create/route.ts

```
import { NextResponse } from 'next/server' import { spawn } from 'child_process'
export async function POST(req: Request) { const { searchParams } = new
URL(req.url) const platform = String(searchParams.get('platform') || '') try {
const env = { ...process.env } const py = process.env.PYTHON_BIN || 'python3' //
Log skeleton to BQ (shadow)
```

AELP2/apps/dashboard/src/app/api/control/opportunity-approve/route.ts

```
import { NextResponse } from 'next/server' import { BigQuery } from
'@google-cloud/bigquery' import { resolveDatasetForAction } from
'../../../../lib/dataset' export async function POST(req: Request) { try { let
body: any = {} const ct = req.headers.get('content-type') || '' if
(ct.includes('application/json')) body = await req.json().catch(() => ({})) else if
(ct.includes('application/x-www-form-urlencoded') ||
ct.includes('multipart/form-data')) {
```

AELP2/apps/dashboard/src/app/api/control/reach-planner/route.ts

```
import { NextResponse } from 'next/server' import { spawn } from 'child_process'
export async function POST() { try { const env = { ...process.env } const py =
process.env.PYTHON_BIN || 'python3' const child = spawn(py, ['-m',
'AELP2.pipelines.youtube_reach_planner'], { env }) let out = '' let err = ''
```

AELP2/apps/dashboard/src/app/api/control/status/route.ts

```
import { NextResponse } from 'next/server' import { BigQuery } from
'@google-cloud/bigquery' import { getDatasetFromCookie } from
'../../../../lib/dataset' export const dynamic = 'force-dynamic' export async
function GET() { const projectId = process.env.GOOGLE_CLOUD_PROJECT as string const
{ dataset } = getDatasetFromCookie() const bq = new BigQuery({ projectId })
```

AELP2/apps/dashboard/src/app/api/control/switch-dataset/route.ts

```
import { NextResponse } from 'next/server' export async function POST(req: Request)
{ const body = await req.json().catch(()=>({})) as any const dataset =
String(body?.dataset || '') if (!dataset) return NextResponse.json({ ok: false,
error: 'dataset required' }, { status: 200 }) // This endpoint only acknowledges
intent; callers must restart app with env. return NextResponse.json({ ok: true,
message: `Switch to dataset ${dataset} acknowledged (restart required).` }) }
```

AELP2/apps/dashboard/src/app/api/control/test-account-ingest/route.ts

```
import { NextResponse } from 'next/server' import { spawn } from 'child_process'
export async function POST() { const sh = 'bash' const args =
['AELP2/scripts/run_ads_ingestion.sh'] const env = { ...process.env, DRY_RUN:
```

```
process.env.DRY_RUN || '1' } const child = spawn(sh, args, { env }) let out = '';
let err = '' child.stdout.on('data', (d)=> out += d.toString())
```

AELP2/apps/dashboard/src/app/api/control/training-run/route.ts

```
import { NextResponse } from 'next/server' import { resolveDatasetForAction } from
'../../../../lib/dataset' import { spawn } from 'node:child_process' export const
dynamic = 'force-dynamic' export async function POST(req: Request) { const {
dataset, mode, allowed, reason } = resolveDatasetForAction('write') if (!allowed)
return NextResponse.json({ ok: false, error: reason, mode, dataset }, { status: 403
})
```

AELP2/apps/dashboard/src/app/api/control/value-upload/google/route.ts

```
import { NextResponse } from 'next/server' import { spawn } from
'node:child_process' export async function POST() { try { if
((process.env.GATES_ENABLED || '1') === '1' &&
(process.env.AELP2_ALLOW_VALUE_UPLOADS || '0') !== '1') { return
NextResponse.json({ ok: false, error: 'flag_denied: AELP2_ALLOW_VALUE_UPLOADS=0' },
{ status: 200 }) } const py = process.env.PYTHON_BIN || 'python3' const child =
spawn(py, ['-m', 'AELP2.pipelines.upload_google_offline_conversions'], { env: {
...process.env } })
```

AELP2/apps/dashboard/src/app/api/control/value-upload/meta/route.ts

```
import { NextResponse } from 'next/server' import { spawn } from
'node:child_process' export async function POST() { try { if
((process.env.GATES_ENABLED || '1') === '1' &&
(process.env.AELP2_ALLOW_VALUE_UPLOADS || '0') !== '1') { return
NextResponse.json({ ok: false, error: 'flag_denied: AELP2_ALLOW_VALUE_UPLOADS=0' },
{ status: 200 }) } const py = process.env.PYTHON_BIN || 'python3' const child =
spawn(py, ['-m', 'AELP2.pipelines.upload_meta_capi_conversions'], { env: {
...process.env } })
```

AELP2/apps/dashboard/src/app/api/creative/generate/route.ts

```
import { NextRequest, NextResponse } from 'next/server' import Anthropic from
'@anthropic-ai/sdk' const anthropic = new Anthropic({ apiKey:
process.env.ANTHROPIC_API_KEY!, }) export async function POST(req: NextRequest) {
try { const body = await req.json()
```

AELP2/apps/dashboard/src/app/api/creative/publish/route.ts

```
import { NextRequest, NextResponse } from 'next/server' import { GoogleAdsApi }
from 'google-ads-api' // Initialize Google Ads client const client = new
GoogleAdsApi({ client_id: process.env.GOOGLE_ADS_CLIENT_ID!, client_secret:
process.env.GOOGLE_ADS_CLIENT_SECRET!, developer_token:
process.env.GOOGLE_ADS_DEVELOPER_TOKEN!, })
```

AELP2/apps/dashboard/src/app/api/dataset/route.ts

```
import { cookies } from 'next/headers' import { NextResponse } from 'next/server'
import { DATASET_COOKIE, SANDBOX_DATASET, PROD_DATASET } from
'../../../lib/dataset' export const dynamic = 'force-dynamic' export async function
GET() { const c = cookies() const mode = c.get(DATASET_COOKIE)?.value === 'prod' ?
'prod' : 'sandbox' const dataset = mode === 'prod' ? PROD_DATASET : SANDBOX_DATASET
```

AELP2/apps/dashboard/src/app/api/kpi-source/route.ts

```
import { NextResponse } from 'next/server' import { cookies } from 'next/headers'
import { KPI_COOKIE, KpiSource } from '../../lib/kpi' export const dynamic =
'force-dynamic' export async function GET() { const c =
cookies().get(KPI_COOKIE)?.value || 'ga4_google' return NextResponse.json({ source:
c }) }
```

AELP2/apps/dashboard/src/app/api/media/generate/route.ts

```
import { NextRequest, NextResponse } from 'next/server' export const dynamic =
'force-dynamic' export async function POST(req: NextRequest) { try { const body =
await req.json().catch(()=>({})) as any const prompt = String(body?.prompt || '')
if (!prompt) return NextResponse.json({ ok:false, error:'prompt required' }, {
status: 400 }) const key = process.env.OPENAI_API_KEY as string
```

AELP2/apps/dashboard/src/app/api/module/[slug]/result/route.ts

```
import { NextRequest, NextResponse } from 'next/server' import {
createBigQueryClient } from '../../../../../lib/bigquery-client' export async
function GET(req: NextRequest, { params }: { params: { slug: string } }) { try {
const url = new URL(req.url) const runId =
String(url.searchParams.get('run_id')||'') if (!runId) return NextResponse.json({
ok:false, error: 'missing_run_id' }, { status: 400 }) const projectId =
process.env.GOOGLE_CLOUD_PROJECT as string const dataset =
process.env.BIGQUERY_TRAINING_DATASET as string
```

AELP2/apps/dashboard/src/app/api/module/[slug]/start/route.ts

```
import { NextRequest, NextResponse } from 'next/server' import { BigQuery } from
'@google-cloud/bigquery' import crypto from 'crypto' export async function
POST(req: NextRequest, { params }: { params: { slug: string } }) { try { const {
slug } = params const body = await req.json().catch(()=>({})) as any const consent
= !!body?.consent const pageUrl = String(body?.page_url||'')
```

AELP2/apps/dashboard/src/app/api/module/[slug]/status/route.ts

```
import { NextRequest, NextResponse } from 'next/server' import {
createBigQueryClient } from '../../../../../lib/bigquery-client' export async
function GET(req: NextRequest, { params }: { params: { slug: string } }) { try {
const url = new URL(req.url) const runId =
String(url.searchParams.get('run_id')||'') if (!runId) return NextResponse.json({
status: 'error', error_code: 'missing_run_id' }, { status: 400 }) const projectId =
process.env.GOOGLE_CLOUD_PROJECT as string const dataset =
process.env.BIGQUERY_TRAINING_DATASET as string
```

AELP2/apps/dashboard/src/app/api/ops/flows/route.ts

```
import { NextResponse } from 'next/server' import { BigQuery } from
'@google-cloud/bigquery' import { getDatasetFromCookie } from
'../../../../lib/dataset' export async function GET() { try { const projectId =
process.env.GOOGLE_CLOUD_PROJECT as string const { dataset } =
getDatasetFromCookie() const bq = new BigQuery({ projectId }) const sql = `
```

AELP2/apps/dashboard/src/app/api/planner/assets/briefs/route.ts

```
import { NextResponse } from 'next/server' export const dynamic = 'force-dynamic'
import { promises as fs } from 'fs' import path from 'path' export async function
GET() { const base = process.env.AELP_REPORTS_DIR || path.resolve(process.cwd(),
'AELP2', 'reports') const p = path.join(base, 'asset_briefs.json') try { const j =
```

```
JSON.parse(await fs.readFile(p, 'utf-8'))
```

AELP2/apps/dashboard/src/app/api/planner/forecasts/route.ts

```
import { NextResponse } from 'next/server' export const dynamic = 'force-dynamic'
import { promises as fs } from 'fs' import path from 'path' export async function
GET() { const reports = process.env.AELP_REPORTS_DIR || path.resolve(process.cwd(),
'AELP2', 'reports') async function readJSON(p: string) { try { return
JSON.parse(await fs.readFile(p, 'utf-8')) } catch { return null } }
```

AELP2/apps/dashboard/src/app/api/planner/packages/route.ts

```
import { NextResponse } from 'next/server' import { promises as fs } from 'fs'
import path from 'path' export const dynamic = 'force-dynamic' type Forecasts = {
budgets: number[] items: Array<{ creative_id: string
```

AELP2/apps/dashboard/src/app/api/planner/rl/route.ts

```
import { NextResponse } from 'next/server' export const dynamic = 'force-dynamic'
import { promises as fs } from 'fs' import path from 'path' export async function
GET() { const reports = process.env.AELP_REPORTS_DIR || path.resolve(process.cwd(),
'AELP2', 'reports') const read = async (name: string) => { try { return
JSON.parse(await fs.readFile(path.join(reports, name), 'utf-8')) } catch { return
null } }
```

AELP2/apps/dashboard/src/app/api/planner/setup/[id]/route.ts

```
import { NextResponse } from 'next/server' import { promises as fs } from 'fs'
import path from 'path' export const dynamic = 'force-dynamic' export async
function GET(_: Request, { params }: { params: { id: string } }) { const id =
params.id const base = process.env.AELP_REPORTS_DIR || path.resolve(process.cwd(),
'AELP2', 'reports') // Load placement hints if available from briefs/rl pack
```

AELP2/apps/dashboard/src/app/api/planner/vendor-scores/route.ts

```
import { NextResponse } from 'next/server' export const dynamic = 'force-dynamic'
import { promises as fs } from 'fs' import path from 'path' export async function
GET() { const base = process.env.AELP_REPORTS_DIR || path.resolve(process.cwd(),
'AELP2', 'reports') const p = path.join(base, 'vendor_scores.json') try { const j =
JSON.parse(await fs.readFile(p, 'utf-8'))
```

AELP2/apps/dashboard/src/app/api/queue/processor/route.ts

```
import { NextRequest, NextResponse } from 'next/server' import { createClient }
from 'redis' import { BigQuery } from '@google-cloud/bigquery' // Redis client for
queue management const redis = createClient({ url: process.env.REDIS_URL ||
'redis://localhost:6379', }) redis.on('error', (err) => console.error('Redis Client
Error', err))
```

AELP2/apps/dashboard/src/app/api/reports/executive/route.ts

```
import { NextRequest, NextResponse } from 'next/server' import {
createBigQueryClient } from '../../../../lib/bigquery-client' import { cookies }
from 'next/headers' import { DATASET_COOKIE, SANDBOX_DATASET, PROD_DATASET } from
'../../../../lib/dataset' export async function GET(req: NextRequest) { try { const
searchParams = req.nextUrl.searchParams const startDate =
searchParams.get('startDate') || new Date(Date.now() - 28 * 24 * 60 * 60 *
1000).toISOString().split('T')[0] const endDate = searchParams.get('endDate') ||
new Date().toISOString().split('T')[0]
```

AELP2/apps/dashboard/src/app/api/research/channels/route.ts

```
import { NextRequest, NextResponse } from 'next/server' import {
createBigQueryClient } from '../../../../lib/bigquery-client' import {
getDatasetFromCookie } from '../../../../lib/dataset' import { BigQuery } from
'@google-cloud/bigquery' export const dynamic = 'force-dynamic' export async
function GET(req: NextRequest) { try { const projectId =
process.env.GOOGLE_CLOUD_PROJECT as string
```

AELP2/apps/dashboard/src/app/api/scenarios/model/route.ts

```
import { NextRequest, NextResponse } from 'next/server' import { BigQuery } from
'@google-cloud/bigquery' import { createBigQueryClient } from
'../../../../lib/bigquery-client' import { cookies } from 'next/headers' import {
DATASET_COOKIE, SANDBOX_DATASET, PROD_DATASET } from '../../../../lib/dataset'
interface ScenarioParams { type: 'budget' | 'bidding' | 'audience' | 'creative' |
'channel' baselineMetrics?: any changes: {
```

AELP2/apps/dashboard/src/app/api/training/start/route.ts

```
import { NextRequest, NextResponse } from 'next/server' import { spawn } from
'child_process' import path from 'path' export async function POST(req:
NextRequest) { try { const body = await req.json() const { modelType = 'rl',
episodes = 1000,
```

AELP2/apps/dashboard/src/app/approvals/page.tsx

```
import React from 'react' import { createBigQueryClient } from
'../../lib/bigquery-client' import { fmtWhen } from '../../lib/utils' export const
dynamic = 'force-dynamic' async function fetchQueue() { const projectId =
process.env.GOOGLE_CLOUD_PROJECT as string const dataset =
process.env.BIGQUERY_TRAINING_DATASET as string const bq =
createBigQueryClient(projectId)
```

AELP2/apps/dashboard/src/app/auctions-monitor/page.tsx

```
import React from 'react' import { BigQuery } from '@google-cloud/bigquery' import
{ TimeSeriesChart } from '../../components/TimeSeriesChart' import { cookies } from
'next/headers' import { DATASET_COOKIE, SANDBOX_DATASET, PROD_DATASET } from
'../../lib/dataset' import { Input } from '../../components/ui/input' import {
Button } from '../../components/ui/button' type MinuteRow = { minute: string,
auctions: number, wins: number, win_rate: number, avg_bid: number, avg_price_paid:
number }
```

AELP2/apps/dashboard/src/app/audiences/page.tsx

```
import React from 'react' import { createBigQueryClient } from
'../../lib/bigquery-client' import { cookies } from 'next/headers' import {
DATASET_COOKIE, PROD_DATASET, SANDBOX_DATASET } from '../../lib/dataset' export
const dynamic = 'force-dynamic' async function fetchSegments() { const projectId =
process.env.GOOGLE_CLOUD_PROJECT as string const mode =
cookies().get(DATASET_COOKIE)?.value === 'prod' ? 'prod' : 'sandbox'
```

AELP2/apps/dashboard/src/app/backstage/page.tsx

```
import React from 'react' export const dynamic = 'force-dynamic' import {
absoluteUrl } from '../lib/url' async function fetchFreshness() { const r = await
fetch(absoluteUrl('/api/bq/freshness'), { cache: 'no-store' }) const j = await
```

```
r.json().catch(()=>({ rows: [] })) return j.rows || [] }
```

AELP2/apps/dashboard/src/app/canvas/page.tsx

```
"use client" import React from 'react' import dynamic from 'next/dynamic' import {
Responsive as RGL, WidthProvider } from 'react-grid-layout' import Card from
'../../components/Card' import MetricTile from '../../components/MetricTile' import
{ TimeSeriesChart } from '../../components/TimeSeriesChart' import { BarChart, Bar,
XAxis, YAxis, Tooltip, ResponsiveContainer, CartesianGrid } from 'recharts' import
ChatViz, { VizSpec } from '../../components/ChatViz'
```

AELP2/apps/dashboard/src/app/channels/page.tsx

```
import React from 'react' export const dynamic = 'force-dynamic' import {
absoluteUrl } from '../lib/url' async function fetchCandidates() { const r = await
fetch(absoluteUrl('/api/research/channels?status=new'), { cache: 'no-store' })
const j = await r.json().catch(()=>({ rows: [] })) return j.rows || [] }
```

AELP2/apps/dashboard/src/app/control/page.tsx

```
"use client" import React, { useState } from 'react' async function post(url:
string): Promise<any> { const r = await fetch(url, { method: 'POST' }) try { return
await r.json() } catch { return { ok: false, error: `HTTP ${r.status}` } } } export
default function ControlPage() { const [kpiIds, setKpiIds] = useState('6453292723')
```

AELP2/apps/dashboard/src/app/creative-center/CreativeCenterClient.tsx

```
"use client" import React, { useState } from 'react' import { fmtUSD, fmtInt,
fmtFloat, fmtPct } from '../../lib/utils' type Row = { date: string campaign_id:
string campaign_name: string ad_group_id: string ad_group_name: string
```

AELP2/apps/dashboard/src/app/creative-center/EnhancedCreativeCenter.tsx

```
"use client" import React, { useState } from 'react' interface GeneratedContent {
type: string generated: any timestamp: string } interface MultiModalHubProps {
```

AELP2/apps/dashboard/src/app/creative-center/page.tsx

```
import React from 'react' import { createBigQueryClient } from
'../../lib/bigquery-client' import { cookies } from 'next/headers' import {
DATASET_COOKIE, SANDBOX_DATASET, PROD_DATASET } from '../../lib/dataset' import {
fmtWhen } from '../../lib/utils' import CreativeCenterClient from
'./CreativeCenterClient' import MultiModalHub from './EnhancedCreativeCenter'
export const dynamic = 'force-dynamic'
```

AELP2/apps/dashboard/src/app/creative-studio/create/page.tsx

```
"use client" import React, { useState } from 'react' export default function
CreateAd() { const [campaignId, setCampaignId] = useState('') const [adGroupId,
setAdGroupId] = useState('') const [finalUrl, setFinalUrl] =
useState('https://buy.aura.com') const [headlines, setHeadlines] =
useState<string>('Aura® Digital Security\nIdentity Theft Protection\n24/7 Customer
Support') const [descriptions, setDescriptions] = useState<string>('Protect your
digital life today.\n#1 rated identity protection.') const [msg, setMsg] =
useState<string>('')
```

AELP2/apps/dashboard/src/app/creative-studio/preview/page.tsx

```
"use client" import React, { useEffect, useState } from 'react' export default
function CreativePreview() { const [data, setData] = useState<any>(null) const
[error, setError] = useState<string>('') useEffect(()=>{ const u = new
URL(window.location.href) const ad_id = u.searchParams.get('ad_id') const
campaign_id = u.searchParams.get('campaign_id')
```

AELP2/apps/dashboard/src/app/exec/ExecClient.tsx

```
"use client" import React, { useState } from 'react' import { TimeSeriesChart }
from '../../components/TimeSeriesChart' import { BanditProposalsTable } from
'../../components/BanditProposalsTable' import { Calendar, Download, RefreshCw }
from 'lucide-react' import dayjs from 'dayjs' interface ExecClientProps {
initialData: any kpiSeries: any[]
```

AELP2/apps/dashboard/src/app/exec/page.tsx

```
import React from 'react' import { TimeSeriesChart } from
'../../components/TimeSeriesChart' import { BanditProposalsTable } from
'../../components/BanditProposalsTable' import ExecClient from './ExecClient'
import dayjs from 'dayjs' import { cookies } from 'next/headers' import {
DATASET_COOKIE, SANDBOX_DATASET, PROD_DATASET } from '../../lib/dataset' import {
fmtWhen } from '../../lib/utils' import { createBigQueryClient } from
'../../lib/bigquery-client' import MetricTile from '../../components/MetricTile'
```

AELP2/apps/dashboard/src/app/experiments/page.tsx

```
import React from 'react' export const dynamic = 'force-dynamic' import {
absoluteUrl } from '../lib/url' async function fetchTests() { const r = await
fetch(absoluteUrl('/api/bq/lp/tests'), { cache: 'no-store' }) const j = await
r.json().catch(()=>({ rows: [] })) return j.rows || [] }
```

AELP2/apps/dashboard/src/app/finance/ScenarioModeler.tsx

```
"use client" import React, { useState } from 'react' import { TrendingUp,
TrendingDown, AlertCircle, CheckCircle, DollarSign, Target, Users, Palette, Share2
} from 'lucide-react' interface ScenarioResult { scenario: { type: string changes:
any timeHorizon: number baseline: any
```

AELP2/apps/dashboard/src/app/finance/page.tsx

```
import React from 'react' import { BigQuery } from '@google-cloud/bigquery' import
{ cookies } from 'next/headers' import { DATASET_COOKIE, SANDBOX_DATASET,
PROD_DATASET } from '../../lib/dataset' import Card from '../../components/Card'
import MMMSlider from '../../components/MMMSlider' import PerChannelMMM from
'../../components/PerChannelMMM' import ValueUploads from
'../../components/ValueUploads' import ScenarioModeler from './ScenarioModeler'
```

AELP2/apps/dashboard/src/app/growth-lab/page.tsx

```
import React from 'react' import { BigQuery } from '@google-cloud/bigquery' import
Card from '../../components/Card' import { fmtWhen } from '../../lib/utils' import
{ cookies } from 'next/headers' import { DATASET_COOKIE, SANDBOX_DATASET,
PROD_DATASET } from '../../lib/dataset' async function fetchMMM() { const projectId
= process.env.GOOGLE_CLOUD_PROJECT as string const mode =
cookies().get(DATASET_COOKIE)?.value === 'prod' ? 'prod' : 'sandbox'
```

AELP2/apps/dashboard/src/app/journeys/page.tsx

```
import React from 'react' import { BigQuery } from '@google-cloud/bigquery' import
{ cookies } from 'next/headers' import { DATASET_COOKIE, SANDBOX_DATASET,
PROD_DATASET } from '../../lib/dataset' import { fmtWhen } from '../../lib/utils'
import Card from '../../components/Card' import JourneysSankey from
'../../components/JourneysSankey' async function fetchGa4() { const projectId =
process.env.GOOGLE_CLOUD_PROJECT as string
```

AELP2/apps/dashboard/src/app/landing/page.tsx

```
import React from 'react' import { createBigQueryClient } from
'../../lib/bigquery-client' import { cookies } from 'next/headers' import {
DATASET_COOKIE, PROD_DATASET, SANDBOX_DATASET } from '../../lib/dataset' export
const dynamic = 'force-dynamic' async function fetchLpTests() { const projectId =
process.env.GOOGLE_CLOUD_PROJECT as string const mode =
cookies().get(DATASET_COOKIE)?.value === 'prod' ? 'prod' : 'sandbox'
```

AELP2/apps/dashboard/src/app/layout.tsx

```
import './globals.css' import React from 'react' import { DatasetSwitcher } from
'../components/DatasetSwitcher' import { KpiSourceSwitcher } from
'../components/KpiSourceSwitcher' import Nav from '../components/Nav' import
Providers from '../components/Providers' import { Inter } from 'next/font/google'
import { AppToaster } from '../components/ui/toaster' const inter = Inter({
subsets: ['latin'], variable: '--font-inter' })
```

AELP2/apps/dashboard/src/app/lib/kpi.ts

```
import { cookies } from 'next/headers' export type KpiSource = 'ads' | 'ga4_all' |
'ga4_google' | 'pacer' export const KPI_COOKIE = 'aelp-kpi-source' export function
getKpiSourceFromCookie(): KpiSource { const c = cookies().get(KPI_COOKIE)?.value as
KpiSource | undefined if (c === 'ga4_all' || c === 'ga4_google' || c === 'ads' || c
=== 'pacer') return c return 'ga4_google' }
```

AELP2/apps/dashboard/src/app/lib/url.ts

```
export function absoluteUrl(path: string) { const base =
process.env.NEXT_PUBLIC_BASE_URL || process.env.INTERNAL_BASE_URL ||
`http://127.0.0.1:${process.env.PORT || 3000}` return
`${base}${path.startsWith('/') ? path : `/${path}`}` }
```

AELP2/apps/dashboard/src/app/onboarding/page.tsx

```
import React from 'react' import { BigQuery } from '@google-cloud/bigquery' import
{ getDatasetFromCookie } from '../../lib/dataset' import fs from 'fs' import path
from 'path' async function credsPresence() { const cfgDir =
path.join(process.cwd(), 'AELP2', 'config') const files = await
fs.promises.readdir(cfgDir).catch(()=>[] as string[]) const present =
files.filter(f => f.includes('credentials') ||
f.includes('google_ads_credentials')).map(f => ({ file: f }))
```

AELP2/apps/dashboard/src/app/ops/chat/page.tsx

```
"use client" import React, { useState } from 'react' import { Button } from
'../../../components/ui/button' import { Input } from
'../../../components/ui/input' import ChatViz, { VizSpec } from
'../../../components/ChatViz' type Msg = { role: 'user'|'assistant', content:
string, viz?: VizSpec } export default function ChatOps() { const [msgs, setMsgs] =
useState<Msg[]>([])
```

AELP2/apps/dashboard/src/app/page.tsx

```
import React from 'react' import { createBigQueryClient } from
'../lib/bigquery-client' import { fmtUSD, fmtFloat } from '../lib/utils' import {
cookies } from 'next/headers' import { DATASET_COOKIE, PROD_DATASET,
SANDBOX_DATASET } from '../lib/dataset' import { getKpiSourceFromCookie } from
'./lib/kpi' import { absoluteUrl } from './lib/url' export const dynamic =
'force-dynamic'
```

AELP2/apps/dashboard/src/app/qs/page.tsx

```
"use client" import React, { useEffect, useState } from 'react' type Row = { date:
string campaign_id: string name: string | null impressions: number clicks: number
spend: number
```

AELP2/apps/dashboard/src/app/rl-insights/page.tsx

```
import React from 'react' import { createBigQueryClient } from
'../../lib/bigquery-client' import { cookies } from 'next/headers' import {
DATASET_COOKIE, PROD_DATASET, SANDBOX_DATASET } from '../../lib/dataset' export
const dynamic = 'force-dynamic' async function fetchPosteriors() { const projectId
= process.env.GOOGLE_CLOUD_PROJECT as string const mode =
cookies().get(DATASET_COOKIE)?.value === 'prod' ? 'prod' : 'sandbox'
```

AELP2/apps/dashboard/src/app/spend-planner/page.tsx

```
import React from 'react' export const dynamic = 'force-dynamic' import {
absoluteUrl } from '../lib/url' async function fetchHeadroom() { const r = await
fetch(absoluteUrl('/api/bq/headroom'), { cache: 'no-store' }) const j = await
r.json().catch(()=>({ rows: [] })) return j.rows || [] }
```

AELP2/apps/dashboard/src/app/training-center/page.tsx

```
import React from 'react' import { TimeSeriesChart } from
'../../components/TimeSeriesChart' import dayjs from 'dayjs' import { BigQuery }
from '@google-cloud/bigquery' import { cookies } from 'next/headers' import {
DATASET_COOKIE, SANDBOX_DATASET, PROD_DATASET } from '../../lib/dataset' async
function fetchEpisodes() { const projectId = process.env.GOOGLE_CLOUD_PROJECT as
string const mode = cookies().get(DATASET_COOKIE)?.value === 'prod' ? 'prod' :
'sandbox'
```

AELP2/apps/dashboard/src/components/BanditProposalsTable.tsx

```
"use client" import React, { useState } from 'react' import { CheckCircle, XCircle,
TrendingUp, TrendingDown, Shuffle } from 'lucide-react' type Proposal = {
timestamp: string platform: string channel: string campaign_id: string ad_id:
string
```

AELP2/apps/dashboard/src/components/Card.tsx

```
import React from 'react' type Props = { title?: string subtitle?: string footer?:
React.ReactNode actions?: React.ReactNode className?: string children?:
React.ReactNode }
```

AELP2/apps/dashboard/src/components/ChatViz.tsx

```
"use client" import React from 'react' import { ResponsiveContainer, LineChart,
Line, XAxis, YAxis, Tooltip, CartesianGrid, BarChart, Bar, Legend } from 'recharts'
export type VizSpec = { type: 'line' | 'bar' title?: string xKey: string yKey:
```

```
string data: any[]
```

AELP2/apps/dashboard/src/components/DatasetSwitcher.tsx

```
"use client" import React, { useEffect, useState } from 'react' import { Database, RefreshCw, Calendar } from 'lucide-react' type FreshnessRow = { table_name: string, max_date: string | null } export function DatasetSwitcher() { const [mode, setMode] = useState<'sandbox'|'prod'>('sandbox') const [fresh, setFresh] = useState<FreshnessRow[]>([]) const [loading, setLoading] = useState(false)
```

AELP2/apps/dashboard/src/components/JourneysSankey.tsx

```
"use client" import React from 'react' import { ResponsiveContainer, Sankey, Tooltip } from 'recharts' export default function JourneysSankey(){ const [data, setData] = React.useState<{nodes:any[], links:any[]}|null>(null) React.useEffect(()=>{ fetch('/api/bq/journeys/sankey').then(r=>r.json()).then(j=>{ if (j.nodes && j.links) setData(j) })
```

AELP2/apps/dashboard/src/components/KpiSourceSwitcher.tsx

```
"use client" import React, { useEffect, useState } from 'react' type Source = 'ads' | 'ga4_all' | 'ga4_google' export function KpiSourceSwitcher() { const [src, setSrc] = useState<Source>('ga4_google') const [loading, setLoading] = useState(false) useEffect(() => { (async()=>{ try { const r = await fetch('/api/kpi-source'); const j = await r.json(); setSrc(j.source as Source) } catch {}
```

AELP2/apps/dashboard/src/components/MMMSlider.tsx

```
"use client" import React from 'react' export default function MMMSlider({ channel = 'google_ads' }: { channel?: string }) { const [grid, setGrid] = React.useState<{ spend:number[], conv:number[] }|null>(null) const [val, setVal] = React.useState<number>(0) const [conv, setConv] = React.useState<number>(0) React.useEffect(()=>{
fetch(`/api/bq/mmm/curves?channel=${encodeURIComponent(channel)}`)
.then(r=>r.json()).then(j=>{
```

AELP2/apps/dashboard/src/components/MetricTile.tsx

```
import React from 'react' import { TrendingUp, TrendingDown, Minus } from 'lucide-react' interface MetricTileProps { label: string value: string hint?: string trend?: 'up' | 'down' | 'neutral' change?: string tone?: 'indigo' | 'emerald' | 'rose' | 'sky'
```

AELP2/apps/dashboard/src/components/Nav.tsx

```
"use client" "use client" import React from 'react' import { usePathname } from 'next/navigation' import { BarChart3, FlaskConical, DollarSign, Palette, GraduationCap,
```

AELP2/apps/dashboard/src/components/PerChannelMMM.tsx

```
"use client" import React from 'react' import MMMSlider from './MMMSlider' export default function PerChannelMMM(){ const [channels, setChannels] = React.useState<string[]>([]) React.useEffect(()=>{
fetch('/api/bq/mmm/channels').then(r=>r.json()).then(j=>
setChannels(j.channels||[])) },[]) if (channels.length === 0) return <div className="text-sm text-slate-600">No channels found.</div>
```

AELP2/apps/dashboard/src/components/Providers.tsx

```
"use client" import React from 'react' import { SessionProvider } from
'next-auth/react' export default function Providers({ children }: { children:
React.ReactNode }){ return <SessionProvider>{children}</SessionProvider> }
```

AELP2/apps/dashboard/src/components/TimeSeriesChart.tsx

```
"use client" import React from 'react' import { LineChart, Line, XAxis, YAxis,
Tooltip, Legend, ResponsiveContainer, CartesianGrid } from 'recharts' export type
Series = { name: string dataKey: string color?: string yAxisId?: string }
```

AELP2/apps/dashboard/src/components/ValueUploads.tsx

```
"use client" import React from 'react' import { Button } from './ui/button' import
{ Input } from './ui/input' import { fmtWhen } from '../lib/utils' export default
function ValueUploads(){ const [channels, setChannels] =
React.useState<string[]>([]) const [form, setForm] = React.useState({ channel:'',
start:'', end:'', multiplier:'1.00', notes:'' }) const [rows, setRows] =
React.useState<any[]>([])
```

AELP2/apps/dashboard/src/components/ui/button.tsx

```
import * as React from 'react' import { Slot } from '@radix-ui/react-slot' import {
cva, type VariantProps } from 'class-variance-authority' import { cn } from
'../../lib/utils' const buttonVariants = cva( 'inline-flex items-center
justify-center whitespace-nowrap rounded-md text-sm font-medium transition-colors
focus-visible:outline-none focus-visible:ring-2 focus-visible:ring-indigo-200
disabled:pointer-events-none disabled:opacity-50', { variants: { variant: {
```

AELP2/apps/dashboard/src/components/ui/input.tsx

```
import * as React from 'react' import { cn } from '../../lib/utils' export
interface InputProps extends React.InputHTMLAttributes<HTMLInputElement> {} const
Input = React.forwardRef<HTMLInputElement, InputProps>(({ className, type, ...props
}, ref) => { return ( <input type={type} className={cn(
```

AELP2/apps/dashboard/src/components/ui/skeleton.tsx

```
import { cn } from '../../lib/utils' export function Skeleton({ className }: {
className?: string }) { return <div className={cn('animate-pulse rounded-md
bg-slate-200', className)} /> }
```

AELP2/apps/dashboard/src/components/ui/toaster.tsx

```
"use client" import { Toaster } from 'sonner' export function AppToaster() { return
<Toaster richColors position="top-right" /> }
```

AELP2/apps/dashboard/src/lib/bigquery-client.ts

```
/** * BigQuery client wrapper that automatically serializes all results * This
ensures no BigQuery objects with {value} structure leak into React */ import {
BigQuery } from '@google-cloud/bigquery' import { serializeBigQueryRows,
debugBigQueryStructure } from './bigquery-serializer' export class
SerializedBigQuery { private bq: BigQuery
```

AELP2/apps/dashboard/src/lib/bigquery-serializer.ts

```
/** * Comprehensive BigQuery object serialization utility * Handles all BigQuery timestamp and nested objects */ export function serializeBigQueryValue(value: any): any { // Handle null/undefined if (value == null) return value; // Handle primitives
```

AELP2/apps/dashboard/src/lib/dataset.ts

```
import { cookies } from 'next/headers' export type DatasetMode = 'sandbox' | 'prod' // Default sandbox to the training dataset if not explicitly provided to avoid hitting a non-existent dataset export const SANDBOX_DATASET = process.env.BIGQUERY_SANDBOX_DATASET || (process.env.BIGQUERY_TRAINING_DATASET as string) export const PROD_DATASET = process.env.BIGQUERY_TRAINING_DATASET as string export const DATASET_COOKIE = 'aelp-dataset' export function getDatasetFromCookie(): { dataset: string, mode: DatasetMode } {
```

AELP2/apps/dashboard/src/lib/modules/registry.ts

```
export type ModuleSlug = 'insight_preview' | 'scam_check' export interface ModuleSpec { slug: ModuleSlug title: string consentText: string } export const MODULES: Record<ModuleSlug, ModuleSpec> = { insight_preview: {
```

AELP2/apps/dashboard/src/lib/utils.ts

```
import { clsx } from 'clsx' import { twMerge } from 'tailwind-merge' export function cn(...inputs: any[]) { return twMerge(clsx(inputs)) } export function toPlain(val: any): any { if (val == null) return val // BigQuery timestamp/datetime often comes as object with { value }
```

AELP2/core/__init__.py

```
"""Core GAELP package (scaffold). This directory will gradually host the unified modules for agents, env/simulator, orchestration, data connectors, intelligence, safety, and monitoring. """
```

AELP2/core/agents/__init__.py

```
"""Agents and policies (scaffold). Contains RL policies (e.g., per-platform bid policies) and bandit policies for allocation (platform/channel/segment) and creative selection. """
```

AELP2/core/agents/subagent_orchestrator.py

```
""" Subagent Orchestrator (Skeleton) Coordinates parallel subagents in a safe, observable way. Intended to be called periodically from the main training loop when enabled via env flags. Flags: - AELP2_SUBAGENTS_ENABLE=1 - AELP2_SUBAGENTS_LIST="creative,budget" (comma-separated) - AELP2_SUBAGENTS_SHADOW=1 (default)
```

AELP2/core/data/google_adapter.py

```
""" Google Ads Platform Adapter for AELP2. Production-ready implementation for Google Ads API integration in shadow mode operation. Transforms Google Ads metrics to unified KPIs and executes normalized actions. """ import os import logging
```

AELP2/core/data/linkedin_adapter.py

```
""" LinkedIn Ads Adapter Stub for AELP2 (shadow-only). """ from datetime import datetime from typing import Any, Dict, Optional from .platform_adapter import PlatformAdapter, UnifiedKPIs, PlatformError, NormalizedAction
```

AELP2/core/data/meta_adapter.py

""" Meta (Facebook) Ads Adapter for AELP2. Full implementation with real API integration for Facebook/Meta Ads. """ import os import json from datetime import datetime, timedelta from typing import Any, Dict, List, Optional, Tuple

AELP2/core/data/platform_adapter.py

""" Production-Grade Platform Adapter Base Classes. Provides normalized action and KPI interfaces for advertising platforms in shadow mode operation for AELP2. """ import os import logging from abc import ABC, abstractmethod

AELP2/core/data/reference_builder.py

""" Calibration Reference Builder Builds a calibration reference JSON (win_rate distribution) from BigQuery Ads views. Requirements: - Env: GOOGLE_CLOUD_PROJECT, BIGQUERY_TRAINING_DATASET - Table: `${BIGQUERY_TRAINING_DATASET}.ads_campaign_performance` with column `impression_share` Output:

AELP2/core/data/tiktok_adapter.py

""" TikTok Ads Adapter Stub for AELP2 (shadow-only). """ from datetime import datetime from typing import Any, Dict, Optional from .platform_adapter import PlatformAdapter, UnifiedKPIs, PlatformError, NormalizedAction

AELP2/core/env/__init__.py

""" AELP2 Environment Module Provides simulator wrappers and calibration systems that enforce real auction mechanics: - LegacyEnvAdapter: Wraps legacy environments with real auction enforcement - AdvancedAuctionCalibrator: Sophisticated calibration system - AuctionWrapper: Dynamic signature detection and auction integration - RealAuctionConfig: Configuration for auction enforcement

AELP2/core/env/calibration.py

""" AELP2 Advanced Auction Calibration System This module provides sophisticated calibration that: - Probes auction signatures dynamically - Fits bid multipliers to achieve 10-30% win rate - Validates against recent data if available - NO hardcoded calibration values - Saves calibration to file for reuse - Handles both (bid, context) and (bid, query_value, context) signatures

AELP2/core/env/simulator.py

""" AELP2 Simulator Wrapper with Real Auction Integration This module provides simulation wrappers that FORCE integration with real legacy auction systems. NO fallbacks or simplified mechanics allowed. - Forces 'use_real_auction' flag to True - Probes auction signatures dynamically - Tracks auction metrics from info dict - Enforces real second-price mechanics

AELP2/core/explainability/__init__.py

"""Explainability modules for AELP2 (ported from GAELP)."""

AELP2/core/explainability/bid_explainability_system.py

$(sed 's/.*/+&/' bid_explainability_system.py)

AELP2/core/ingestion/bq_loader.py

""" BigQuery Loader utilities for AELP2. Strict behavior: - No hardcoded project/dataset; read from env. - Fail-fast with clear errors; no dummy data. """ import os import logging

AELP2/core/intelligence/reward_attribution.py

""" Production-Grade Reward Attribution System for AELP2 This module provides a comprehensive reward attribution wrapper that integrates with the existing AttributionEngine to handle multi-touch attribution with delayed rewards (3-14 day windows). Key Features: - NO HARDCODED values - all parameters from environment configuration - Multi-touch attribution with configurable windows (3-14 days) - Delayed reward credit distribution across touchpoint timeframes

AELP2/core/monitoring/bq_writer.py

""" Production-grade BigQuery monitoring and telemetry writer for AELP2. This module provides robust, fault-tolerant BigQuery integration for training telemetry, safety events, and A/B test results with proper error handling, retry logic, and batch writes. """ import os import json

AELP2/core/monitoring/convergence_monitoring_integration_demo.py

$(sed 's/.*/+&/' convergence_monitoring_integration_demo.py)

AELP2/core/monitoring/drift_monitor.py

""" Drift Monitor utilities for AELP2. Checks distributional drift between RL win_rate and Ads impression_share and optionally triggers recalibration or logs safety events. Env controls (optional): - AELP2_DRIFT_MONITOR_ENABLE=1 - AELP2_DRIFT_MONITOR_DAYS=14 - AELP2_DRIFT_MONITOR_MAX_KS=0.45

AELP2/core/monitoring/example_usage.py

#!/usr/bin/env python3 """ Example usage of BigQueryWriter for AELP2 monitoring. This script demonstrates how to integrate BigQueryWriter into your AELP2 training pipeline for real-time monitoring and analysis. """ import os import time

AELP2/core/monitoring/gaelp_success_criteria_monitor.py

$(sed 's/.*/+&/' gaelp_success_criteria_monitor.py)

AELP2/core/optimization/__init__.py

"""Optimization modules for AELP2 (ported from GAELP)."""

AELP2/core/optimization/audience_bandit_service.py

#!/usr/bin/env python3 """ Audience Bandit Service (shadow): Thompson Sampling over audience segments. Arms: segments from `<project>.<dataset>.segment_scores_daily` with proxy successes based on positive uplift score. Dry-run uses synthetic segments. Writes decisions to `bandit_decisions` (platform='audience', channel='segments'). """ import os import json

AELP2/core/optimization/bandit_orchestrator.py

#!/usr/bin/env python3 """ Bandit Orchestrator (shadow + HITL) Reads recent `bandit_decisions` and translates them into executable creative change proposals (enable/pause/adjust exploration split) under global guardrails: - Exploration

budget cap: default 10% (env `AELP2_EXPLORATION_PCT`) - CAC guardrail: default 200 (env `AELP2_CAC_CAP`) - Shadow-only by default; logs proposals to BigQuery `bandit_change_proposals`.

AELP2/core/optimization/bandit_service.py

#!/usr/bin/env python3 """ Bandit v1 (Thompson Sampling, shadow-only) for Creative Selection on Google Ads. Reads creative arms (ads) for a target campaign from BigQuery `ads_ad_performance` over a lookback window, constructs Beta posteriors using clicks/impressions (CTR proxy), draws Thompson samples, selects the best arm, and logs the decision to `<project>.<dataset>.bandit_decisions` (creating it if missing). Notes:

AELP2/core/optimization/budget_optimizer.py

$(sed 's/.*/+&/' budget_optimizer.py)

AELP2/core/optimization/budget_orchestrator.py

#!/usr/bin/env python3 """ Budget Orchestrator (shadow): Reads the latest MMM allocation for channel 'google_ads', compares to recent spend, and issues shadow canary budget proposals for top campaigns by recent spend using the existing apply_google_canary.py (shadow mode). Caps are enforced by that script. Usage: python -m AELP2.core.optimization.budget_orchestrator --days 14 --top_n 1

AELP2/core/optimization/pmax_bandit_service.py

#!/usr/bin/env python3 """ PMax Bandit Service (shadow): logs decisions for Performance Max campaigns. Uses ads_campaign_performance filtered by advertising_channel_sub_type='PERFORMANCE_MAX'. Falls back to dry-run if table/field missing. """ import os, json, argparse from datetime import datetime from typing import List, Dict, Any import numpy as np

AELP2/core/orchestration/budget_broker.py

#!/usr/bin/env python3 """ Cross-platform Budget Broker (shadow-only, stub): Aggregates proposed allocations (from MMM or portfolio) and produces a `broker_allocations` table keyed by platform and campaign_id with shares. Inputs (if present): - `<project>.<dataset>.portfolio_allocations` (preferred) - `<project>.<dataset>.mmm_allocations` (fallback single-channel)

AELP2/core/orchestration/policy_enforcer.py

#!/usr/bin/env python3 """ Policy Enforcer (platform-agnostic, stub): evaluates high-level policies and records decisions. Writes `<project>.<dataset>.policy_enforcement` with the evaluated rule and target. """ import os from datetime import datetime from google.cloud import bigquery from google.cloud.exceptions import NotFound

AELP2/core/orchestration/production_orchestrator.py

"""Production Orchestrator for AELP2 Training System Full-featured production orchestrator with strict requirements: - NO hardcoded values - all parameters from CLI args and environment variables - Full integration with all AELP2 components (monitoring, safety, intelligence) - Real auction mechanics via legacy environment integration - Comprehensive per-episode metrics tracking - BigQuery telemetry writing with proper error handling - Safety gate evaluation with HITL approval workflows - Reward attribution with multi-touch attribution

AELP2/core/safety/feature_flags.py

```
""" Feature Flags SDK integration (GrowthBook/Unleash/Flagsmith compatible).
Server-side fetch with simple in-memory cache. If no provider configured, falls
back to environment variables and optional local JSON, same as feature_gates.py.
""" from __future__ import annotations import os
```

AELP2/core/safety/feature_gates.py

```
#!/usr/bin/env python3 """ Feature gates / HITL flags wrapper. Reads environment
variables and optional local JSON to decide if an action is allowed. This is a
minimal GrowthBook-like gate without network calls. Env vars: - GATES_ENABLED=1 to
enable gate checks (default: 1) - GATES_FLAGS_JSON=/path/to/flags.json (optional;
{"flag_name": true/false})
```

AELP2/core/safety/hitl.py

```
""" Production-grade Safety Gates and Human-In-The-Loop (HITL) Approval System for
AELP2. This module provides comprehensive safety mechanisms including: -
ConfigurableSafetyGates: Dynamic threshold evaluation from environment variables -
HITLApprovalQueue: Production approval workflow with timeout handling -
PolicyChecker: Content compliance and creative validation - SafetyEventLogger:
Comprehensive audit trail for all safety events All thresholds are configurable via
environment variables with no hardcoded fallbacks.
```

AELP2/examples/safety_demo.py

```
#!/usr/bin/env python3 """ AELP2 Safety System Demonstration This script
demonstrates how to use the AELP2 safety gates and HITL system for validating
actions and ensuring compliance with safety policies. REQUIRED ENVIRONMENT
VARIABLES: - AELP2_MIN_WIN_RATE: Minimum win rate threshold (e.g., "0.15") -
AELP2_MAX_CAC: Maximum Customer Acquisition Cost (e.g., "50.0")
```

AELP2/external/growth-compass-77/src/App.tsx

```
import { Toaster } from "@/components/ui/toaster"; import { Toaster as Sonner }
from "@/components/ui/sonner"; import { TooltipProvider } from
"@/components/ui/tooltip"; import { QueryClient, QueryClientProvider } from
"@tanstack/react-query"; import { BrowserRouter, Routes, Route } from
"react-router-dom"; import { useEffect } from 'react' import { api } from
'./integrations/aelp-api/client' import { CONFIG } from './lib/config' import Index
from "./pages/Index"; import CreativeCenter from "./pages/CreativeCenter";
```

AELP2/external/growth-compass-77/src/components/ads/AdPreview.tsx

```
import React from 'react' function clean(text:string){ try{ // Replace
{KeyWord:...} style with fallback text after ':' text =
text.replace(/\{([^:{}]+):([^}]+)\}/g, (_,key,fb) => fb) // Replace
{LOCATION(...):fallback} -> fallback text =
text.replace(/\{LOCATION\([^)]*\):([^}]+)\}/g, (_,fb) => fb) // Replace
{CUSTOMIZER.[^:]+:fallback} -> fallback text =
text.replace(/\{CUSTOMIZER\.[^:}]+:([^}]+)\}/g, (_,fb) => fb)
```

AELP2/external/growth-compass-77/src/components/dashboard/KPICard.tsx

```
import { ReactNode } from "react"; import { Card } from "@/components/ui/card";
import { Badge } from "@/components/ui/badge"; import { TrendingUp, TrendingDown }
from "lucide-react"; interface KPICardProps { title: string; value: string; change:
number; changeLabel: string;
```

AELP2/external/growth-compass-77/src/components/dashboard/MetricChart.tsx

```
import { Card } from "@/components/ui/card"; import { Button } from
"@/components/ui/button"; import { Badge } from "@/components/ui/badge"; import {
MoreHorizontal, TrendingUp, Eye } from "lucide-react"; import { LineChart, Line,
ResponsiveContainer, XAxis, YAxis, Tooltip, Area, AreaChart } from "recharts";
interface MetricChartProps { title: string; subtitle?: string; data?: Array<{ name:
string; value: number; target?: number }>;
```

AELP2/external/growth-compass-77/src/components/dashboard/Overview.tsx

```
import { DollarSign, Target, TrendingUp, Users, CheckCircle, AlertTriangle, Filter,
Calendar } from "lucide-react";
```

AELP2/external/growth-compass-77/src/components/dashboard/TopAdsByLP.tsx

```
import { useEffect, useMemo, useState } from 'react' import { Card } from
'@/components/ui/card' import { api } from '@/integrations/aelp-api/client' import
{ useCreatives } from '@/hooks/useAelp' type Row = { lp: string impressions: number
clicks: number cost: number
```

AELP2/external/growth-compass-77/src/components/landing-page/ABTestManager.tsx

```
import React, { useState, useEffect } from 'react'; import { Card, CardContent,
CardHeader, CardTitle } from '@/components/ui/card'; import { Button } from
'@/components/ui/button'; import { Input } from '@/components/ui/input'; import {
Label } from '@/components/ui/label'; import { Textarea } from
'@/components/ui/textarea'; import { Select, SelectContent, SelectItem,
SelectTrigger, SelectValue } from '@/components/ui/select'; import { Badge } from
'@/components/ui/badge'; import { Progress } from '@/components/ui/progress';
import { Tabs, TabsContent, TabsList, TabsTrigger } from '@/components/ui/tabs';
```

AELP2/external/growth-compass-77/src/components/landing-page/AICampaignManager.tsx

```
import React, { useState, useEffect } from 'react'; import { Card, CardContent,
CardHeader, CardTitle } from '@/components/ui/card'; import { Button } from
'@/components/ui/button'; import { Input } from '@/components/ui/input'; import {
Label } from '@/components/ui/label'; import { Textarea } from
'@/components/ui/textarea'; import { Select, SelectContent, SelectItem,
SelectTrigger, SelectValue } from '@/components/ui/select'; import { Badge } from
'@/components/ui/badge'; import { Progress } from '@/components/ui/progress';
import { Tabs, TabsContent, TabsList, TabsTrigger } from '@/components/ui/tabs';
```

AELP2/external/growth-compass-77/src/components/landing-page/APIIntegrationHub.tsx

```
import React, { useState, useEffect } from 'react'; import { Card, CardContent,
CardHeader, CardTitle } from '@/components/ui/card'; import { Button } from
'@/components/ui/button'; import { Input } from '@/components/ui/input'; import {
Label } from '@/components/ui/label'; import { Textarea } from
'@/components/ui/textarea'; import { Switch } from '@/components/ui/switch'; import
{ Badge } from '@/components/ui/badge'; import { Tabs, TabsContent, TabsList,
TabsTrigger } from '@/components/ui/tabs'; import {
```

AELP2/external/growth-compass-77/src/components/landing-page/DynamicPageBuilder.tsx

```
import React, { useState, useCallback } from 'react'; import { Card, CardContent,
CardHeader, CardTitle } from '@/components/ui/card'; import { Button } from
'@/components/ui/button'; import { Input } from '@/components/ui/input'; import {
```

```
Label } from '@/components/ui/label'; import { Textarea } from
'@/components/ui/textarea'; import { Select, SelectContent, SelectItem,
SelectTrigger, SelectValue } from '@/components/ui/select'; import { Tabs,
TabsContent, TabsList, TabsTrigger } from '@/components/ui/tabs'; import { Badge }
from '@/components/ui/badge'; import { Plus, Trash2, Move, Settings, Eye, Save,
Rocket } from 'lucide-react';
```

AELP2/external/growth-compass-77/src/components/layout/AppSidebar.tsx

```
import { NavLink, useLocation } from "react-router-dom"; import { BarChart3, Brain,
Palette, FileText, DollarSign, Radio, Shield, Target,
```

AELP2/external/growth-compass-77/src/components/layout/DashboardHeader.tsx

```
import { Bell, Search, User, ChevronDown, Activity, Database } from "lucide-react";
import { Button } from "@/components/ui/button"; import { Input } from
"@/components/ui/input"; import { Badge } from "@/components/ui/badge"; import {
SidebarTrigger } from "@/components/ui/sidebar"; import { DropdownMenu,
DropdownMenuContent, DropdownMenuItem, DropdownMenuTrigger
```

AELP2/external/growth-compass-77/src/components/layout/DashboardLayout.tsx

```
import { ReactNode } from "react"; import { SidebarProvider } from
"@/components/ui/sidebar"; import { AppSidebar } from
"@/components/layout/AppSidebar"; import { DashboardHeader } from
"@/components/layout/DashboardHeader"; interface DashboardLayoutProps { children:
ReactNode; } export function DashboardLayout({ children }: DashboardLayoutProps) {
```

AELP2/external/growth-compass-77/src/components/ui/accordion.tsx

```
import * as React from "react"; import * as AccordionPrimitive from
"@radix-ui/react-accordion"; import { ChevronDown } from "lucide-react"; import {
cn } from "@/lib/utils"; const Accordion = AccordionPrimitive.Root; const
AccordionItem = React.forwardRef< React.ElementRef<typeof AccordionPrimitive.Item>,
```

AELP2/external/growth-compass-77/src/components/ui/alert-dialog.tsx

```
import * as React from "react"; import * as AlertDialogPrimitive from
"@radix-ui/react-alert-dialog"; import { cn } from "@/lib/utils"; import {
buttonVariants } from "@/components/ui/button"; const AlertDialog =
AlertDialogPrimitive.Root; const AlertDialogTrigger = AlertDialogPrimitive.Trigger;
```

AELP2/external/growth-compass-77/src/components/ui/alert.tsx

```
import * as React from "react"; import { cva, type VariantProps } from
"class-variance-authority"; import { cn } from "@/lib/utils"; const alertVariants =
cva( "relative w-full rounded-lg border p-4 [&>svg~*]:pl-7
[&>svg+div]:translate-y-[-3px] [&>svg]:absolute [&>svg]:left-4 [&>svg]:top-4
[&>svg]:text-foreground", { variants: { variant: {
```

AELP2/external/growth-compass-77/src/components/ui/aspect-ratio.tsx

```
import * as AspectRatioPrimitive from "@radix-ui/react-aspect-ratio"; const
AspectRatio = AspectRatioPrimitive.Root; export { AspectRatio };
```

AELP2/external/growth-compass-77/src/components/ui/avatar.tsx

```
import * as React from "react"; import * as AvatarPrimitive from
"@radix-ui/react-avatar"; import { cn } from "@/lib/utils"; const Avatar =
```

```
React.forwardRef< React.ElementRef<typeof AvatarPrimitive.Root>,
React.ComponentPropsWithoutRef<typeof AvatarPrimitive.Root> >(({ className,
...props }, ref) => ( <AvatarPrimitive.Root
```

AELP2/external/growth-compass-77/src/components/ui/badge.tsx

```
import * as React from "react"; import { cva, type VariantProps } from
"class-variance-authority"; import { cn } from "@/lib/utils"; const badgeVariants =
cva( "inline-flex items-center rounded-full border px-2.5 py-0.5 text-xs
font-semibold transition-colors focus:outline-none focus:ring-2 focus:ring-ring
focus:ring-offset-2", { variants: { variant: {
```

AELP2/external/growth-compass-77/src/components/ui/breadcrumb.tsx

```
import * as React from "react"; import { Slot } from "@radix-ui/react-slot"; import
{ ChevronRight, MoreHorizontal } from "lucide-react"; import { cn } from
"@/lib/utils"; const Breadcrumb = React.forwardRef< HTMLElement,
React.ComponentPropsWithoutRef<"nav"> & { separator?: React.ReactNode;
```

AELP2/external/growth-compass-77/src/components/ui/button.tsx

```
import * as React from "react"; import { Slot } from "@radix-ui/react-slot"; import
{ cva, type VariantProps } from "class-variance-authority"; import { cn } from
"@/lib/utils"; const buttonVariants = cva( "inline-flex items-center justify-center
gap-2 whitespace-nowrap rounded-md text-sm font-medium ring-offset-background
transition-colors focus-visible:outline-none focus-visible:ring-2
focus-visible:ring-ring focus-visible:ring-offset-2 disabled:pointer-events-none
disabled:opacity-50 [&_svg]:pointer-events-none [&_svg]:size-4 [&_svg]:shrink-0", {
variants: {
```

AELP2/external/growth-compass-77/src/components/ui/calendar.tsx

```
import * as React from "react"; import { ChevronLeft, ChevronRight } from
"lucide-react"; import { DayPicker } from "react-day-picker"; import { cn } from
"@/lib/utils"; import { buttonVariants } from "@/components/ui/button"; export type
CalendarProps = React.ComponentProps<typeof DayPicker>; function Calendar({
className, classNames, showOutsideDays = true, ...props }: CalendarProps) {
```

AELP2/external/growth-compass-77/src/components/ui/card.tsx

```
import * as React from "react"; import { cn } from "@/lib/utils"; const Card =
React.forwardRef<HTMLDivElement, React.HTMLAttributes<HTMLDivElement>>(({
className, ...props }, ref) => ( <div ref={ref} className={cn("rounded-lg border
bg-card text-card-foreground shadow-sm", className)} {...props} /> ));
Card.displayName = "Card"; const CardHeader = React.forwardRef<HTMLDivElement,
React.HTMLAttributes<HTMLDivElement>>(
```

AELP2/external/growth-compass-77/src/components/ui/carousel.tsx

```
import * as React from "react"; import useEmblaCarousel, { type
UseEmblaCarouselType } from "embla-carousel-react"; import { ArrowLeft, ArrowRight
} from "lucide-react"; import { cn } from "@/lib/utils"; import { Button } from
"@/components/ui/button"; type CarouselApi = UseEmblaCarouselType[1]; type
UseCarouselParameters = Parameters<typeof useEmblaCarousel>; type CarouselOptions =
UseCarouselParameters[0];
```

AELP2/external/growth-compass-77/src/components/ui/chart.tsx

```
import * as React from "react"; import * as RechartsPrimitive from "recharts";
import { cn } from "@/lib/utils"; // Format: { THEME_NAME: CSS_SELECTOR } const
THEMES = { light: "", dark: ".dark" } as const; export type ChartConfig = { [k in
string]: {
```

AELP2/external/growth-compass-77/src/components/ui/checkbox.tsx

```
import * as React from "react"; import * as CheckboxPrimitive from
"@radix-ui/react-checkbox"; import { Check } from "lucide-react"; import { cn }
from "@/lib/utils"; const Checkbox = React.forwardRef< React.ElementRef<typeof
CheckboxPrimitive.Root>, React.ComponentPropsWithoutRef<typeof
CheckboxPrimitive.Root> >(({ className, ...props }, ref) => (
```

AELP2/external/growth-compass-77/src/components/ui/collapsible.tsx

```
import * as CollapsiblePrimitive from "@radix-ui/react-collapsible"; const
Collapsible = CollapsiblePrimitive.Root; const CollapsibleTrigger =
CollapsiblePrimitive.CollapsibleTrigger; const CollapsibleContent =
CollapsiblePrimitive.CollapsibleContent; export { Collapsible, CollapsibleTrigger,
CollapsibleContent };
```

AELP2/external/growth-compass-77/src/components/ui/command.tsx

```
import * as React from "react"; import { type DialogProps } from
"@radix-ui/react-dialog"; import { Command as CommandPrimitive } from "cmdk";
import { Search } from "lucide-react"; import { cn } from "@/lib/utils"; import {
Dialog, DialogContent } from "@/components/ui/dialog"; const Command =
React.forwardRef< React.ElementRef<typeof CommandPrimitive>,
```

AELP2/external/growth-compass-77/src/components/ui/context-menu.tsx

```
import * as React from "react"; import * as ContextMenuPrimitive from
"@radix-ui/react-context-menu"; import { Check, ChevronRight, Circle } from
"lucide-react"; import { cn } from "@/lib/utils"; const ContextMenu =
ContextMenuPrimitive.Root; const ContextMenuTrigger = ContextMenuPrimitive.Trigger;
```

AELP2/external/growth-compass-77/src/components/ui/dialog.tsx

```
import * as React from "react"; import * as DialogPrimitive from
"@radix-ui/react-dialog"; import { X } from "lucide-react"; import { cn } from
"@/lib/utils"; const Dialog = DialogPrimitive.Root; const DialogTrigger =
DialogPrimitive.Trigger;
```

AELP2/external/growth-compass-77/src/components/ui/drawer.tsx

```
import * as React from "react"; import { Drawer as DrawerPrimitive } from "vaul";
import { cn } from "@/lib/utils"; const Drawer = ({ shouldScaleBackground = true,
...props }: React.ComponentProps<typeof DrawerPrimitive.Root>) => (
<DrawerPrimitive.Root shouldScaleBackground={shouldScaleBackground} {...props} />
); Drawer.displayName = "Drawer";
```

AELP2/external/growth-compass-77/src/components/ui/dropdown-menu.tsx

```
import * as React from "react"; import * as DropdownMenuPrimitive from
"@radix-ui/react-dropdown-menu"; import { Check, ChevronRight, Circle } from
"lucide-react"; import { cn } from "@/lib/utils"; const DropdownMenu =
DropdownMenuPrimitive.Root; const DropdownMenuTrigger =
DropdownMenuPrimitive.Trigger;
```

AELP2/external/growth-compass-77/src/components/ui/form.tsx

```
import * as React from "react"; import * as LabelPrimitive from
"@radix-ui/react-label"; import { Slot } from "@radix-ui/react-slot"; import {
Controller, ControllerProps, FieldPath, FieldValues, FormProvider, useFormContext }
from "react-hook-form"; import { cn } from "@/lib/utils"; import { Label } from
"@/components/ui/label"; const Form = FormProvider;
```

AELP2/external/growth-compass-77/src/components/ui/hover-card.tsx

```
import * as React from "react"; import * as HoverCardPrimitive from
"@radix-ui/react-hover-card"; import { cn } from "@/lib/utils"; const HoverCard =
HoverCardPrimitive.Root; const HoverCardTrigger = HoverCardPrimitive.Trigger; const
HoverCardContent = React.forwardRef<
```

AELP2/external/growth-compass-77/src/components/ui/input-otp.tsx

```
import * as React from "react"; import { OTPInput, OTPInputContext } from
"input-otp"; import { Dot } from "lucide-react"; import { cn } from "@/lib/utils";
const InputOTP = React.forwardRef<React.ElementRef<typeof OTPInput>,
React.ComponentPropsWithoutRef<typeof OTPInput>>( ({ className, containerClassName,
...props }, ref) => ( <OTPInput ref={ref}
```

AELP2/external/growth-compass-77/src/components/ui/input.tsx

```
import * as React from "react"; import { cn } from "@/lib/utils"; const Input =
React.forwardRef<HTMLInputElement, React.ComponentProps<"input">>( ({ className,
type, ...props }, ref) => { return ( <input type={type} className={cn(
```

AELP2/external/growth-compass-77/src/components/ui/label.tsx

```
import * as React from "react"; import * as LabelPrimitive from
"@radix-ui/react-label"; import { cva, type VariantProps } from
"class-variance-authority"; import { cn } from "@/lib/utils"; const labelVariants =
cva("text-sm font-medium leading-none peer-disabled:cursor-not-allowed
peer-disabled:opacity-70"); const Label = React.forwardRef< React.ElementRef<typeof
LabelPrimitive.Root>,
```

AELP2/external/growth-compass-77/src/components/ui/menubar.tsx

```
import * as React from "react"; import * as MenubarPrimitive from
"@radix-ui/react-menubar"; import { Check, ChevronRight, Circle } from
"lucide-react"; import { cn } from "@/lib/utils"; const MenubarMenu =
MenubarPrimitive.Menu; const MenubarGroup = MenubarPrimitive.Group;
```

AELP2/external/growth-compass-77/src/components/ui/navigation-menu.tsx

```
import * as React from "react"; import * as NavigationMenuPrimitive from
"@radix-ui/react-navigation-menu"; import { cva } from "class-variance-authority";
import { ChevronDown } from "lucide-react"; import { cn } from "@/lib/utils"; const
NavigationMenu = React.forwardRef< React.ElementRef<typeof
NavigationMenuPrimitive.Root>, React.ComponentPropsWithoutRef<typeof
NavigationMenuPrimitive.Root>
```

AELP2/external/growth-compass-77/src/components/ui/pagination.tsx

```
import * as React from "react"; import { ChevronLeft, ChevronRight, MoreHorizontal
} from "lucide-react"; import { cn } from "@/lib/utils"; import { ButtonProps,
buttonVariants } from "@/components/ui/button"; const Pagination = ({ className,
```

```
...props }: React.ComponentProps<"nav">) => ( <nav role="navigation"
aria-label="pagination"
```

AELP2/external/growth-compass-77/src/components/ui/popover.tsx

```
import * as React from "react"; import * as PopoverPrimitive from
"@radix-ui/react-popover"; import { cn } from "@/lib/utils"; const Popover =
PopoverPrimitive.Root; const PopoverTrigger = PopoverPrimitive.Trigger; const
PopoverContent = React.forwardRef<
```

AELP2/external/growth-compass-77/src/components/ui/progress.tsx

```
import * as React from "react"; import * as ProgressPrimitive from
"@radix-ui/react-progress"; import { cn } from "@/lib/utils"; const Progress =
React.forwardRef< React.ElementRef<typeof ProgressPrimitive.Root>,
React.ComponentPropsWithoutRef<typeof ProgressPrimitive.Root> >(({ className,
value, ...props }, ref) => ( <ProgressPrimitive.Root
```

AELP2/external/growth-compass-77/src/components/ui/radio-group.tsx

```
import * as React from "react"; import * as RadioGroupPrimitive from
"@radix-ui/react-radio-group"; import { Circle } from "lucide-react"; import { cn }
from "@/lib/utils"; const RadioGroup = React.forwardRef< React.ElementRef<typeof
RadioGroupPrimitive.Root>, React.ComponentPropsWithoutRef<typeof
RadioGroupPrimitive.Root> >(({ className, ...props }, ref) => {
```

AELP2/external/growth-compass-77/src/components/ui/resizable.tsx

```
import { GripVertical } from "lucide-react"; import * as ResizablePrimitive from
"react-resizable-panels"; import { cn } from "@/lib/utils"; const
ResizablePanelGroup = ({ className, ...props }: React.ComponentProps<typeof
ResizablePrimitive.PanelGroup>) => ( <ResizablePrimitive.PanelGroup
className={cn("flex h-full w-full data-[panel-group-direction=vertical]:flex-col",
className)} {...props} />
```

AELP2/external/growth-compass-77/src/components/ui/scroll-area.tsx

```
import * as React from "react"; import * as ScrollAreaPrimitive from
"@radix-ui/react-scroll-area"; import { cn } from "@/lib/utils"; const ScrollArea =
React.forwardRef< React.ElementRef<typeof ScrollAreaPrimitive.Root>,
React.ComponentPropsWithoutRef<typeof ScrollAreaPrimitive.Root> >(({ className,
children, ...props }, ref) => ( <ScrollAreaPrimitive.Root ref={ref}
className={cn("relative overflow-hidden", className)} {...props}>
```

AELP2/external/growth-compass-77/src/components/ui/select.tsx

```
import * as React from "react"; import * as SelectPrimitive from
"@radix-ui/react-select"; import { Check, ChevronDown, ChevronUp } from
"lucide-react"; import { cn } from "@/lib/utils"; const Select =
SelectPrimitive.Root; const SelectGroup = SelectPrimitive.Group;
```

AELP2/external/growth-compass-77/src/components/ui/separator.tsx

```
import * as React from "react"; import * as SeparatorPrimitive from
"@radix-ui/react-separator"; import { cn } from "@/lib/utils"; const Separator =
React.forwardRef< React.ElementRef<typeof SeparatorPrimitive.Root>,
React.ComponentPropsWithoutRef<typeof SeparatorPrimitive.Root> >(({ className,
orientation = "horizontal", decorative = true, ...props }, ref) => (
<SeparatorPrimitive.Root
```

AELP2/external/growth-compass-77/src/components/ui/sheet.tsx

```
import * as SheetPrimitive from "@radix-ui/react-dialog"; import { cva, type
VariantProps } from "class-variance-authority"; import { X } from "lucide-react";
import * as React from "react"; import { cn } from "@/lib/utils"; const Sheet =
SheetPrimitive.Root; const SheetTrigger = SheetPrimitive.Trigger;
```

AELP2/external/growth-compass-77/src/components/ui/sidebar.tsx

```
import * as React from "react"; import { Slot } from "@radix-ui/react-slot"; import
{ VariantProps, cva } from "class-variance-authority"; import { PanelLeft } from
"lucide-react"; import { useIsMobile } from "@/hooks/use-mobile"; import { cn }
from "@/lib/utils"; import { Button } from "@/components/ui/button"; import { Input
} from "@/components/ui/input"; import { Separator } from
"@/components/ui/separator";
```

AELP2/external/growth-compass-77/src/components/ui/skeleton.tsx

```
import { cn } from "@/lib/utils"; function Skeleton({ className, ...props }:
React.HTMLAttributes<HTMLDivElement>) { return <div className={cn("animate-pulse
rounded-md bg-muted", className)} {...props} />; } export { Skeleton };
```

AELP2/external/growth-compass-77/src/components/ui/slider.tsx

```
import * as React from "react"; import * as SliderPrimitive from
"@radix-ui/react-slider"; import { cn } from "@/lib/utils"; const Slider =
React.forwardRef< React.ElementRef<typeof SliderPrimitive.Root>,
React.ComponentPropsWithoutRef<typeof SliderPrimitive.Root> >(({ className,
...props }, ref) => ( <SliderPrimitive.Root
```

AELP2/external/growth-compass-77/src/components/ui/sonner.tsx

```
import { useTheme } from "next-themes"; import { Toaster as Sonner, toast } from
"sonner"; type ToasterProps = React.ComponentProps<typeof Sonner>; const Toaster =
({ ...props }: ToasterProps) => { const { theme = "system" } = useTheme(); return (
<Sonner
```

AELP2/external/growth-compass-77/src/components/ui/switch.tsx

```
import * as React from "react"; import * as SwitchPrimitives from
"@radix-ui/react-switch"; import { cn } from "@/lib/utils"; const Switch =
React.forwardRef< React.ElementRef<typeof SwitchPrimitives.Root>,
React.ComponentPropsWithoutRef<typeof SwitchPrimitives.Root> >(({ className,
...props }, ref) => ( <SwitchPrimitives.Root
```

AELP2/external/growth-compass-77/src/components/ui/table.tsx

```
import * as React from "react"; import { cn } from "@/lib/utils"; const Table =
React.forwardRef<HTMLTableElement, React.HTMLAttributes<HTMLTableElement>>( ({
className, ...props }, ref) => ( <div className="relative w-full overflow-auto">
<table ref={ref} className={cn("w-full caption-bottom text-sm", className)}
{...props} /> </div> ),
```

AELP2/external/growth-compass-77/src/components/ui/tabs.tsx

```
import * as React from "react"; import * as TabsPrimitive from
"@radix-ui/react-tabs"; import { cn } from "@/lib/utils"; const Tabs =
TabsPrimitive.Root; const TabsList = React.forwardRef< React.ElementRef<typeof
```

```
TabsPrimitive.List>, React.ComponentPropsWithoutRef<typeof TabsPrimitive.List>
```

AELP2/external/growth-compass-77/src/components/ui/textarea.tsx

```
import * as React from "react"; import { cn } from "@/lib/utils"; export interface
TextareaProps extends React.TextareaHTMLAttributes<HTMLTextAreaElement> {} const
Textarea = React.forwardRef<HTMLTextAreaElement, TextareaProps>(({ className,
...props }, ref) => { return ( <textarea className={cn(
```

AELP2/external/growth-compass-77/src/components/ui/toast.tsx

```
import * as React from "react"; import * as ToastPrimitives from
"@radix-ui/react-toast"; import { cva, type VariantProps } from
"class-variance-authority"; import { X } from "lucide-react"; import { cn } from
"@/lib/utils"; const ToastProvider = ToastPrimitives.Provider; const ToastViewport
= React.forwardRef<
```

AELP2/external/growth-compass-77/src/components/ui/toaster.tsx

```
import { useToast } from "@/hooks/use-toast"; import { Toast, ToastClose,
ToastDescription, ToastProvider, ToastTitle, ToastViewport } from
"@/components/ui/toast"; export function Toaster() { const { toasts } = useToast();
return ( <ToastProvider> {toasts.map(function ({ id, title, description, action,
...props }) { return (
```

AELP2/external/growth-compass-77/src/components/ui/toggle-group.tsx

```
import * as React from "react"; import * as ToggleGroupPrimitive from
"@radix-ui/react-toggle-group"; import { type VariantProps } from
"class-variance-authority"; import { cn } from "@/lib/utils"; import {
toggleVariants } from "@/components/ui/toggle"; const ToggleGroupContext =
React.createContext<VariantProps<typeof toggleVariants>>({ size: "default",
variant: "default",
```

AELP2/external/growth-compass-77/src/components/ui/toggle.tsx

```
import * as React from "react"; import * as TogglePrimitive from
"@radix-ui/react-toggle"; import { cva, type VariantProps } from
"class-variance-authority"; import { cn } from "@/lib/utils"; const toggleVariants
= cva( "inline-flex items-center justify-center rounded-md text-sm font-medium
ring-offset-background transition-colors hover:bg-muted hover:text-muted-foreground
focus-visible:outline-none focus-visible:ring-2 focus-visible:ring-ring
focus-visible:ring-offset-2 disabled:pointer-events-none disabled:opacity-50
data-[state=on]:bg-accent data-[state=on]:text-accent-foreground", { variants: {
```

AELP2/external/growth-compass-77/src/components/ui/tooltip.tsx

```
import * as React from "react"; import * as TooltipPrimitive from
"@radix-ui/react-tooltip"; import { cn } from "@/lib/utils"; const TooltipProvider
= TooltipPrimitive.Provider; const Tooltip = TooltipPrimitive.Root; const
TooltipTrigger = TooltipPrimitive.Trigger;
```

AELP2/external/growth-compass-77/src/components/ui/use-toast.ts

```
import { useToast, toast } from "@/hooks/use-toast"; export { useToast, toast };
```

AELP2/external/growth-compass-77/src/hooks/use-mobile.tsx

```
import * as React from "react"; const MOBILE_BREAKPOINT = 768; export function
useIsMobile() { const [isMobile, setIsMobile] = React.useState<boolean |
undefined>(undefined); React.useEffect(() => { const mql =
window.matchMedia(`(max-width: ${MOBILE_BREAKPOINT - 1}px)`); const onChange = ()
=> {
```

AELP2/external/growth-compass-77/src/hooks/use-toast.ts

```
import * as React from "react"; import type { ToastActionElement, ToastProps } from
"@/components/ui/toast"; const TOAST_LIMIT = 1; const TOAST_REMOVE_DELAY = 1000000;
type ToasterToast = ToastProps & { id: string; title?: React.ReactNode;
```

AELP2/external/growth-compass-77/src/hooks/useAelp.ts

```
import { useQuery } from '@tanstack/react-query' import { api } from
'../integrations/aelp-api/client' export const useDataset = () => useQuery({
queryKey: ['dataset'], queryFn: api.dataset.get }) export const useKpiSummary = ()
=> useQuery({ queryKey: ['kpi','summary'], queryFn: api.kpi.summary, staleTime:
10_000 }) export const useKpiDaily = (days = 28) => useQuery({ queryKey:
['kpi','daily',days], queryFn: () => api.kpi.daily(days), staleTime: 10_000 })
export const useHeadroom = () => useQuery({ queryKey: ['headroom'], queryFn:
api.headroom, staleTime: 10_000 }) export const useCreatives = () => useQuery({
queryKey: ['creatives'], queryFn: api.creatives, staleTime: 15_000 }) export const
useChannelAttribution = () => useQuery({ queryKey: ['channel-attribution'],
queryFn: api.channelAttribution }) export const useGa4Channels = () => useQuery({
queryKey: ['ga4-channels'], queryFn: api.ga4Channels })
```

AELP2/external/growth-compass-77/src/integrations/aelp-api/client.ts

```
import type { DatasetInfo, KpiSummary, HeadroomResult, TableResult,
CreativePerfRow, ChannelAttributionRow, Ga4ChannelRow, MmmAllocationRow,
AbExperimentRow,
```

AELP2/external/growth-compass-77/src/integrations/aelp-api/types.ts

```
export type DatasetMode = 'sandbox' | 'prod' export interface DatasetInfo { mode:
DatasetMode; dataset: string } export interface KpiRow { date: string | Date
conversions: number revenue: number cost: number cac?: number
```

AELP2/external/growth-compass-77/src/integrations/supabase/client.ts

```
// This file is automatically generated. Do not edit it directly. import {
createClient } from '@supabase/supabase-js'; import type { Database } from
'./types'; const SUPABASE_URL = "https://vwzxjscfucsvoxysgmec.supabase.co"; const
SUPABASE_PUBLISHABLE_KEY = "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpc3MiOiJzdXBhYm
FzZSIsInJlZiI6InZ3enhqc2NmdWNzdm94eXNnbWVjIiwicm9sZSI6ImFub24iLCJpYXQiOjE3NTc3MDc1M
DYsImV4cCI6MjA3MzI4MzUwNn0.nhlfLdtFNi80XZMNWYMWEiDczm0HNBxsHpFFqO9bjgc"; // Import
the supabase client like this: // import { supabase } from
"@/integrations/supabase/client";
```

AELP2/external/growth-compass-77/src/integrations/supabase/types.ts

```
export type Json = | string | number | boolean | null | { [key: string]: Json |
undefined } | Json[] export type Database = { // Allows to automatically
instantiate createClient with right options
```

AELP2/external/growth-compass-77/src/lib/config.ts

```
export const CONFIG = { API_BASE: (import.meta.env.VITE_API_BASE_URL as string) ||
'', DATASET_MODE: ((import.meta.env.VITE_DATASET_MODE as string) === 'prod' ?
'prod' : 'sandbox') as 'sandbox' | 'prod', }
```

AELP2/external/growth-compass-77/src/lib/utils.ts

```
import { clsx, type ClassValue } from "clsx"; import { twMerge } from
"tailwind-merge"; export function cn(...inputs: ClassValue[]) { return
twMerge(clsx(inputs)); }
```

AELP2/external/growth-compass-77/src/main.tsx

```
import { createRoot } from "react-dom/client"; import App from "./App.tsx"; import
"./index.css"; createRoot(document.getElementById("root")!).render(<App />);
```

AELP2/external/growth-compass-77/src/pages/Affiliates.tsx

```
import React, { useMemo } from 'react' import DashboardLayout from
'@/components/layout/DashboardLayout' import { useQuery } from
'@tanstack/react-query' import { Card, CardContent, CardHeader, CardTitle } from
'@/components/ui/card' import { Input } from '@/components/ui/input' import {
Button } from '@/components/ui/button' function useTopPartners(days: number,
minPayout: number, limit: number) { return useQuery({ queryKey:
['impact-top-partners', days, minPayout, limit],
```

AELP2/external/growth-compass-77/src/pages/Approvals.tsx

```
import { DashboardLayout } from "@/components/layout/DashboardLayout"; import {
Card } from "@/components/ui/card"; import { Button } from
"@/components/ui/button"; import { Badge } from "@/components/ui/badge"; import {
Clock, CheckCircle, XCircle, AlertTriangle, Users,
```

AELP2/external/growth-compass-77/src/pages/AuctionsMonitor.tsx

```
import { DashboardLayout } from "@/components/layout/DashboardLayout"; import {
Card } from "@/components/ui/card"; import { Badge } from "@/components/ui/badge";
import { Button } from "@/components/ui/button"; import { Tabs, TabsContent,
TabsList, TabsTrigger } from "@/components/ui/tabs"; import { Zap, TrendingUp, Eye,
Target,
```

AELP2/external/growth-compass-77/src/pages/Audiences.tsx

```
import { DashboardLayout } from "@/components/layout/DashboardLayout"; import {
Card } from "@/components/ui/card"; import { Button } from
"@/components/ui/button"; import { toast } from "sonner"; export default function
Audiences(){ const sync = async()=>{ try { const base = (import.meta as
any).env.VITE_API_BASE_URL || '' const r = await
fetch(`${base}/api/control/audience/sync`, { method:'POST', credentials:'include'
})
```

AELP2/external/growth-compass-77/src/pages/Backstage.tsx

```
import { DashboardLayout } from "@/components/layout/DashboardLayout"; import {
Card } from "@/components/ui/card"; import { Badge } from "@/components/ui/badge";
import { Button } from "@/components/ui/button"; import { useEffect, useState }
from "react"; export default function Backstage(){ const [fresh, setFresh] =
useState<any>({ rows: [] }) const [conn, setConn] = useState<any>({ checks: {} })
const [ds, setDs] = useState<any>({ mode: 'sandbox', dataset: '' })
```

AELP2/external/growth-compass-77/src/pages/Canvas.tsx

```
import { DashboardLayout } from "@/components/layout/DashboardLayout"; import {
Card } from "@/components/ui/card"; import { Button } from
"@/components/ui/button"; import { useEffect, useState } from "react"; import {
toast } from "sonner"; export default function Canvas(){ const [items, setItems] =
useState<any[]>([]) const load = async ()=>{ try {
```

AELP2/external/growth-compass-77/src/pages/Channels.tsx

```
import { DashboardLayout } from "@/components/layout/DashboardLayout"; import {
Card } from "@/components/ui/card"; import { Button } from
"@/components/ui/button"; import { Badge } from "@/components/ui/badge"; import {
Radio, Plus, TrendingUp } from "lucide-react"; import { useChannelAttribution,
useGa4Channels, useMmmAllocations } from "@/hooks/useAelp"; import { toast } from
"sonner"; export default function Channels() { const attrib =
useChannelAttribution()
```

AELP2/external/growth-compass-77/src/pages/CreativeCenter.tsx

```
import { DashboardLayout } from "@/components/layout/DashboardLayout"; import {
Card } from "@/components/ui/card"; import { Button } from
"@/components/ui/button"; import { Badge } from "@/components/ui/badge"; import {
Input } from "@/components/ui/input"; import { Textarea } from
"@/components/ui/textarea"; import { Tabs, TabsContent, TabsList, TabsTrigger }
from "@/components/ui/tabs"; import { Palette, Plus,
```

AELP2/external/growth-compass-77/src/pages/CreativePlanner.tsx

```
import React from 'react' import { useQuery } from '@tanstack/react-query' import {
Card, CardContent, CardHeader, CardTitle } from '@/components/ui/card' import {
Button } from '@/components/ui/button' import { DashboardLayout } from
'@/components/layout/DashboardLayout' type BudgetKey = '30000'|'50000' function
usePlanner() { return useQuery({
```

AELP2/external/growth-compass-77/src/pages/ExecutiveDashboard.tsx

```
import React from "react"; import { DashboardLayout } from
"@/components/layout/DashboardLayout"; import { Card } from "@/components/ui/card";
import { Badge } from "@/components/ui/badge"; import { Button } from
"@/components/ui/button"; import { KPICard } from "@/components/dashboard/KPICard";
import { MetricChart } from "@/components/dashboard/MetricChart"; import {
useKpiDaily } from "@/hooks/useAelp"; import { useHeadroom, useGa4Channels,
useDataset } from "@/hooks/useAelp"; import { api } from
"@/integrations/aelp-api/client";
```

AELP2/external/growth-compass-77/src/pages/Experiments.tsx

```
import { DashboardLayout } from "@/components/layout/DashboardLayout"; import {
Card } from "@/components/ui/card"; import { Button } from
"@/components/ui/button"; import { Badge } from "@/components/ui/badge"; import {
Tabs, TabsContent, TabsList, TabsTrigger } from "@/components/ui/tabs"; import {
TestTube, Plus, Play, Pause,
```

AELP2/external/growth-compass-77/src/pages/Finance.tsx

```
import { DashboardLayout } from "@/components/layout/DashboardLayout"; import {
Card } from "@/components/ui/card"; import { Badge } from "@/components/ui/badge";
import { Button } from "@/components/ui/button"; import { Tabs, TabsContent,
```

```
TabsList, TabsTrigger } from "@/components/ui/tabs"; import { Input } from
"@/components/ui/input"; import { DollarSign, TrendingUp, Users,
```

AELP2/external/growth-compass-77/src/pages/Index.tsx

```
import { DashboardLayout } from "@/components/layout/DashboardLayout"; import {
Overview } from "@/components/dashboard/Overview"; const Index = () => { return (
<DashboardLayout> <Overview /> </DashboardLayout> ); };
```

AELP2/external/growth-compass-77/src/pages/LandingPages.tsx

```
import React, { useState, useEffect } from 'react'; import { DashboardLayout } from
'@/components/layout/DashboardLayout'; import { Card, CardContent, CardHeader,
CardTitle } from '@/components/ui/card'; import { Button } from
'@/components/ui/button'; import { Badge } from '@/components/ui/badge'; import {
Tabs, TabsContent, TabsList, TabsTrigger } from '@/components/ui/tabs'; import {
Plus, Eye, Edit,
```

AELP2/external/growth-compass-77/src/pages/NotFound.tsx

```
import { useLocation } from "react-router-dom"; import { useEffect } from "react";
const NotFound = () => { const location = useLocation(); useEffect(() => {
console.error("404 Error: User attempted to access non-existent route:",
location.pathname); }, [location.pathname]);
```

AELP2/external/growth-compass-77/src/pages/OpsChat.tsx

```
import { DashboardLayout } from "@/components/layout/DashboardLayout"; import {
Card } from "@/components/ui/card"; import { Button } from
"@/components/ui/button"; import { Input } from "@/components/ui/input"; import {
Badge } from "@/components/ui/badge"; import { Avatar, AvatarFallback } from
"@/components/ui/avatar"; import { MessageSquare, Send, BarChart3, TrendingUp, Pin,
Bot, User } from "lucide-react"; import { useState, useEffect } from "react";
import { toast } from "sonner";
```

AELP2/external/growth-compass-77/src/pages/QS.tsx

```
import { DashboardLayout } from "@/components/layout/DashboardLayout"; import {
Card } from "@/components/ui/card"; import { Badge } from "@/components/ui/badge";
import { Button } from "@/components/ui/button"; import { Input } from
"@/components/ui/input"; import { Tabs, TabsList, TabsTrigger, TabsContent } from
"@/components/ui/tabs"; import { Select, SelectContent, SelectItem, SelectTrigger,
SelectValue } from "@/components/ui/select"; import { useEffect, useState } from
"react"; type Row = {
```

AELP2/external/growth-compass-77/src/pages/RLInsights.tsx

```
import { DashboardLayout } from "@/components/layout/DashboardLayout"; import {
Card } from "@/components/ui/card"; import { Badge } from "@/components/ui/badge";
import { Tabs, TabsContent, TabsList, TabsTrigger } from "@/components/ui/tabs";
import { Activity } from "lucide-react"; import { useOffpolicy, useInterference }
from "@/hooks/useAelp"; import { useEffect, useMemo, useState } from "react";
import { ResponsiveContainer, LineChart as RLineChart, Line, XAxis, YAxis, Tooltip,
CartesianGrid } from 'recharts' import { Link } from "react-router-dom";
```

AELP2/external/growth-compass-77/src/pages/SpendPlanner.tsx

```
import { DashboardLayout } from "@/components/layout/DashboardLayout"; import {
Card } from "@/components/ui/card"; import { Button } from
```

```
"@/components/ui/button"; import { Badge } from "@/components/ui/badge"; import {
Target, TrendingUp, DollarSign, Users, AlertCircle,
```

AELP2/external/growth-compass-77/src/pages/TrainingCenter.tsx

```
import { DashboardLayout } from "@/components/layout/DashboardLayout"; import {
Card } from "@/components/ui/card"; import { Badge } from "@/components/ui/badge";
import { GraduationCap, CheckCircle, AlertTriangle } from "lucide-react"; import {
useOpsStatus } from "@/hooks/useAelp"; import { useQuery } from
"@tanstack/react-query"; import { api } from "@/integrations/aelp-api/client";
import { toast } from "sonner"; export default function TrainingCenter() {
```

AELP2/external/growth-compass-77/src/vite-env.d.ts

```
/// <reference types="vite/client" />
```

AELP2/external/growth-compass-77/supabase/functions/ai-brainstorm/index.ts

```
import { serve } from "https://deno.land/std@0.168.0/http/server.ts"; import {
createClient } from 'https://esm.sh/@supabase/supabase-js@2'; const corsHeaders = {
'Access-Control-Allow-Origin': '*', 'Access-Control-Allow-Headers': 'authorization,
x-client-info, apikey, content-type', }; const supabaseUrl =
Deno.env.get('SUPABASE_URL')!; const supabaseKey =
Deno.env.get('SUPABASE_SERVICE_ROLE_KEY')!;
```

AELP2/external/growth-compass-77/supabase/functions/ai-generate-components/index.ts

```
import { serve } from "https://deno.land/std@0.168.0/http/server.ts"; import {
createClient } from 'https://esm.sh/@supabase/supabase-js@2'; const corsHeaders = {
'Access-Control-Allow-Origin': '*', 'Access-Control-Allow-Headers': 'authorization,
x-client-info, apikey, content-type', }; const supabaseUrl =
Deno.env.get('SUPABASE_URL')!; const supabaseKey =
Deno.env.get('SUPABASE_SERVICE_ROLE_KEY')!;
```

AELP2/external/growth-compass-77/supabase/functions/instagram-search/index.ts

```
import { serve } from "https://deno.land/std@0.168.0/http/server.ts"; import {
createClient } from 'https://esm.sh/@supabase/supabase-js@2'; const corsHeaders = {
'Access-Control-Allow-Origin': '*', 'Access-Control-Allow-Headers': 'authorization,
x-client-info, apikey, content-type', }; // Initialize Supabase client const
supabaseUrl = Deno.env.get('SUPABASE_URL')!;
```

AELP2/external/growth-compass-77/supabase/functions/test-integration/index.ts

```
import { serve } from "https://deno.land/std@0.168.0/http/server.ts"; const
corsHeaders = { 'Access-Control-Allow-Origin': '*', 'Access-Control-Allow-Headers':
'authorization, x-client-info, apikey, content-type', }; serve(async (req) => { //
Handle CORS preflight requests if (req.method === 'OPTIONS') {
```

AELP2/external/growth-compass-77/tailwind.config.ts

```
import type { Config } from "tailwindcss"; export default { darkMode: ["class"],
content: ["./pages/**/*.{ts,tsx}", "./components/**/*.{ts,tsx}",
"./app/**/*.{ts,tsx}", "./src/**/*.{ts,tsx}"], prefix: "", theme: { container: {
center: true, padding: "2rem",
```

AELP2/external/growth-compass-77/vite.config.ts

```
import { defineConfig } from "vite"; import react from "@vitejs/plugin-react";
import path from "path"; // https://vitejs.dev/config/ export default
defineConfig(({ mode }) => ({ define: { 'process.env.NODE_ENV':
JSON.stringify('development'), }, optimizeDeps: {
```

AELP2/flows/creative_batch.py

```
#!/usr/bin/env python3 from __future__ import annotations """ Prefect flow
(optional) to orchestrate: build assets -> assemble themes -> log DNA. If Prefect
is not installed, prints a CLI sequence instead. """ import shutil, subprocess def
have_prefect(): return shutil.which('prefect') is not None
```

AELP2/ops/gx/run_checks.py

```
#!/usr/bin/env python3 """ Great Expectations data quality gates for AELP2.
Implements real GE suites over Pandas DataFrames by default and optionally over
BigQuery tables when AELP2_GX_USE_BQ=1. Suites: - ads_campaign_performance: nonneg
metrics; clicks <= impressions; freshness <= 7d - ads_ad_performance: nonneg;
clicks <= impressions
```

AELP2/ops/prefect_flows.py

```
#!/usr/bin/env python3 """ Prefect flows scaffolding for MMM, Bandit, Uplift, and
Opportunity Scanner. Runs existing modules as tasks. If Prefect is not installed,
prints instructions. """ import os import subprocess import sys import json
```

AELP2/ops/reproducibility_snapshot.py

```
#!/usr/bin/env python3 """ Reproducibility snapshot: fetch last training_run config
and print commands/env to rehydrate. """ import os from google.cloud import
bigquery def main(): project = os.getenv('GOOGLE_CLOUD_PROJECT')
```

AELP2/ops/scheduler_stub.py

```
#!/usr/bin/env python3 """ Scheduler stub: prints gcloud commands for Cloud Run
Jobs / Cloud Scheduler. No live mutations; set DRY_RUN=0 to actually run gcloud if
available. """ import os def main(): project = os.getenv('GOOGLE_CLOUD_PROJECT',
'<project>')
```

AELP2/ops/todo_list.py

```
#!/usr/bin/env python3 import argparse import hashlib import json import os import
re import sys import time from datetime import datetime
```

AELP2/pipelines/ads_common.py

```
""" Shared utilities for Google Ads → BigQuery loaders. Strictness: - Read env for
all configuration (no hardcoding). - Redact free-text fields by default (hash) to
avoid storing raw campaign/ad/search strings. - Fail-fast with clear errors if
credentials missing. """ import os
```

AELP2/pipelines/ads_mcc_coordinator.py

```
#!/usr/bin/env python3 """ MCC Coordinator: Orchestrate Ads data loads across all
child accounts. Runs selected Ads loaders for each child account under the MCC,
with simple rate limiting between accounts to respect API quotas. Requirements: -
Env: GOOGLE_CLOUD_PROJECT, BIGQUERY_TRAINING_DATASET - Ads env:
GOOGLE_ADS_DEVELOPER_TOKEN, GOOGLE_ADS_CLIENT_ID, GOOGLE_ADS_CLIENT_SECRET,
```

AELP2/pipelines/attribution_engine_stub.py

```
#!/usr/bin/env python3 """ Real Attribution Engine Implementation for AELP2 Full multi-touch attribution with delayed rewards: - Data-driven attribution models (not simplified) - Real conversion lag modeling - Path analysis with proper credit distribution - Integration with existing attribution system - NO FALLBACKS - production implementation only
```

AELP2/pipelines/audience_expansion.py

```
#!/usr/bin/env python3 """ Audience Expansion Tooling (shadow-only). Reads recent search terms/keywords from Ads tables (if present) and proposes new keywords/audiences with simple heuristics. Writes to `<project>.<dataset>.audience_expansion_candidates`. Safe defaults: if inputs are missing, ensures the table and writes a single stub row with a clear note.
```

AELP2/pipelines/auto_recalibration.py

```
#!/usr/bin/env python3 """ Auto-recalibration (shadow-only): detect drift and log proposals. Logic: - Read last 14 days from `${dataset}.fidelity_evaluations`. - If KS or error metrics exceed thresholds (env or defaults), write a safety event and insert a proposal into `${dataset}.calibration_proposals` with recommended action. Never mutates live parameters. HITL required to apply any change elsewhere.
```

AELP2/pipelines/bandits_writer.py

```
#!/usr/bin/env python3 """Bandits Posteriors Writer (heuristic P0) Reads explore_cells and writes a simple CAC posterior summary to bandit_posteriors. Replace with MABWiser TS later. """ import os, json, random, time from datetime import datetime, timezone from google.cloud import bigquery # type: ignore PROJECT = os.environ.get('GOOGLE_CLOUD_PROJECT')
```

AELP2/pipelines/bid_edit_proposals.py

```
#!/usr/bin/env python3 """ Bid Edit Proposals (shadow-only, stub). Proposes ad-group max CPC adjustments based on simple CPC vs CTR proxy. Writes `<project>.<dataset>.bid_edit_proposals` with shadow=true. """ import os from datetime import datetime from google.cloud import bigquery
```

AELP2/pipelines/bid_landscape_modeling.py

```
#!/usr/bin/env python3 """ Bid Landscape Modeling (stub): derive CPC↔volume curves per campaign. Writes `<project>.<dataset>.bid_landscape_curves` with simple log curves. """ import os import math from google.cloud import bigquery from google.cloud.exceptions import NotFound
```

AELP2/pipelines/bing_to_bq.py

```
#!/usr/bin/env python3 """Stub: Bing Ads to BigQuery adapter (pilot) Fetch daily performance via API (to be implemented) and write to ext tables. """ import os from google.cloud import bigquery # type: ignore PROJECT = os.environ.get('GOOGLE_CLOUD_PROJECT') DATASET = os.environ.get('BIGQUERY_TRAINING_DATASET')
```

AELP2/pipelines/build_affiliate_triggered.py

```
#!/usr/bin/env python3 """ Affiliate-triggered (delayed-reward) series from GA4 export. Outputs in <project>.<dataset>: - ga_affiliate_triggered_channel_daily(date, channel, triggered_conversions) -
```

ga_affiliate_triggered_daily(date, triggered_conversions) Affiliate touch heuristic: - medium contains 'affiliate' OR

AELP2/pipelines/build_ga_aligned_daily.py

#!/usr/bin/env python3 """ Build GA-aligned KPI daily table that keeps GA as the majority signal and fills only the non-GA delta from pacer to reach 98–100% totals. Outputs in <project>.<dataset>: - ga4_purchases_utc_daily(date, purchases) # GA purchases by UTC day incl. intraday - ga_aligned_daily(date, ga_enrollments, pacer_subs, non_ga_delta, aligned_enrollments, spend)

AELP2/pipelines/build_impact_partner_domains.py

#!/usr/bin/env python3 """ Build partner→domain mapping from Impact MediaPartners table. Reads: <project>.<dataset>.impact_media_partners Writes: <project>.<dataset>.impact_partner_domains (partner_id STRING, partner STRING, domain STRING, root STRING)

AELP2/pipelines/build_triggered_series.py

#!/usr/bin/env python3 """ Build touch-aligned (delayed-reward) daily series from GA4 export. Outputs in <project>.<dataset>: - ga_triggered_channel_daily(date, channel, triggered_conversions) - ga_triggered_daily(date, triggered_conversions) Method - For each purchase, collect prior touches in a 14-day window.

AELP2/pipelines/calibration_parallel_stub.py

#!/usr/bin/env python3 """ Parallel calibration probes (stub): demonstrates multiprocessing fanout across probe params. """ import os import multiprocessing as mp def _probe(seed: int) -> int: # Placeholder: return 0 (success)

AELP2/pipelines/calibration_stratified_views.py

#!/usr/bin/env python3 """ Create calibration stratified views (by channel/device) for RL vs Ads. Views created when source tables exist: - `${dataset}.calibration_rl_by_channel_device` - `${dataset}.calibration_ads_by_channel_device` Idempotent; supports --dry_run. No data writes. """

AELP2/pipelines/canary_monitoring.py

#!/usr/bin/env python3 """ Canary Monitoring (shadow): detect anomalies and write ops_alerts. Rules (simple): - Spend spike on canary campaigns vs prior 7d avg > AELP2_ALERT_SPEND_DELTA_PCT (default 0.5) Writes `<project>.<dataset>.ops_alerts`. """ import os

AELP2/pipelines/canary_timeline_writer.py

#!/usr/bin/env python3 """ Writes a simple canary timeline to BQ for dashboard consumption. Table: `<project>.<dataset>.canary_timeline` with T-0/T-1/T-2 rows. Idempotent, partitioned, and safe: - DAY partitioned on `start_date`. - Never drops the table; replaces rows for stages T-0/T-1/T-2 via DML. - Supports `--dry_run` to avoid network access. """

AELP2/pipelines/channel_attribution_r.py

#!/usr/bin/env python3 """ ChannelAttribution (R) weekly job — containerized stub with BQ writer. Writes weekly attribution summary to <project>.<dataset>.channel_attribution_weekly. Shadow-only; runs container only

when AELP2_CA_ALLOW_RUN=1 and docker is present. Otherwise, writes a summary row noting stub status. """ from __future__ import annotations

AELP2/pipelines/channel_attribution_runner.py

#!/usr/bin/env python3 """ Channel Attribution Runner (skeleton) Intended to run an R-based ChannelAttribution (Markov/Shapley) job weekly and write summaries to BigQuery. If R/deps are unavailable, writes a placeholder summary and exits 0. """ import os from datetime import date, timedelta from google.cloud import bigquery

AELP2/pipelines/check_data_quality.py

#!/usr/bin/env python3 """ Basic data quality and freshness checks for core BigQuery tables. Checks: - Existence of required tables - Freshness (rows within last N days) for ads_campaign_performance and training_episodes - Null ratios for key fields Exit codes: 0 ok, 1 warnings, 2 failures.

AELP2/pipelines/competitive_intel_ingest.py

#!/usr/bin/env python3 """ Competitive Intelligence ingest (stub): ensure auction insights table exists. Writes `<project>.<dataset>.ads_auction_insights` schema only if missing. """ import os from google.cloud import bigquery from google.cloud.exceptions import NotFound

AELP2/pipelines/copy_optimizer_stub.py

#!/usr/bin/env python3 """ Copy Optimization Loop (policy-safe, stub). Generates simple copy suggestions based on recent CTR deltas by campaign. Writes `<project>.<dataset>.copy_suggestions` with shadow-only proposals. """ import os from datetime import datetime from google.cloud import bigquery

AELP2/pipelines/cost_monitoring_stub.py

#!/usr/bin/env python3 """ Cost monitoring (stub): compute daily cost and emit ops_alerts if exceeding cap. Env: AELP2_DAILY_COST_CAP (float) """ import os from datetime import datetime from google.cloud import bigquery

AELP2/pipelines/create_bq_views.py

#!/usr/bin/env python3 """ Create standard BigQuery views for dashboards/subagents. Views: - training_episodes_daily - ads_campaign_daily Requirements: - Env: GOOGLE_CLOUD_PROJECT, BIGQUERY_TRAINING_DATASET

AELP2/pipelines/create_channel_views.py

#!/usr/bin/env python3 """ Create channel-specific daily views from ads_campaign_performance using advertising_channel_type. Views created (if column exists): - google_search_campaign_daily - google_video_campaign_daily - google_discovery_campaign_daily - google_pmax_campaign_daily """

AELP2/pipelines/creative_ab_planner.py

#!/usr/bin/env python3 """ Creative AB Planner (shadow-only): proposes a test and writes to BQ. Tables: - `<project>.<dataset>.ab_experiments` - `<project>.<dataset>.creative_variants` """ import os from datetime import datetime

AELP2/pipelines/creative_bandit_head.py

```
#!/usr/bin/env python3 """ Creative Bandit Head (stub): logs decision proposals to ab_experiments (shadow-only). """ import os from datetime import datetime from google.cloud import bigquery def main():
```

AELP2/pipelines/creative_embeddings_stub.py

```
#!/usr/bin/env python3 """ Production Creative Embeddings System for AELP2 Real creative analysis and embeddings generation: - Vision AI for image analysis and feature extraction - NLP models for text/copy analysis and embeddings - Creative performance prediction based on visual/textual features - Real-time creative optimization recommendations - Integration with creative bandit systems
```

AELP2/pipelines/creative_fatigue_alerts.py

```
#!/usr/bin/env python3 """ Creative Fatigue Detection (stub): flags CTR/CVR decay over recent days. Writes `<project>.<dataset>.creative_fatigue_alerts`. """ import os from google.cloud import bigquery from google.cloud.exceptions import NotFound
```

AELP2/pipelines/cross_platform_kpi_daily.py

```
#!/usr/bin/env python3 """ Cross-Platform KPI Daily (stub): aggregates KPI metrics across platforms. For now, includes only Google Ads data into `<project>.<dataset>.cross_platform_kpi_daily`. """ import os from google.cloud import bigquery from google.cloud.exceptions import NotFound
```

AELP2/pipelines/dayparting_optimizer.py

```
#!/usr/bin/env python3 """ Dayparting Optimizer (stub): propose hour/day schedule caps. Writes `<project>.<dataset>.dayparting_schedules` with safe default caps. """ import os from google.cloud import bigquery from google.cloud.exceptions import NotFound
```

AELP2/pipelines/delayed_conversions_stub.py

```
#!/usr/bin/env python3 """ Production Delayed Conversions Processing for AELP2 Real delayed conversion attribution with: - Multi-day attribution windows (3-14 days) - Conversion lag modeling based on real data - Attribution credit redistribution for delayed events - Integration with attribution engine for retroactive reward assignment - No stub implementations - production delayed conversion system
```

AELP2/pipelines/fidelity_evaluation.py

```
#!/usr/bin/env python3 """ Fidelity Evaluation: Compare simulation/RL telemetry vs GA4/Ads (MAPE/RMSE/KS) and write results to BigQuery. Requirements: - Env: GOOGLE_CLOUD_PROJECT, BIGQUERY_TRAINING_DATASET - Threshold envs (fail fast if set but not valid): - AELP2_FIDELITY_MAX_MAPE_ROAS, AELP2_FIDELITY_MAX_MAPE_CAC (floats) - AELP2_FIDELITY_MAX_RMSE_ROAS, AELP2_FIDELITY_MAX_RMSE_CAC (floats) - AELP2_FIDELITY_MAX_KS_WINRATE (float) — KS for RL win_rate vs Ads impression_share
```

AELP2/pipelines/fidelity_parallel_stub.py

```
#!/usr/bin/env python3 """ Parallel fidelity evaluation (stub): shards dates and would call fidelity_evaluation per shard. """ import os from datetime import date, timedelta def main(): shards = int(os.getenv('AELP2_FIDELITY_SHARDS', '4'))
```

AELP2/pipelines/ga4_backfill_audience_breakdown.py

```
#!/usr/bin/env python3 import os from datetime import date, timedelta from google.cloud import bigquery import yaml def load_mapping(path: str): with
```

```
open(path, 'r') as f: return yaml.safe_load(f)
```

AELP2/pipelines/ga4_build_attribution.py

```
#!/usr/bin/env python3 """ Build GA attribution daily tables from export using purchase-level cookies (first pass LND proxy): - ga4_attribution_daily(date, source, medium, purchases, initial_payment_revenue) Notes: - Uses purchase event's source_cookie/medium_cookie as last-non-direct proxy. - Adds host (landing host) for debugging. """ import os
```

AELP2/pipelines/ga4_build_audience_breakdown.py

```
#!/usr/bin/env python3 """ Build high-intent breakdown by channel (source/medium) from GA4 export. Output: <project>.<dataset>.ga4_high_intent_by_channel_daily - date, source, medium, enrollment_loaded, begin_checkout, form_submit_enroll, high_intent_no_purchase_7 Notes: - Uses daily user-level intent flags and picks the latest non-null source/medium cookie seen that day per user. - Excludes users who purchase within post_trial_window_days (from mapping) after the intent day.
```

AELP2/pipelines/ga4_build_audiences.py

```
#!/usr/bin/env python3 """ Build high-intent audiences from GA4 export: - ga4_high_intent_daily: date, enrollment_loaded, begin_checkout, form_submit_enroll, high_intent_no_purchase_7 Uses: - enrollment_loaded param presence - begin_checkout / Checkout events - form_submit on enrollment/onboarding pages - Excludes users who purchase within N days (default 7)
```

AELP2/pipelines/ga4_build_derived.py

```
#!/usr/bin/env python3 """ Build GA-derived daily/monthly KPI tables and cohort summaries from GA4 export (events_*). Outputs (in <project>.<dataset>): - ga4_derived_daily(date, enrollments, d2p_starts, post_trial_subs, mobile_subs) - ga4_derived_monthly(month, enrollments, d2p_starts, post_trial_subs, mobile_subs) - ga4_offer_code_daily(date, plan_code, cc, source, medium, host, purchases, revenue) Config: AELP2/config/ga4_mapping.yaml
```

AELP2/pipelines/ga4_extract_affiliate_clickids.py

```
#!/usr/bin/env python3 """ Extract affiliate click IDs from GA4 export into a joinable table. Output: <project>.<dataset>.ga_affiliate_clickids Fields: date DATE, event_ts TIMESTAMP, user_pseudo_id STRING, click_id STRING, source STRING, medium STRING We parse clickid/irclickid/ir_clickid from page_location and from event_params. """ import os
```

AELP2/pipelines/ga4_lagged_attribution.py

```
#!/usr/bin/env python3 """ GA4 Lagged Attribution Importer Computes a lag-aware attribution of GA4 conversions to prior days and writes results into `<project>.<dataset>.ga4_lagged_attribution`. The attribution redistributes GA4 daily conversions for each day D backward to dates D-L where L is within [MIN_LAG_DAYS, MAX_LAG_DAYS] using a triangular weighting that peaks at PEAK_LAG_DAYS (default 7).
```

AELP2/pipelines/ga4_paths_attribution.py

```
#!/usr/bin/env python3 """ Build multi-touch attribution from GA4 export: - Position-based 40/20/40 daily credits per channel: <project>.<dataset>.ga4_attribution_daily_pb - Markov removal effects (30d aggregate): <project>.<dataset>.ga4_markov_contrib_30d Approach - Construct
```

per-user paths of channels (source/medium) for each purchase, using a 30-day lookback. - Position-based weights: 0.4 to first, 0.4 to last, remaining 0.2 split equally among middle touches. - Markov: aggregate paths (Start -> ... -> Conversion) and compute removal effects in Python.

AELP2/pipelines/ga4_permissions_check.py

#!/usr/bin/env python3 """ GA4 permissions check (dry-run friendly): verifies Data API access or provides guidance. """ import os import argparse def main(): ap = argparse.ArgumentParser()

AELP2/pipelines/ga4_to_bq.py

#!/usr/bin/env python3 """ Ingest GA4 aggregates into BigQuery (ga4_aggregates). Note: For raw events, set up native GA4→BigQuery export. This script uses the GA4 Data API for aggregated daily metrics to bootstrap calibration/monitoring. Requirements: - Env: GOOGLE_CLOUD_PROJECT, BIGQUERY_TRAINING_DATASET, GA4_PROPERTY_ID - ADC configured for BigQuery and GA4 Data API access

AELP2/pipelines/generate_qs_alerts.py

#!/usr/bin/env python3 import os from datetime import datetime from google.cloud import bigquery def main(): project=os.environ['GOOGLE_CLOUD_PROJECT'] dataset=os.environ['BIGQUERY_TRAINING_DATASET'] bq=bigquery.Client(project=project) tbl=f"{project}.{dataset}.ops_alerts"

AELP2/pipelines/generate_qs_fix_tickets.py

#!/usr/bin/env python3 import os, time from datetime import datetime from google.cloud import bigquery def main(): project=os.environ['GOOGLE_CLOUD_PROJECT'] dataset=os.environ['BIGQUERY_TRAINING_DATASET'] bq=bigquery.Client(project=project) table=f"{project}.{dataset}.qs_fix_tickets"

AELP2/pipelines/google_ads_ad_performance_to_bq.py

#!/usr/bin/env python3 """ Load Google Ads ad performance into BigQuery (ads_ad_performance). Fields include ad id, ad group, campaign, (redacted) ad name/headline, impressions, clicks, cost, conversions, value. Requirements: - Env: GOOGLE_CLOUD_PROJECT, BIGQUERY_TRAINING_DATASET - Ads env: GOOGLE_ADS_DEVELOPER_TOKEN, GOOGLE_ADS_CLIENT_ID, GOOGLE_ADS_CLIENT_SECRET, GOOGLE_ADS_REFRESH_TOKEN, GOOGLE_ADS_CUSTOMER_ID

AELP2/pipelines/google_ads_adgroups_to_bq.py

#!/usr/bin/env python3 """ Load Google Ads ad group performance into BigQuery (ads_ad_group_performance). Redacts campaign/ad_group names by default. """ import os import argparse import logging from datetime import datetime

AELP2/pipelines/google_ads_assets_to_bq.py

#!/usr/bin/env python3 """ Google Ads Assets ingestion. Ensures `<project>.<dataset>.ads_assets` table and fetches basic asset metadata (TEXT/YOUTUBE_VIDEO/IMAGE). Stores text (for TEXT assets), youtube_id, image_url, and policy summary if available. """ import os from datetime import datetime

AELP2/pipelines/google_ads_conversion_actions_to_bq.py

#!/usr/bin/env python3 """ Load Google Ads conversion actions into BigQuery (ads_conversion_actions). This loader ingests conversion action definitions (id,

name, category, type, status, primary_for_goal) for use in attribution mapping and downstream analytics. Names are redacted by default. Requirements: - Env: GOOGLE_CLOUD_PROJECT, BIGQUERY_TRAINING_DATASET - Ads env: GOOGLE_ADS_DEVELOPER_TOKEN, GOOGLE_ADS_CLIENT_ID, GOOGLE_ADS_CLIENT_SECRET,

AELP2/pipelines/google_ads_conversion_stats_by_action_to_bq.py

#!/usr/bin/env python3 """ Load Google Ads conversion stats by conversion_action into BigQuery (ads_conversion_action_stats). Fields: date, customer_id, campaign_id, conversion_action_id, conversion_action_name (redacted), conversions, conversion_value, cost_micros Usage: python -m AELP2.pipelines.google_ads_conversion_stats_by_action_to_bq --start YYYY-MM-DD --end YYYY-MM-DD [--customer 1234567890]

AELP2/pipelines/google_ads_discover_accounts.py

#!/usr/bin/env python3 """ Discover accessible Google Ads accounts (including MCC/manager accounts). Prints customer IDs with basic metadata and whether each is a manager (MCC). Requirements: - Env: GOOGLE_ADS_DEVELOPER_TOKEN, GOOGLE_ADS_CLIENT_ID, GOOGLE_ADS_CLIENT_SECRET, GOOGLE_ADS_REFRESH_TOKEN Usage:

AELP2/pipelines/google_ads_geo_device_to_bq.py

#!/usr/bin/env python3 """ Load Google Ads device-level performance into BigQuery (ads_geo_device_performance). Captures performance by device for campaign rollups. """ import os import argparse import logging

AELP2/pipelines/google_ads_keywords_to_bq.py

#!/usr/bin/env python3 """ Load Google Ads keyword performance into BigQuery (ads_keyword_performance). Fields include impressions, clicks, cost, conversions, value, CTR, CPC, keyword text (redacted by default). """ import os import argparse import logging

AELP2/pipelines/google_ads_mcc_to_bq.py

#!/usr/bin/env python3 """ Enumerate all child accounts under an MCC and load Ads performance into BigQuery. Requirements: - Env: GOOGLE_CLOUD_PROJECT, BIGQUERY_TRAINING_DATASET - Ads env: GOOGLE_ADS_DEVELOPER_TOKEN, GOOGLE_ADS_CLIENT_ID, GOOGLE_ADS_CLIENT_SECRET, GOOGLE_ADS_REFRESH_TOKEN, GOOGLE_ADS_LOGIN_CUSTOMER_ID (MCC) """

AELP2/pipelines/google_ads_search_terms_to_bq.py

#!/usr/bin/env python3 """ Load Google Ads search terms into BigQuery (ads_search_terms). Redacts search term text by default. """ import os import argparse import logging from datetime import datetime

AELP2/pipelines/google_ads_to_bq.py

#!/usr/bin/env python3 """ Ingest Google Ads performance into BigQuery (ads_campaign_performance). Requirements: - Env: GOOGLE_CLOUD_PROJECT, BIGQUERY_TRAINING_DATASET - Google Ads env: GOOGLE_ADS_DEVELOPER_TOKEN, GOOGLE_ADS_CLIENT_ID, GOOGLE_ADS_CLIENT_SECRET, GOOGLE_ADS_REFRESH_TOKEN, GOOGLE_ADS_CUSTOMER_ID (10 digits, no dashes) - ADC configured for BigQuery: gcloud auth application-default login

AELP2/pipelines/google_recommendations_scanner.py

#!/usr/bin/env python3 """ Google Ads Recommendations Scanner (safe, shadow-only). - Tries the Google Ads Recommendations API when `google-ads` + creds are available. - Else, falls back to simple BQ heuristics (if available) to propose quick wins. - If neither is available, writes a stub note so dashboards have a safe default. Outputs: - `<project>.<dataset>.platform_skeletons` (paused candidates with rationale)

AELP2/pipelines/gsc_to_bq.py

#!/usr/bin/env python3 """ Google Search Console → BigQuery (brand vs non-brand trend). Env: GSC_SITE_URL, GOOGLE_CLOUD_PROJECT, BIGQUERY_TRAINING_DATASET For auth: Application Default Credentials with webmasters.readonly scope. Writes: <project>.<dataset>.gsc_brand_monthly (month, brand_clicks, all_clicks, brand_share) """ import os

AELP2/pipelines/hints_to_proposals.py

#!/usr/bin/env python3 """ Promote policy hints to shadow proposals (HITL path, stub). Reads `<project>.<dataset>.policy_hints` and writes shadow proposals into `<project>.<dataset>.bandit_change_proposals`. """ import os from datetime import datetime from google.cloud import bigquery

AELP2/pipelines/impact_backfill_performance.py

#!/usr/bin/env python3 """ Impact.com daily performance backfill (auto-discovery). Steps: 1) List advertiser reports and pick candidates that look daily (contain "Performance" and "Date"). 2) For each candidate, try fetching CSV month-by-month over the requested window with several date param styles. 3) On first candidate that returns non-empty rows, backfill months into <project>.<dataset>.impact_partner_performance (partitioned by date), upserting per month.

AELP2/pipelines/impact_clicks_to_bq.py

#!/usr/bin/env python3 from __future__ import annotations import os, time, csv, io import argparse import requests from typing import Dict, Any, List from google.cloud import bigquery API_BASE = "https://api.impact.com"

AELP2/pipelines/impact_entities_to_bq.py

#!/usr/bin/env python3 """ Impact.com entities → BigQuery (MediaPartners, Ads, Deals, Invoices, TrackingValueRequests). Auth (env): IMPACT_ACCOUNT_SID, IMPACT_AUTH_TOKEN (Basic) or IMPACT_BEARER_TOKEN (Bearer) BQ (env): GOOGLE_CLOUD_PROJECT, BIGQUERY_TRAINING_DATASET Usage: python3 -m AELP2.pipelines.impact_entities_to_bq.py --entities media_partners,ads,deals,invoices,tvr """

AELP2/pipelines/impact_run_report_to_bq.py

#!/usr/bin/env python3 from __future__ import annotations import os, time, csv, io, json import argparse import requests from typing import Dict, Any, List from google.cloud import bigquery API_BASE = "https://api.impact.com"

AELP2/pipelines/impact_to_bq.py

#!/usr/bin/env python3 """ Impact.com (Impact Radius) → BigQuery loader. Pulls Advertiser Reports via Impact.com REST API using Basic Auth (Account SID + Auth Token) and writes normalized rows to <project>.<dataset>.impact_partner_performance. Auth (env): IMPACT_ACCOUNT_SID,

IMPACT_AUTH_TOKEN

AELP2/pipelines/journey_path_summary.py

#!/usr/bin/env python3 """ Summarize user journey paths and transition probabilities. Reads from <project>.gaelp_users.journey_sessions with columns: - user_id STRING - session_start TIMESTAMP - default_channel_group STRING (or channel STRING) Writes to <project>.<dataset>.journey_paths_daily for current date.

AELP2/pipelines/journeys_populate.py

#!/usr/bin/env python3 """ Journeys Populate (bootstrap) Ensures gaelp_users.journey_sessions and persistent_touchpoints exist. If GA4 export dataset is available (env GA4_EXPORT_DATASET), extracts a minimal set of sessions and touchpoints for the last N days and writes sample rows. Safe to run repeatedly; no-ops if GA4 export isn’t configured. """

AELP2/pipelines/kpi_consistency_check.py

#!/usr/bin/env python3 """ KPI Consistency Check: compares KPI-only view vs training episodes daily metrics. Writes `<project>.<dataset>.kpi_consistency_checks` with diffs for CAC/ROAS. """ import os from google.cloud import bigquery from google.cloud.exceptions import NotFound

AELP2/pipelines/kpi_crosscheck.py

#!/usr/bin/env python3 """ KPI Cross-check: compare ads_kpi_daily view vs aggregated ads_campaign_performance. Writes results to `<project>.<dataset>.kpi_crosscheck_daily` with per-day diffs and status. """ import os from google.cloud import bigquery

AELP2/pipelines/linkedin_to_bq.py

#!/usr/bin/env python3 import os from google.cloud import bigquery from google.cloud.exceptions import NotFound def ensure_table(bq: bigquery.Client, project: str, dataset: str): table_id = f"{project}.{dataset}.linkedin_ad_performance" try: bq.get_table(table_id) return

AELP2/pipelines/load_affiliate_ach_costs.py

#!/usr/bin/env python3 """ Load external affiliate (ACH) payouts into BigQuery to complete partner costs. Input CSV (optional): AELP2/data/affiliate_ach_costs.csv Columns (header required): date, partner_id, partner, amount, memo - date: YYYY-MM-DD (posting or performance date you want costs recognized) - partner_id: Impact MediaId as string (preferred). If empty, we try to map by partner name. - partner: Partner name (for mapping / display) - amount: positive float USD

AELP2/pipelines/load_promo_calendar.py

#!/usr/bin/env python3 import os, csv from google.cloud import bigquery def main(): project=os.environ['GOOGLE_CLOUD_PROJECT'] dataset=os.environ['BIGQUERY_TRAINING_DATASET'] path=os.environ.get('PROMO_CALENDAR_PATH','AELP2/config/promo_calendar.csv') bq=bigquery.Client(project=project) table_id=f"{project}.{dataset}.promo_calendar"

AELP2/pipelines/lp_ab_hooks_stub.py

#!/usr/bin/env python3 """ Landing-Page A/B Hooks (stub): propose UTM cohorts and GA4 goal names. Writes `<project>.<dataset>.lp_ab_candidates` with shadow-only

proposals. """ import os from datetime import datetime from google.cloud import bigquery from google.cloud.exceptions import NotFound

AELP2/pipelines/ltv_priors.py

#!/usr/bin/env python3 """ LTV Priors Daily: computes simple 30/90-day LTV priors per segment from uplift scores. Writes to <project>.<dataset>.ltv_priors_daily. """ import os, json from datetime import date from google.cloud import bigquery from google.cloud.exceptions import NotFound

AELP2/pipelines/meta_to_bq.py

#!/usr/bin/env python3 """ Meta → BigQuery loader (schema ensure + ingestion). Standardized schema: <dataset>.meta_ad_performance - date (DATE), campaign_id STRING, adset_id STRING, ad_id STRING, impressions INT64, clicks INT64, cost FLOAT64, conversions FLOAT64, revenue FLOAT64, ctr FLOAT64, cvr FLOAT64, avg_cpc FLOAT64, name_hash STRING Auth: Provide either a user access token (OAuth login; 60■day) or a Business System User token.

AELP2/pipelines/mmm_attributed_service.py

#!/usr/bin/env python3 """ MMM with Proper Delayed Reward Attribution (3-14 day windows) This version properly attributes conversions back to the spend that caused them, not just same-day attribution. Uses the sophisticated reward_attribution.py logic. Key improvements: - Tracks user journeys over 3-14 days - Attributes conversions to the right touchpoints

AELP2/pipelines/mmm_lightweightmmm.py

#!/usr/bin/env python3 """ LightweightMMM runner with safe fallbacks and BQ logging. Goals (P0): - Try to run a LightweightMMM Bayesian fit and write curves + credible intervals to BigQuery. - If dependencies are missing or runtime fails, log a clear "install needed" note and fall back to MMM v1 (bootstrap log–log + adstock) with bootstrap CIs, or to a dry-run synthetic mode. - Shadow-only: This module only reads BigQuery and writes results back; no platform mutations.

AELP2/pipelines/mmm_service.py

#!/usr/bin/env python3 """ MMM v1 (Lightweight, dependency-free): Fits a simple diminishing-returns response curve on daily Google Ads spend and conversions/revenue using a log-log model with an optional adstock transform. Writes two BigQuery tables: - <project>.<dataset>.mmm_curves Fields: timestamp, channel, window_start, window_end, model, params(JSON), spend_grid(JSON), conv_grid(JSON), rev_grid(JSON), diagnostics(JSON)

AELP2/pipelines/model_registry_stub.py

#!/usr/bin/env python3 """ Production Model Registry for AELP2 Real model versioning, storage, and lifecycle management: - MLflow integration for model tracking - Model performance monitoring - Automatic model deployment pipelines - A/B testing framework for model variants - No stub implementations - production-ready system

AELP2/pipelines/module_runner.py

#!/usr/bin/env python3 """ LP Module Runner (background job) Scans lp_module_runs with status='running', executes a lightweight connector, writes module_results and marks status='done'. Demo-safe for P0. """ import os, json, time from datetime import datetime, timedelta, timezone from typing import Dict, Any

AELP2/pipelines/namespace_refactor_report.py

```
#!/usr/bin/env python3 """ Namespace Refactor Report (stub): scans repo for non-AELP2 imports and prints summary. Writes `<project>.<dataset>.namespace_refactor_report` with counts per prefix (optional). """ import os, re from datetime import datetime try: from google.cloud import bigquery
```

AELP2/pipelines/offpolicy_eval.py

```
#!/usr/bin/env python3 """ Off-policy evaluation (stub): compares hints vs realized outcomes. Writes `<project>.<dataset>.offpolicy_eval_results` with placeholder metrics. """ import os from datetime import datetime from google.cloud import bigquery from google.cloud.exceptions import NotFound
```

AELP2/pipelines/opportunity_outcomes.py

```
#!/usr/bin/env python3 """ Opportunity Outcomes (stub): summarize approvals and subsequent performance. Writes `<project>.<dataset>.opportunity_outcomes` with counts and recent spend after approvals. If inputs missing, ensures table and exits. """ import os from google.cloud import bigquery from google.cloud.exceptions import NotFound
```

AELP2/pipelines/opportunity_scanner.py

```
#!/usr/bin/env python3 """ Opportunity Scanner v1 (Google-first, shadow) Surfaces headroom and expansion candidates across: - Search (brand vs non-brand headroom via impression_share and CPA) - YouTube/Demand Gen/Discovery (placeholder until adapters enriched) - PMax (placeholder; gated) Outputs are logged to `platform_skeletons` with rationale/expected CAC/volume (shadow).
```

AELP2/pipelines/ops_alerts_stub.py

```
#!/usr/bin/env python3 """ Ops Alerts (stub): writes a placeholder alert row. """ import os from datetime import datetime from google.cloud import bigquery from google.cloud.exceptions import NotFound
```

AELP2/pipelines/parity_report.py

```
#!/usr/bin/env python3 """ Parity Report (stub): compares ads_kpi_daily vs training_episodes_daily. Writes `<project>.<dataset>.parity_reports` with daily RMSE and rel diffs. """ import os, math from google.cloud import bigquery from google.cloud.exceptions import NotFound
```

AELP2/pipelines/permissions_check.py

```
#!/usr/bin/env python3 """ Permissions & Accounts Checker (writes to BQ). Checks presence of key env vars and records a status row in `<project>.<dataset>.permissions_checks`. """ import os from datetime import datetime from google.cloud import bigquery from google.cloud.exceptions import NotFound
```

AELP2/pipelines/platform_skeleton_log.py

```
#!/usr/bin/env python3 import os import json from datetime import datetime from typing import Optional from google.cloud import bigquery from google.cloud.exceptions import NotFound import argparse
```

AELP2/pipelines/policy_hints_writer.py

```
#!/usr/bin/env python3 """ Policy Hints writer (stub): writes exploration/budget tilt hints to BQ. Writes `<project>.<dataset>.policy_hints` with sample hint rows (shadow). """ import os from datetime import datetime from google.cloud import bigquery from google.cloud.exceptions import NotFound
```

AELP2/pipelines/portfolio_optimizer.py

```
#!/usr/bin/env python3 """ Portfolio Optimizer (stub): propose daily cross-campaign allocations under CAC cap. Writes `<project>.<dataset>.portfolio_allocations` with proposed budget shares. """ import os from google.cloud import bigquery from google.cloud.exceptions import NotFound
```

AELP2/pipelines/privacy_audit_stub.py

```
#!/usr/bin/env python3 """ Privacy audit (stub): ensures free-text fields are not stored; writes summary row. """ import os from datetime import datetime from google.cloud import bigquery def main():
```

AELP2/pipelines/propensity_uplift.py

```
#!/usr/bin/env python3 """ Propensity/Uplift Bootstrap: writes `segment_scores_daily` using simple exposed vs unexposed deltas. If journey tables are empty, creates the table and exits gracefully. """ import os import json from google.cloud import bigquery from google.cloud.exceptions import NotFound
```

AELP2/pipelines/quality_signal_daily.py

```
#!/usr/bin/env python3 """ Quality Signal Daily (stub): computes a simple quality proxy (trial→paid, retention). Writes `<project>.<dataset>.quality_signal_daily` with safe defaults if inputs are missing. """ import os from google.cloud import bigquery from google.cloud.exceptions import NotFound
```

AELP2/pipelines/quality_softgate_stub.py

```
#!/usr/bin/env python3 """ Quality soft-gate (stub): reads quality_signal_daily and emits safety_events when below threshold. Env: AELP2_QUALITY_MIN (float 0..1) """ import os from datetime import datetime from google.cloud import bigquery
```

AELP2/pipelines/realtime_budget_pacer.py

```
#!/usr/bin/env python3 """ Real-time budget pacer (stub): emits pacing proposals at minute cadence. Writes `<project>.<dataset>.budget_pacing_proposals`. """ import os from datetime import datetime from google.cloud import bigquery from google.cloud.exceptions import NotFound
```

AELP2/pipelines/reconcile_posthoc.py

```
#!/usr/bin/env python3 """ Post-hoc reconciliation of RL vs Ads/GA4 metrics by date. Writes to `<project>.<dataset>.training_episodes_posthoc`: - date, rl_spend, rl_revenue, rl_conversions, rl_roas, rl_cac - ads_cost, ads_conversions, ads_revenue, ads_ctr, ads_cvr, ads_roas, ads_cac, ads_is_p50 - ga4_conversions (if available) - computed_at timestamp
```

AELP2/pipelines/rl_policy_hints_writer.py

```
#!/usr/bin/env python3 """ RL Policy Hints Writer (bootstrap) Ensures the `policy_hints` table exists and provides a simple CLI to write hint rows produced
```

by the RL lab (offline). This does not run RL; it’s a bridge. Schema fields: - timestamp TIMESTAMP - source STRING (e.g., 'rl_lab')

AELP2/pipelines/robyn_runner.py

#!/usr/bin/env python3 """ Robyn Validator Runner (skeleton) Placeholder that would containerize and run Robyn (R) weekly to validate MMM curves. Currently prints guidance and exits 0 to avoid blocking. """ import sys print('Robyn runner scaffold: containerize R job to validate MMM curves; not blocking now.') sys.exit(0)

AELP2/pipelines/robyn_validator.py

#!/usr/bin/env python3 """ Robyn weekly validator (containerized R) — runner stub with BQ summary write. Shadow-only: Does not mutate platforms. Attempts to run an R container only if `AELP2_ROBYN_ALLOW_RUN=1` and `docker` is available; otherwise writes a summary row to BigQuery noting that the run is pending/setup required. Creates/uses table: <project>.<dataset>.mmm_validation (time-partitioned) Fields:

AELP2/pipelines/rule_engine.py

#!/usr/bin/env python3 """ Rule Engine (stub): evaluates safe rules and records actions (HITL required). Writes `<project>.<dataset>.rule_engine_actions`. """ import os from datetime import datetime from google.cloud import bigquery from google.cloud.exceptions import NotFound

AELP2/pipelines/security_audit.py

#!/usr/bin/env python3 """ Security Audit (enhanced stub): records IAM/audit status notes and ADC context. Writes `<project>.<dataset>.iam_audit` with notes including ADC project/account and presence of recommended env vars. Does not modify IAM; informational only. """ import os from datetime import datetime from google.cloud import bigquery

AELP2/pipelines/segments_to_audiences.py

#!/usr/bin/env python3 """ Segments → Audiences mapping (shadow-only). Reads latest `segment_scores_daily`, selects top-N segments, and maps to platform audience keys (generic placeholders). Writes rows to `<project>.<dataset>.segment_audience_map` with shadow=true and rationale. If dependencies/tables are missing, creates the table and exits gracefully. """

AELP2/pipelines/slo_watch_stub.py

#!/usr/bin/env python3 """ SLO/Alerting stub: scans ops_flow_runs and safety_events for failures; emits ops_alerts. """ import os from datetime import datetime from google.cloud import bigquery def main():

AELP2/pipelines/tiktok_to_bq.py

#!/usr/bin/env python3 import os from google.cloud import bigquery from google.cloud.exceptions import NotFound def ensure_table(bq: bigquery.Client, project: str, dataset: str): table_id = f"{project}.{dataset}.tiktok_ad_performance" try: bq.get_table(table_id) return

AELP2/pipelines/training_posthoc_reconciliation.py

#!/usr/bin/env python3 """ Post-hoc reconciliation (lag-aware KPIs). Writes `<project>.<dataset>.training_episodes_posthoc` by joining Ads spend with GA4

lagged conversions for each date, computing lag-aware CAC/ROAS. Safe if inputs missing: ensures table and exits with a note. """ from __future__ import annotations

AELP2/pipelines/trust_gates_evaluator.py

#!/usr/bin/env python3 """ Trust Gates Evaluator: computes pass/fail for pilot gates and writes to BQ. Table: `<project>.<dataset>.trust_gates` with rows per gate and status. """ import os import json from datetime import datetime from google.cloud import bigquery

AELP2/pipelines/uplift_eval.py

#!/usr/bin/env python3 """ Uplift v1: Bootstrap segment uplift evaluation from journey tables (if present). Reads `gaelp_users.persistent_touchpoints` joined to simple conversion flags and computes per-segment exposure rates and conversion rates for exposed vs. unexposed cohorts. Writes results to `<project>.<dataset>.uplift_segment_daily`. If journey tables are missing, creates the output table and exits gracefully. """

AELP2/pipelines/upload_google_offline_conversions.py

#!/usr/bin/env python3 """ Google Offline Conversions Upload (HITL-gated): Builds payloads with hashed PII for Enhanced Conversions and logs an entry to `<project>.<dataset>.value_uploads_log`. If `AELP2_ALLOW_VALUE_UPLOADS=1` and `AELP2_VALUE_UPLOAD_DRY_RUN=0`, will attempt real API call via google-ads. Otherwise logs intent only. """ import os

AELP2/pipelines/upload_meta_capi_conversions.py

#!/usr/bin/env python3 """ Meta CAPI Conversions Upload (HITL-gated): builds hashed payload and logs intent. """ import os from datetime import datetime from google.cloud import bigquery from google.cloud.exceptions import NotFound import json import urllib.request

AELP2/pipelines/users_db_stub.py

#!/usr/bin/env python3 """ Production User Database Management for AELP2 Real user data pipeline with: - User profile management and segmentation - Journey tracking with real behavioral data - Privacy-compliant user data handling - Real-time user state management - Integration with RecSim for user simulation

AELP2/pipelines/value_bridge.py

#!/usr/bin/env python3 """ Value-Based Bidding Bridge (stubs) Prepares offline conversion payloads with predicted values for upload to ads platforms. This is a stub that assembles payloads from BQ and prints or writes to staging table. Note: Actual upload requires platform-specific SDKs and consented hashing; we gate that behind HITL and separate scripts. """

AELP2/pipelines/youtube_reach_planner.py

#!/usr/bin/env python3 """ YouTube Reach Planner — real API when available. Writes estimated reach metrics to BigQuery. Attempts to call Google Ads ReachPlanService when google-ads SDK and credentials are present; otherwise falls back to a safe stub row with a clear note. """ import os from datetime import date

AELP2/scripts/add_search_assets.py

```
#!/usr/bin/env python3 """ Add a Search ad group, keywords, and a paused RSA to an existing campaign. Env required: - GOOGLE_ADS_DEVELOPER_TOKEN - GOOGLE_ADS_CLIENT_ID - GOOGLE_ADS_CLIENT_SECRET - GOOGLE_ADS_REFRESH_TOKEN - (optional) GOOGLE_ADS_LOGIN_CUSTOMER_ID (MCC)
```

AELP2/scripts/apply_google_canary.py

```
#!/usr/bin/env python3 """ Apply safe, canary budget changes to Google Ads campaigns. Defaults to shadow mode (dry run). Set AELP2_ALLOW_GOOGLE_MUTATIONS=1 and AELP2_SHADOW_MODE=0 to perform real mutations. Guardrails enforce a maximum delta percent and a max number of changes per run. Env: GOOGLE_ADS_DEVELOPER_TOKEN, GOOGLE_ADS_CLIENT_ID, GOOGLE_ADS_CLIENT_SECRET,
```

AELP2/scripts/apply_google_creatives.py

```
#!/usr/bin/env python3 """ Apply creative changes (shadow-first, flag-gated). Defaults to shadow mode (no live mutations). Real platform changes require both: - GATES_ENABLED=1 and AELP2_ALLOW_BANDIT_MUTATIONS=1 (and ALLOW_REAL_CREATIVE=1 optional) BigQuery tables (ensured if credentials present): - `${GOOGLE_CLOUD_PROJECT}.${BIGQUERY_TRAINING_DATASET}.creative_changes` (DAY partitioned on timestamp)
```

AELP2/scripts/apply_schemas.py

```
#!/usr/bin/env python3 """Apply core BigQuery schemas (idempotent)."""" import os from google.cloud import bigquery # type: ignore PROJECT = os.environ.get('GOOGLE_CLOUD_PROJECT') DATASET = os.environ.get('BIGQUERY_TRAINING_DATASET') USERS = os.environ.get('BIGQUERY_USERS_DATASET', 'gaelp_users') DDL = [
```

AELP2/scripts/assess_headroom.py

```
#!/usr/bin/env python3 """ Assess optimization headroom from real Ads + KPI data in BigQuery. Outputs: - Baseline KPI CAC/ROAS over last 30d - % spend with zero KPI conversions - % spend with CAC > 1.5× median KPI CAC (tail) - Impression share hints: overbidding vs budget-constrained spend shares - Top campaigns in high-CAC and zero-conversion buckets
```

AELP2/scripts/baseline_report.py

```
#!/usr/bin/env python3 """ Baseline Report (last 28 days) from BigQuery for Google Ads and GA4. Outputs consolidated KPIs: - Impression share (p50), impressions, clicks, conversions, CTR, CVR, CAC, ROAS (Ads) - GA4 sessions and conversions (if available) Env required: - GOOGLE_CLOUD_PROJECT, BIGQUERY_TRAINING_DATASET
```

AELP2/scripts/canary_rollback.py

```
#!/usr/bin/env python3 """ Canary rollback (shadow-only): writes rollback intents into BigQuery. Reads last N rows from `<project>.<dataset>.canary_changes` and creates `<project>.<dataset>.canary_rollbacks` entries reverting budgets to old_budget. No live mutations are performed. """ import os import argparse
```

AELP2/scripts/check_canary_readiness.py

```
#!/usr/bin/env python3 """ Check canary readiness gates based on recent fidelity and KPI metrics. Gates (configurable via env): - AELP2_FIDELITY_MAX_MAPE_ROAS (default 0.5) - AELP2_FIDELITY_MAX_MAPE_CAC (default 0.5) - AELP2_FIDELITY_MAX_KS_WINRATE (default 0.35) - AELP2_MIN_ROAS (default 0.70) Window: last 14 days by default (AELP2_FIDELITY_WINDOW_DAYS)
```

AELP2/scripts/check_env.py

```
#!/usr/bin/env python3 import os import sys REQUIRED = [ 'GOOGLE_CLOUD_PROJECT',
'BIGQUERY_TRAINING_DATASET', 'AELP2_MIN_WIN_RATE', 'AELP2_MAX_CAC',
'AELP2_MIN_ROAS',
```

AELP2/scripts/create_google_ads_search_campaign.py

```
#!/usr/bin/env python3 """ Create a basic Google Ads Search campaign with an Ad
Group, keywords, and a Responsive Search Ad (RSA). Defaults: - Status PAUSED for
safety - Channel SEARCH, Google Search only (no content network) - Optional
campaign-level Final URL Suffix for UTM parameters Env required:
```

AELP2/scripts/e2e_orchestrator.py

```
#!/usr/bin/env python3 """ End-to-End Orchestrator for AELP2 Runs the full stack in
a smart, gated sequence with clear output and actionable diagnostics. Defaults to
dry-run safe modes where applicable. Steps (in order): 1) Preflight ensures + views
2) Great Expectations gate (dry or BQ mode)
```

AELP2/scripts/emergency_stop.py

```
#!/usr/bin/env python3 """ Trigger an emergency stop safety event with a reason and
optional context. Usage: python -m AELP2.scripts.emergency_stop --reason
"manual_stop" --note "Nightly drill" Env: Must have GOOGLE_CLOUD_PROJECT and
BIGQUERY_TRAINING_DATASET set to enable BQ logging via writer. """
```

AELP2/scripts/ga4_event_to_bq_enrollments.py

```
#!/usr/bin/env python3 """ Write GA4 event counts per day for a chosen event into
BigQuery table <project>.<dataset>.ga4_enrollments_daily so reconciliation can use
exact GA4 "Daily Enrollments" even without GA4 export. Env: - GOOGLE_CLOUD_PROJECT,
BIGQUERY_TRAINING_DATASET, GA4_PROPERTY_ID - Auth as in
ga4_find_enrollment_event.py (OAuth refresh token or SA JSON)
```

AELP2/scripts/ga4_events_overview.py

```
#!/usr/bin/env python3 """ List top GA4 events and counts over a lookback window to
help map to pacer metrics. Requires: GA4_PROPERTY_ID and GA4 auth (service account
or OAuth refresh). Usage: export GA4_PROPERTY_ID=properties/XXXX python3
AELP2/scripts/ga4_events_overview.py --days 120 --limit 100 """
```

AELP2/scripts/ga4_export_pacer_to_bq.py

```
#!/usr/bin/env python3 """ Build GA4 → Pacer metrics daily table in BigQuery using
the native GA4 export dataset (events_*). Inputs: - Env: GOOGLE_CLOUD_PROJECT,
BIGQUERY_TRAINING_DATASET - Env: GA4_EXPORT_DATASET (e.g.,
ga360-bigquery-datashare.analytics_308028264). If not set, --export-dataset must be
provided. - Mapping file: AELP2/config/ga4_pacer_mapping.yaml (same shape as
ga4_pacer_to_bq.py) Outputs:
```

AELP2/scripts/ga4_find_enrollment_event.py

```
#!/usr/bin/env python3 """ List top GA4 event names for the configured property so
we can pick the "enrollment" event used in the Daily Enrollments exploration. Env
required: - GA4_PROPERTY_ID = properties/<id> This uses the GA4 Data API. Auth
order: 1) OAuth refresh token (GA4_OAUTH_CLIENT_ID/SECRET/REFRESH_TOKEN)
```

AELP2/scripts/ga4_kpi_discovery.py

#!/usr/bin/env python3 """ GA4 KPI Discovery Discovers GA4 conversion events and summarizes last-N-days metrics to help calibrate the true KPI (e.g., sign_up, generate_lead, purchase). Outputs: - Top GA4 events by conversions with: conversions, event_count, users, event_value, purchase_revenue

AELP2/scripts/ga4_pacer_to_bq.py

#!/usr/bin/env python3 """ Build GA4 → Pacer metrics daily table in BigQuery using GA4 Data API. Inputs: - GA4 property: env GA4_PROPERTY_ID (e.g., properties/123456789) - Auth: service account via GOOGLE_APPLICATION_CREDENTIALS, or OAuth refresh (GA4_OAUTH_REFRESH_TOKEN, GA4_OAUTH_CLIENT_ID, GA4_OAUTH_CLIENT_SECRET) - Mapping file: AELP2/config/ga4_pacer_mapping.yaml Outputs:

AELP2/scripts/ga4_reconcile_ads.py

#!/usr/bin/env python3 """ Create GA4-based KPI views and an Ads↔GA4 reconciliation view in BigQuery. Views created in <project>.<dataset>: - ga4_enrollments_daily (source: GA4 export events if --event and --export-dataset provided; else from ga4_daily) - ads_vs_ga4_kpi_daily (joins ads_kpi_daily with GA4 enrollments; computes ga4_cac) Env required: GOOGLE_CLOUD_PROJECT, BIGQUERY_TRAINING_DATASET Optional env: GA4_EXPORT_DATASET (fallback for --export-dataset)

AELP2/scripts/google_ads_backfill_children.py

#!/usr/bin/env python3 """ Enumerate child Google Ads customer accounts under the MCC and backfill ads_campaign_performance with capacity metrics (impression share and lost-IS/top-IS) for each child. Env required: - GOOGLE_ADS_DEVELOPER_TOKEN, GOOGLE_ADS_CLIENT_ID, GOOGLE_ADS_CLIENT_SECRET, GOOGLE_ADS_REFRESH_TOKEN, GOOGLE_ADS_LOGIN_CUSTOMER_ID (MCC), GOOGLE_ADS_CUSTOMER_ID (MCC) - GOOGLE_CLOUD_PROJECT, BIGQUERY_TRAINING_DATASET

AELP2/scripts/lock_kpi_and_fidelity.py

#!/usr/bin/env python3 """ Lock Ads KPI conversion_action_ids (based on PURCHASE actions with real volume), refresh KPI views, print headroom, and optionally run a stabilization training pass followed by KPI-only fidelity. Usage: GOOGLE_CLOUD_PROJECT=... BIGQUERY_TRAINING_DATASET=... \ python3 -m AELP2.scripts.lock_kpi_and_fidelity [--last_days 30] [--top 5] \ [--run_training] [--episodes 10] [--steps 600] [--budget 8000]

AELP2/scripts/meta_csv_to_bq.py

#!/usr/bin/env python3 """ Load a Meta Ads Manager CSV export into BigQuery (fallback when API access is blocked). Input: CSV exported at ad level with daily breakdown from Ads Manager. Maps common column names to the standard table schema used by meta_to_bq.py: - date (DATE), campaign_id, adset_id, ad_id, impressions, clicks, cost, conversions, revenue, ctr, cvr, avg_cpc, name_hash Usage:

AELP2/scripts/pacing_excel_to_bq.py

#!/usr/bin/env python3 """ Load a multi-sheet pacing Excel workbook to BigQuery (month tabs supported). Maps common columns: - Date → date (YYYY-MM-DD) - Direct Spend/Spend/Cost/Amount → cost (FLOAT) - D2C Total Subscribers (or Post Trial Subscribers) → conversions (FLOAT) Writes to: <project>.<dataset>.pacing_daily

AELP2/scripts/pacing_reconcile_ga4.py

#!/usr/bin/env python3 """ Create views to reconcile internal pacing spend vs GA4 enrollments, with CAC. Creates in <project>.<dataset>: - pacing_vs_ga4_kpi_daily - pacing_vs_ga4_summary (28/45/90d rollups) - pacing_vs_ga4_monthly (month-level tie-out) Prereqs:

AELP2/scripts/pacing_to_bq.py

#!/usr/bin/env python3 """ Load an internal pacing CSV into BigQuery for KPI tie-out. Expected minimal columns (case-insensitive): - date: YYYY-MM-DD (or MM/DD/YYYY) - cost or spend: numeric (can include commas) Optional columns (detected automatically if present): - platform (e.g., google_ads, meta, bing, impact)

AELP2/scripts/preflight.py

#!/usr/bin/env python3 """ Pre-flight environment and tooling checks for AELP2. Performs: - Print detected env vars (redacted), tool versions - Verify/ensure BigQuery dataset and required tables/views - Attempt ingestion connectivity checks (dry/health checks) Env required:

AELP2/scripts/rollback_last_canary.py

#!/usr/bin/env python3 """ Rollback the last applied canary budget changes using the changefeed in `<project>.<dataset>.canary_changes`. Env: GOOGLE_CLOUD_PROJECT, BIGQUERY_TRAINING_DATASET GOOGLE_ADS_DEVELOPER_TOKEN, GOOGLE_ADS_CLIENT_ID, GOOGLE_ADS_CLIENT_SECRET, GOOGLE_ADS_REFRESH_TOKEN, GOOGLE_ADS_CUSTOMER_ID AELP2_ALLOW_GOOGLE_MUTATIONS=1 (required to actually apply rollback)

AELP2/scripts/run_recommendations.py

#!/usr/bin/env python3 """ Run AELP2 end-to-end and print actionable recommendations from BigQuery. What it does - Loads env (.env + optional Gmail Ads mapping) and applies safe defaults - Runs the e2e orchestrator (full mode) with GE gate - Summarizes latest MMM allocations, canary (budget) proposals, and bandit suggestions - Prints next-step hints for going live

AELP2/scripts/seed_auctions_stub.py

#!/usr/bin/env python3 """ Seed synthetic bidding events into BigQuery for the Auctions Monitor. Tables: bidding_events_per_minute, bidding_events """ import os, random from datetime import datetime, timedelta, timezone from google.cloud import bigquery # type: ignore PROJECT = os.getenv('GOOGLE_CLOUD_PROJECT')

AELP2/scripts/seed_bandit_posteriors.py

#!/usr/bin/env python3 """Seed bandit_posteriors from recent ads performance (heuristic). Computes CAC mean and an approximate 95% CI per cell_key (campaign:ad_id) using last 14 days of ads_ad_performance and writes to bandit_posteriors. """ import os, math, time from datetime import date, timedelta from google.cloud import bigquery # type: ignore from datetime import datetime, timezone

AELP2/scripts/snapshot_canary_budgets.py

#!/usr/bin/env python3 """ Snapshot current budgets for a set of Google Ads campaigns into BigQuery table `<project>.<dataset>.canary_budgets_snapshot`. Env: GOOGLE_CLOUD_PROJECT, BIGQUERY_TRAINING_DATASET GOOGLE_ADS_DEVELOPER_TOKEN,

GOOGLE_ADS_CLIENT_ID, GOOGLE_ADS_CLIENT_SECRET, GOOGLE_ADS_REFRESH_TOKEN, GOOGLE_ADS_CUSTOMER_ID (10 digits) AELP2_GOOGLE_CANARY_CAMPAIGN_IDS: comma-separated campaign IDs

AELP2/scripts/training_stub.py

```
#!/usr/bin/env python3 """ Production RL Training System for AELP2 Real reinforcement learning training using PPO/Q-learning with: - RecSim for user simulation - AuctionGym for auction mechanics - Real attribution with delayed rewards - No fallbacks or simplifications
```

AELP2/scripts/update_status.py

```
#!/usr/bin/env python3 import sys import re from datetime import datetime, timezone def mark(path: str, needle: str, status: str): with open(path, 'r', encoding='utf-8') as f: txt = f.read() ts = datetime.utcnow().strftime('%Y-%m-%d %H:%M UTC')
```

AELP2/scripts/validate_no_stubs.py

```
#!/usr/bin/env python3 """ Validation Script: Ensure No Stub/Mock/Fallback Code Remains This script systematically validates that all stub implementations have been replaced with real, production-ready code. It enforces the NO FALLBACKS rule by scanning for violations and verifying that systems actually work. Run this after replacing stub files to ensure compliance with CLAUDE.md requirements. """
```

AELP2/tests/test_bandit_service.py

```
#!/usr/bin/env python3 """ Quick sanity tests for bandit_service. Run as: python3 -m AELP2.tests.test_bandit_service Exits 0 on success, 1 on failure. """ import sys from AELP2.core.optimization.bandit_service import thompson_select, mabwiser_select, HAVE_MABWISER
```

AELP2/tests/test_budget_cac_cap.py

```
#!/usr/bin/env python3 import os import sys import types from unittest import mock def run_with_mocks(): # Prepare env os.environ['GOOGLE_CLOUD_PROJECT'] = 'proj'
```

AELP2/tests/test_gx_checks.py

```
#!/usr/bin/env python3 import sys import subprocess def test_gx_dry_run_pass(): rc = subprocess.run([sys.executable, 'AELP2/ops/gx/run_checks.py', '--dry_run'], check=False).returncode assert rc == 0
```

AELP2/tests/test_journey_paths.py

```
#!/usr/bin/env python3 from AELP2.pipelines.journey_path_summary import compute_paths from datetime import datetime, timedelta def test_compute_paths_basic(): now = datetime.utcnow() rows = [ {'user_id': 'u1', 'session_start': now - timedelta(days=2), 'ch': 'search'}, {'user_id': 'u1', 'session_start': now - timedelta(days=1), 'ch': 'display'},
```

AELP2/tests/test_value_payloads.py

```
#!/usr/bin/env python3 from AELP2.adapters.google_enhanced_conversions import build_enhanced_conversions from AELP2.adapters.meta_capi import build_capi_events def test_google_ec_payload(): rows=[{'email':'User@Example.com','phone':'(555) 123-4567','first_name':'Ann','last_name':'Lee','country':'us','postal_code':'02139','value':123.45,'currency':'USD','conversion_action':'123','order_id':'o1'}]
```

```
out=build_enhanced_conversions(rows) assert out and
out[0]['conversion_action']=='123' and 'user_identifiers' in out[0]
```

AELP2/tools/ad_level_accuracy.py

```
#!/usr/bin/env python3 """ Compute ad-level offline accuracy metrics from
historical data and simulator predictions. Inputs (best-effort; all local/offline):
- AELP2/reports/sim_fidelity_campaigns.json (or *_temporal_v2.json) -
AELP2/reports/creative/*.json (realized outcomes per creative_id) -
AELP2/reports/rl_shadow_score.json (example simulated predictions per campaign)
Outputs:
```

AELP2/tools/ad_level_calibration_v22.py

```
#!/usr/bin/env python3 from __future__ import annotations import os, json, math
from pathlib import Path from typing import List, Dict import numpy as np # Light
ML stack from sklearn.linear_model import LogisticRegression from sklearn.isotonic
import IsotonicRegression
```

AELP2/tools/ad_level_calibration_v23.py

```
#!/usr/bin/env python3 from __future__ import annotations import json, math from
pathlib import Path from typing import List import numpy as np from
sklearn.linear_model import LogisticRegression from sklearn.isotonic import
IsotonicRegression ROOT = Path(__file__).resolve().parents[2]
```

AELP2/tools/ad_level_ranker_v24.py

```
#!/usr/bin/env python3 from __future__ import annotations import json, math from
pathlib import Path from typing import List, Tuple import numpy as np from
sklearn.linear_model import LogisticRegression ROOT =
Path(__file__).resolve().parents[2] CRE_DIR = ROOT / 'AELP2' / 'reports' /
'creative_enriched'
```

AELP2/tools/add_novelty_and_export_rl_pack.py

```
#!/usr/bin/env python3 from __future__ import annotations import json from pathlib
import Path import numpy as np ROOT = Path(__file__).resolve().parents[2] MOD =
ROOT / 'AELP2' / 'models' / 'new_ad_ranker' FEATJL = ROOT / 'AELP2' / 'reports' /
'creative_features' / 'creative_features.jsonl' BLUE = ROOT / 'AELP2' / 'reports' /
'ad_blueprints_top20.json'
```

AELP2/tools/add_phone_look.py

```
#!/usr/bin/env python3 from __future__ import annotations """ Apply a subtle
smartphone look: grain, slight vignette, gentle motion jitter. Usage: python3
AELP2/tools/add_phone_look.py input.mp4 output.mp4 """ import sys, shutil,
subprocess FFMPEG = shutil.which('ffmpeg') or 'ffmpeg'
```

AELP2/tools/annotate_weekly_with_policy.py

```
#!/usr/bin/env python3 from __future__ import annotations """ Join adset/campaign
config into weekly creatives and mark policy compliance. Inputs: -
AELP2/raw/ad_config/adsets.jsonl (from export_meta_ad_configs.py) -
AELP2/reports/weekly_creatives/*.json Outputs:
```

AELP2/tools/assemble_boring_good.py

```
#!/usr/bin/env python3 from __future__ import annotations """ Assemble 3 "boringly good" Aura variants: - Hook: MJ still (normalized) → subtle motion - Proof: proof_clips/lock_card.mp4 (or safe_browsing_block.mp4) - Relief: MJ still → subtle motion - End card: brand demo storyboard - Branded captions + VO optional (kept minimal here to ensure legibility)
```

AELP2/tools/assemble_enforced.py

```
#!/usr/bin/env python3 from __future__ import annotations """ Assemble a 9:16 ad with enforced slot grammar using a manifest + recipe. Inputs: - Manifest JSON with explicit slot assets, e.g. AELP2/manifests/ugc_sample1.json { "recipe": "AELP2/creative/recipes/ugc_pattern_v1.json", "slots": {
```

AELP2/tools/assemble_from_assets.py

```
#!/usr/bin/env python3 from __future__ import annotations import json, subprocess from pathlib import Path ROOT = Path(__file__).resolve().parents[2] AROLL = ROOT / 'AELP2' / 'outputs' / 'a_roll' / 'hooks' PROOF = ROOT / 'AELP2' / 'outputs' / 'proof_images' FIN = ROOT / 'AELP2' / 'outputs' / 'finals' PATKB = ROOT / 'AELP2' / 'competitive' / 'pattern_to_kb.json'
```

AELP2/tools/assemble_original_batch.py

```
#!/usr/bin/env python3 from __future__ import annotations """ Assemble fully original ads from Runway clips and VO packs. Creates 6 variants: 3 scripts × 2 visual mixes. Each ~12.0s: Hook(2.0) → Proof(4.0) → Relief(3.0) → End(3.0) Inputs: - Runway clips: AELP2/outputs/renders/runway/*.mp4
```

AELP2/tools/assemble_pattern.py

```
#!/usr/bin/env python3 from __future__ import annotations """ Assemble a pattern-driven ad using: - Base video (Veo sample_0.mp4) - Overlay UI from ui_overlays.json based on pattern_to_kb - End-card + claim/CTA from pattern_to_kb.json - VO lines already generated Output: AELP2/outputs/finals/<pattern>_final.mp4
```

AELP2/tools/assemble_real_ad.py

```
#!/usr/bin/env python3 from __future__ import annotations """ Assemble a "real" ad using Runway hooks, a proof clip, a relief clip, captions, and ElevenLabs VO. Inputs (auto-picked with sane fallbacks): - Hook: AELP2/outputs/renders/runway/*.mp4 (newest) or AELP2/outputs/a_roll/hooks/*.mp4 - Proof: AELP2/assets/proof_clips/*.mp4 (lock_card preferred) or reuse Hook - Relief: next-best Runway or A-roll - Endcard: created via demo_storyboard.py (04_cta.png)
```

AELP2/tools/assemble_spot_the_tell.py

```
#!/usr/bin/env python3 from __future__ import annotations """ Assemble a 9:16 "Spot The Tell" video: - Hook: 5–7s phone-shot (or placeholder) with three SMS bubbles (overlay) - Proof: Safe Browsing block overlay - End-card Output: AELP2/outputs/finals/spot_the_tell_v1.mp4 """
```

AELP2/tools/assemble_two_screens.py

```
#!/usr/bin/env python3 from __future__ import annotations """ Assemble a 9:16 split-screen "Two Screens, Two Outcomes" video. Inputs (optional): - Left (chaos) and Right (control) hook mp4s under AELP2/outputs/renders/* If none found, uses generated placeholders. - Proof overlay PNG from AELP2/outputs/proof_images/proof_lock_card.png - End-card from
```

AELP2/outputs/endcards/endcard_default.png

AELP2/tools/assemble_with_vo.py

#!/usr/bin/env python3 from __future__ import annotations """ Assemble a Veo/Runway hook with ElevenLabs VO, add end-card and disclaimer overlay, normalize audio. Inputs (auto-detected): - AELP2/outputs/renders/sample_0.mp4 (Veo) or runway/*.mp4 - AELP2/outputs/audio/*identity*.mp3 - AELP2/branding/end_card_spec.json + overlays.json Outputs: AELP2/outputs/finals/*.mp4

AELP2/tools/attention_heuristics.py

#!/usr/bin/env python3 from __future__ import annotations """ First-3s attention heuristics (rough): - assume captions present - estimate scene pace from keyframe interval (ffprobe) - estimate on-screen text presence (OCR skipped -> heuristic = captions=True) Outputs: AELP2/reports/attention_scores.json """ import json, subprocess

AELP2/tools/backfill_ad_daily_insights.py

#!/usr/bin/env python3 from __future__ import annotations """ Backfill ad-level daily insights in small date chunks with checkpointing. Writes JSONL shards per chunk under AELP2/raw/ad_daily/YYYYMMDD_YYYYMMDD.jsonl Fields: ad_id, adset_id, campaign_id, date_start, impressions, clicks, spend, frequency, actions. """ import os, json, time from datetime import date, timedelta from pathlib import Path

AELP2/tools/bigspy_auto_export.py

#!/usr/bin/env python3 from __future__ import annotations """ Headless BigSpy exporter (best-effort, cookie-based login reuse) What it does - Reuses your existing BigSpy session via cookies (Netscape cookies.txt or JSON) - Opens BigSpy in a headless Chromium, optionally applies basic filters - Scrolls results and scrapes core creative metadata - Writes a CSV to AELP2/vendor_imports/bigspy_export_YYYYMMDD_HHMM.csv

AELP2/tools/build_brand_pack.py

#!/usr/bin/env python3 from __future__ import annotations """ Build a minimal brand pack from existing ads + site: - Reads Meta creative_objects/* for domain(s), CTAs, titles - Reads KB products for CTAs and disclaimers - Fetches brand site (aura.com) pages + CSS to extract colors and fonts Outputs: - AELP2/branding/brand_pack.json

AELP2/tools/build_cool_variant.py

#!/usr/bin/env python3 from __future__ import annotations """ Build a polished preview ad from your brand + MJ assets without waiting on remote video models. Steps: 1) Pick 3 normalized stills under AELP2/assets/backplates/normalized/ 2) Animate each with a subtle Ken Burns pan/zoom locally via ffmpeg (2.0s / 3.5s / 3.0s) 3) Concatenate + append a brand end-card (demo storyboard) 4) Overlay a timed quiz band (brand colors/fonts)

AELP2/tools/build_features_from_creative_objects.py

#!/usr/bin/env python3 from __future__ import annotations """ Build lightweight numeric features for creatives using the cached Meta `creative_objects` JSON files. Outputs - `AELP2/reports/creative_features/creative_features.jsonl` - `AELP2/reports/creative_features/index.json` (creative_id -> file path)

AELP2/tools/build_labels_weekly.py

```
#!/usr/bin/env python3 from __future__ import annotations """ Construct pairwise training data (diff features, label win vs baseline) from weekly_creatives + creative_features. Outputs AELP2/reports/new_ranker/pairs.jsonl (fields: cid, week, x[], y, placement) AELP2/reports/new_ranker/feature_map.json (name -> index) """
```

AELP2/tools/build_placement_calibrators.py

```
#!/usr/bin/env python3 from __future__ import annotations import json from pathlib import Path import math ROOT = Path(__file__).resolve().parents[2] INP = ROOT / 'AELP2' / 'reports' / 'placement_conversions' OUT = ROOT / 'AELP2' / 'reports'
```

AELP2/tools/build_proof_assets.py

```
#!/usr/bin/env python3 from __future__ import annotations """ Build simple proof overlay PNGs and an end-card image from brand pack. Outputs under AELP2/outputs/proof_images and AELP2/outputs/endcards. """ import json from pathlib import Path from PIL import Image, ImageDraw, ImageFont
```

AELP2/tools/build_system_overview_pdf.py

```
#!/usr/bin/env python3 from __future__ import annotations """ Build a comprehensive PDF (for non-coders) that explains AELP/AELP2: - Problem statement and solution overview - Architecture (ASCII flow charts) - Data pipelines, models, RL, dashboards, APIs - Vendors/importers and forecasting methodology - Accuracy/coverage snapshots from reports/ - How to run + ops
```

AELP2/tools/build_topk_from_scores.py

```
#!/usr/bin/env python3 from __future__ import annotations import json from pathlib import Path ROOT = Path(__file__).resolve().parents[2] SCORES = ROOT / 'AELP2' / 'reports' / 'vendor_scores.json' COBJ = ROOT / 'AELP2' / 'reports' / 'creative_objects' OUT = ROOT / 'AELP2' / 'reports' / 'vendor_top20.json'
```

AELP2/tools/build_weekly_predictions.py

```
#!/usr/bin/env python3 from __future__ import annotations """ Build weekly (calendar) predictions and actuals per creative from ad-level daily insights. For each campaign and ISO week t, use prior 2 weeks (t-2,t-1) to estimate a CVR prior and predict purchases_t = clicks_t * E[cvr_prior]. Also compute actual purchases_t and CAC_t. Exports per-campaign weekly JSON files for WBUA evaluation. Outputs: AELP2/reports/weekly_creatives/<campaign_id>_<iso_year>W<iso_week>.json """ import json, math from pathlib import Path
```

AELP2/tools/cascade_dr_topk.py

```
#!/usr/bin/env python3 from __future__ import annotations """ Cascade-DR style estimator for a Top-K slate policy. Simplified examination model: prob(examine rank r) = eta^(r-1), eta estimated from CTR drop-off. """ import json, os from pathlib import Path import numpy as np
```

AELP2/tools/check_dual_gate_thresholds.py

```
#!/usr/bin/env python3 from __future__ import annotations """ Check dual-gate precision and yield against thresholds and write a status file. Inputs: AELP2/reports/dual_gate_weekly.json Env: - AELP2_DUAL_MIN_PREC (default 0.25) - AELP2_DUAL_MIN_YIELD (default 0.01)
```

AELP2/tools/collect_kb_overlays.py

```
#!/usr/bin/env python3 from __future__ import annotations """ Collect claim overlays (text + mandatory disclaimer) and a default CTA from KB for use in end-cards. Writes AELP2/branding/overlays.json """ import json from pathlib import Path ROOT = Path(__file__).resolve().parents[2]
```

AELP2/tools/compute_aesthetic_metrics.py

```
#!/usr/bin/env python3 from __future__ import annotations """ Compute lightweight aesthetic/quality proxies (no-ref): - Laplacian variance (sharpness) - Tenengrad (Sobel) focus measure - Colorfulness (Hasler-Süsstrunk) Outputs: AELP2/reports/aesthetic_features.jsonl """ import json
```

AELP2/tools/compute_locked_targets.py

```
#!/usr/bin/env python3 from __future__ import annotations """ Compute locked target_CAC per campaign from recent weekly files to stabilize gates for the first policy-compliant period. Logic: - Read weekly files (policy dir if present), take the latest N weeks (default 2). - For each campaign, compute the median CAC across those weeks (per-week medians, then median-of-medians). - Write AELP2/reports/target_cac_locked.json: {campaign_id: target_cac}
```

AELP2/tools/compute_mmm_bands.py

```
#!/usr/bin/env python3 from __future__ import annotations """ Compute simple MMM-like daily spend bands (min/base/max) from recent spend and returns using a concave response proxy. Outputs: AELP2/reports/mmm_bands.json """ import json from pathlib import Path import numpy as np
```

AELP2/tools/compute_motion_features.py

```
#!/usr/bin/env python3 from __future__ import annotations """ Compute simple motion features from finals using OpenCV optical flow. Outputs: AELP2/reports/motion_features.jsonl """ import subprocess, json from pathlib import Path import cv2 import numpy as np
```

AELP2/tools/compute_target_cac_frontier.py

```
#!/usr/bin/env python3 from __future__ import annotations """ Compute per-campaign target_CAC by scanning historical CAC percentiles and choosing the point that maximizes utility U = purchases - spend/target_CAC on held-out days (proxy frontier). Outputs: AELP2/reports/target_cac.json """ import json from pathlib import Path import numpy as np
```

AELP2/tools/compute_us_meta_baselines.py

```
#!/usr/bin/env python3 from __future__ import annotations """ Compute US Meta baselines from BigQuery and write to reports/us_meta_baselines.json. Assumes insights were ingested to <project>.<dataset>.meta_ad_performance via AELP2/pipelines/meta_to_bq.py. If geo fields are absent, we treat the account as US■focused. Env required: GOOGLE_CLOUD_PROJECT
```

AELP2/tools/compute_us_paid_baselines_by_place.py

```
#!/usr/bin/env python3 from __future__ import annotations """ Compute US Meta paid-event baselines by publisher_platform and placement from `<project>.<dataset>.meta_ad_performance_by_place`. Writes: AELP2/reports/us_meta_baselines_by_place.json Fields per key
```

"{platform}/{placement}": cpm_p10/p50/p90, ctr_p10/p50/p90, cvr_p10/p50/p90, and totals (imps, clicks, cost, conversions).

AELP2/tools/conformal_topk.py

#!/usr/bin/env python3 from __future__ import annotations import json from pathlib import Path import numpy as np ROOT = Path(__file__).resolve().parents[2] import os CRE_DIR = Path(os.getenv('AELP2_CREATIVE_DIR') or (ROOT / 'AELP2' / 'reports' / 'creative_enriched')) OUT = ROOT / 'AELP2' / 'reports'

AELP2/tools/conformal_topk_weekly.py

#!/usr/bin/env python3 from __future__ import annotations """ Selection-conditional conformal bounds for weekly creatives. Method (simple): - For each campaign, split its weekly files into calibration (all but last 2 weeks) and evaluation (last 2 weeks). - For calibration, compute residuals r = actual_purchases - sim_score for items that would be selected by a simple selection rule: top-K by sim_score (K from env, default 5). - Let q = quantile_{1-alpha} of |r| (alpha from env, default 0.1 → 90% lower bound).

AELP2/tools/crawl_youtube_titles.py

#!/usr/bin/env python3 from __future__ import annotations """ Lightweight crawler to fetch video titles/descriptions from public YouTube channel pages without API keys. Provide comma-separated channel URLs in YT_CHANNELS env, e.g.: export YT_CHANNELS="https://www.youtube.com/@Aura,https://www.youtube.com/@AuraPrivacy" Writes AELP2/reports/youtube_copy.json """ import os, re, json from pathlib import Path

AELP2/tools/demo_storyboard.py

#!/usr/bin/env python3 from __future__ import annotations """ Generate a 4-frame 9:16 storyboard from the Identity KB: 1) Hook (non-claim benefit) 2) Proof (generic) 3) Claim + mandatory disclaimer 4) End card + CTA Outputs PNGs to AELP2/outputs/demo_ads/01_hook.png .. 04_cta.png.

AELP2/tools/detect_objects_yolo.py

#!/usr/bin/env python3 from __future__ import annotations """ Detect objects on mid-frames using YOLOv8n (CPU). Reports counts of person and cell phone. Outputs: AELP2/reports/object_counts.jsonl """ import json from pathlib import Path from PIL import Image import numpy as np

AELP2/tools/download_runway_results.py

#!/usr/bin/env python3 from __future__ import annotations import json, sys from pathlib import Path import requests ROOT = Path(__file__).resolve().parents[2] LOG = ROOT / 'AELP2' / 'reports' / 'runway_tasks.json' OUTDIR = ROOT / 'AELP2' / 'outputs' / 'renders' / 'runway'

AELP2/tools/dual_gate_weekly.py

#!/usr/bin/env python3 from __future__ import annotations """ Dual-gate weekly evaluator. Gate a variant in week t if ALL hold: 1) predicted CAC <= target_CAC * (1 - delta) 2) predicted purchases >= min_volume 3) sim_score_variant > sim_score_baseline_slate (baseline chosen from previous up-to-4 weeks by max utility)

AELP2/tools/eleven_design_voice.py

```
#!/usr/bin/env python3 from __future__ import annotations """ Design and create an ElevenLabs voice from a natural-language prompt, then store it in our library. Inputs via CLI args or env: --prompt "An earnest voice ..." --name "Earnest Parent (UK)" Env: ELEVENLABS_API_KEY
```

AELP2/tools/enrich_creatives_with_objects.py

```
#!/usr/bin/env python3 from __future__ import annotations import json, re, math from pathlib import Path from datetime import datetime import re ROOT = Path(__file__).resolve().parents[2] BASIC = ROOT / 'AELP2' / 'reports' / 'creative_with_dna' if not BASIC.exists():
```

AELP2/tools/enrich_weekly_with_cpc.py

```
#!/usr/bin/env python3 from __future__ import annotations """ Enrich weekly creatives with a CPC-mix based predicted CAC. For each campaign-week file: - Compute a campaign CVR prior from the previous W weeks (default 2): e_cvr_prev = sum(purchases)/sum(clicks). - Load per-campaign creative aggregates from AELP2/reports/creative/<campaign_id>.json and compute cpc_hat for each creative as spend/clicks from placement_stats or `test_cpc` if available. - For each item in the week file, set pred_cac_cpc = cpc_hat / max(e_cvr_prev, eps).
```

AELP2/tools/estimate_policy_uplift.py

```
#!/usr/bin/env python3 from __future__ import annotations """ Estimate coarse uplift scalars for key policy switches by comparing compliant-like vs noncompliant items within campaign-weeks. Outputs: AELP2/reports/policy_uplift.json with ratios for CVR (purchases/clicks) and CAC. """ import json, math from pathlib import Path from statistics import median
```

AELP2/tools/eval_ablation.py

```
#!/usr/bin/env python3 from __future__ import annotations import json from pathlib import Path ROOT = Path(__file__).resolve().parents[2] R = ROOT / 'AELP2' / 'reports' def loadp(p: Path): try:
```

AELP2/tools/eval_wbua.py

```
#!/usr/bin/env python3 from __future__ import annotations """ Evaluate Weekly Baseline Uplift Accuracy (WBUA) on weekly_creatives/* files. For each campaign-week, choose baseline = best ad by previous week actual utility (purchases - spend/target_CAC), then measure fraction of variants where (sim_score_variant > sim_score_baseline) matches (actual_variant beats baseline at equal/lower CAC). Outputs: AELP2/reports/wbua_summary.json """ import json, math, os from pathlib import Path
```

AELP2/tools/eval_wbua_cluster.py

```
#!/usr/bin/env python3 from __future__ import annotations """ Cluster-holdout WBUA: treat creative "families" as clusters using ad name heuristics from creative_objects. Evaluate only items in week t whose family did not appear in weeks t-4..t-1 for the same campaign. Outputs: AELP2/reports/wbua_cluster_summary.json """ import json, math, re, random
```

AELP2/tools/eval_wbua_forward.py

```
#!/usr/bin/env python3 from __future__ import annotations """ Forward-holdout WBUA: freeze decisions using only history up to week t-1 and evaluate on week t. Produces accuracy and bootstrap 95% CI across campaign-weeks. Outputs: AELP2/reports/wbua_forward.json """ import json, math, re, random from pathlib import Path
```

AELP2/tools/eval_wbua_novel.py

```
#!/usr/bin/env python3 from __future__ import annotations """ Compute WBUA (Weekly Baseline Uplift Accuracy) for novel-only creatives. Definition For each campaign-week t, pick the best historical baseline using weeks t-4..t-1 (utility aggregation over those weeks). For week t items, mark a variant as a "win vs baseline" if it has >= actual_score and <= actual_cac relative to the baseline. Count whether the simulator's sim_score ranking agrees with that
```

AELP2/tools/export_meta_ad_configs.py

```
#!/usr/bin/env python3 from __future__ import annotations """ Export current campaign and adset configuration from Meta Graph API. Writes JSONL snapshots under AELP2/raw/ad_config/{campaigns,adsets}.jsonl Fields captured (best-effort): - Campaign: id, name, objective, bid_strategy, status, configured_status, special_ad_categories, created_time, updated_time - Adset: id, name, campaign_id, optimization_goal, bid_strategy, is_dynamic_creative, targeting, daily_budget, effective_status, created_time, updated_time
```

AELP2/tools/export_meta_creative_outcomes.py

```
#!/usr/bin/env python3 """ Export historical per-ad outcomes from Meta (read-only) and build files for ad-level accuracy. For each campaign, we compute: - Aggregated actuals per ad_id over the last 7 test days (clicks, spend, purchases, CAC) - A simple per-ad simulated score using a campaign-level CVR prior estimated from the preceding 14 train days Outputs one file per campaign: AELP2/reports/creative/<campaign_id>.json with structure:
```

AELP2/tools/export_ui_overlays_stub.py

```
#!/usr/bin/env python3 from __future__ import annotations """ Stub exporter for UI overlays: create timing JSON entries for proof inserts. In practice, we will replace with real iOS/Android screen records. Outputs: AELP2/competitive/ui_overlays.json """ import json from pathlib import Path
```

AELP2/tools/extract_audio_features.py

```
#!/usr/bin/env python3 from __future__ import annotations """ Extract simple audio features via ffprobe: - duration, avg bitrate, channels - LUFS placeholder (not computed here) Writes AELP2/reports/audio_features.json """ import json, subprocess from pathlib import Path
```

AELP2/tools/extract_audio_lufs.py

```
#!/usr/bin/env python3 from __future__ import annotations """ Extract LUFS using ffmpeg loudnorm analyze mode for finals. Outputs: AELP2/reports/audio_lufs.jsonl """ import json, subprocess from pathlib import Path
ROOT=Path(__file__).resolve().parents[2]
```

AELP2/tools/extract_keyframes.py

```
#!/usr/bin/env python3 from __future__ import annotations """ Extract simple keyframes for local MP4s (finals) to AELP2/reports/keyframes/<stem>/tXXXX.jpg. """
```

```
import subprocess, os from pathlib import Path
ROOT=Path(__file__).resolve().parents[2] FIN=ROOT/'AELP2'/'outputs'/'finals'
```

AELP2/tools/extract_legibility_features.py

```
#!/usr/bin/env python3 from __future__ import annotations """ Compute legibility
proxy per final by reusing self_judge (contrast of top band). Outputs:
AELP2/reports/legibility_features.jsonl """ import json, subprocess from pathlib
import Path ROOT=Path(__file__).resolve().parents[2]
```

AELP2/tools/extract_visual_embeddings.py

```
#!/usr/bin/env python3 from __future__ import annotations """ Compute OpenCLIP
embeddings for keyframes under AELP2/reports/keyframes/* and save a video-level
mean embedding per stem. Outputs:
AELP2/reports/creative_visual_embeddings/<stem>.npy
AELP2/reports/creative_visual_embeddings/index.json """ import json
```

AELP2/tools/fetch_campaign_placement_conversions.py

```
#!/usr/bin/env python3 from __future__ import annotations import json, os, time
from pathlib import Path import requests ROOT = Path(__file__).resolve().parents[2]
OUT = ROOT / 'AELP2' / 'reports' / 'placement_conversions' OUT.mkdir(parents=True,
exist_ok=True) META_BASE = 'https://graph.facebook.com/v23.0'
```

AELP2/tools/fetch_meta_adlibrary.py

```
#!/usr/bin/env python3 from __future__ import annotations """ Meta Ad Library
fetcher - Reads `competitive/brand_scope.json` for search terms. - Calls the Meta
Ad Library API (ads_archive) when an access token is available. - Falls back to
cached Aura creative objects when the API is not accessible. Environment variables:
```

AELP2/tools/fetch_meta_creatives.py

```
#!/usr/bin/env python3 """ Fetch Meta Ad and AdCreative objects for historical ads
listed in AELP2/reports/creative/*.json. Outputs one JSON per ad under
AELP2/reports/creative_objects/<ad_id>.json including: - ad: id, name,
created_time, effective_status, creative{id} - creative: id, object_story_spec,
asset_feed_spec, body/title/link_url if present Offline-only: read-only Graph
calls. """ from __future__ import annotations
```

AELP2/tools/fetch_searchapi_meta.py

```
#!/usr/bin/env python3 from __future__ import annotations """ Fetch Meta Ad Library
results via SearchAPI.io (self-serve) and write CSVs that our vendor importer
understands. Then you can run the normal pipeline to build features and score
creatives. Env: SEARCHAPI_API_KEY (required)
```

AELP2/tools/fidelity_eval_roll.py

```
#!/usr/bin/env python3 """ Rolling-origin evaluation for the temporal (Phase 2)
simulator. Splits within the last 28 days: - 14→7, 21→7, and 7→7 (if enough days)
Outputs JSON summary: - AELP2/reports/sim_fidelity_roll.json """
```

AELP2/tools/finalize_gate.py

```
#!/usr/bin/env python3 from __future__ import annotations """ Finalize a candidate
video with hard gates so low-quality outputs never publish. Checks: - QC gates
(aspect/text/blur/loudness) via qc_gates.py --role final - Self-judge scores via
```

self_judge.py (Interestingness, Relevance) Thresholds (defaults, override via CLI):

AELP2/tools/fit_cac_calibrator.py

#!/usr/bin/env python3 from __future__ import annotations """ Fit a monotonic calibrator mapping predicted CAC to actual CAC using isotonic regression. Inputs: - AELP2/reports/weekly_creatives_cpc/*.json (pred_cac_cpc preferred), else use weekly_creatives/*.json with spend/sim proxy. Outputs: - AELP2/reports/cac_calibrator.json with fields:

AELP2/tools/forecast_us_cac_volume.py

#!/usr/bin/env python3 from __future__ import annotations """ Monte Carlo CAC/volume forecast for Top-20 blueprints at $30k and $50k budgets. Inputs: - us_meta_baselines.json (CPM/CTR/CVR p10/p50/p90) - ad_blueprints_top20.json (p_win, lcb) Outputs:

AELP2/tools/forward_forecast.py

#!/usr/bin/env python3 """ Forward forecast check using Phase 2 (temporal) model. Train: first 21 days of last-28 window; Forecast: next 7 days (holdout) Outputs JSON: AELP2/reports/sim_forward_forecast.json """ from __future__ import annotations import json

AELP2/tools/gen_eleven_vo.py

#!/usr/bin/env python3 from __future__ import annotations """ Generate VO lines with ElevenLabs v3 and save MP3s. Env: ELEVENLABS_API_KEY Usage: python3 AELP2/tools/gen_eleven_vo.py Writes to AELP2/outputs/audio/ """ import os, json

AELP2/tools/gen_eleven_vo_pack.py

#!/usr/bin/env python3 from __future__ import annotations """ Generate a pack of VO lines from AELP2/prompts/vo_scripts.json using ElevenLabs. Writes MP3s to AELP2/outputs/audio/<script_id>/hook.mp3|proof.mp3|cta.mp3 Env: ELEVENLABS_API_KEY """ import os, json from pathlib import Path

AELP2/tools/gen_runway_hooks.py

#!/usr/bin/env python3 from __future__ import annotations """ Generate 9:16 hook clips on Runway (Gen■4 Turbo) using image_to_video. Reads prompts from AELP2/prompts/runway_prompts.json. Creates a small 720x1280 seed PNG (neutral) as promptImage (data URI). Submits tasks, polls until SUCCEEDED/FAILED, downloads MP4s to: AELP2/outputs/renders/runway/<id>.mp4

AELP2/tools/gen_veo_hooks.py

#!/usr/bin/env python3 from __future__ import annotations """ Generate 9:16 hook clips on Vertex AI (Veo 3 Fast) and write to GCS. Reads prompts from AELP2/prompts/veo_prompts.json. For each prompt, calls predictLongRunning and polls the operation until done. Env: GOOGLE_CLOUD_PROJECT, CREATIVE_GCS_BUCKET """

AELP2/tools/generate_advantage_manifest.py

#!/usr/bin/env python3 from __future__ import annotations """ Create a simple Advantage+ Creative manifest from finals. Writes AELP2/outputs/advantage_manifest.json """ import json from pathlib import Path ROOT = Path(__file__).resolve().parents[2]

AELP2/tools/generate_balance_blueprints.py

```
#!/usr/bin/env python3 from __future__ import annotations """ Generate
Balance-focused creative blueprints from Aura parental-controls concepts and
produce a Top-20 list + forecasts + RL pack (separate from the core security set).
""" import json, itertools, subprocess, sys from pathlib import Path ROOT =
Path(__file__).resolve().parents[2]
```

AELP2/tools/generate_blueprints.py

```
#!/usr/bin/env python3 from __future__ import annotations """ Generate synthetic
creative blueprints (visual + copy/offer/CTA/placement) as Meta-like
creative_objects and score them with the trained ranker. Outputs: -
AELP2/reports/ad_blueprints_top20.json Notes:
```

AELP2/tools/generate_score_loop.py

```
#!/usr/bin/env python3 from __future__ import annotations """ Orchestrate generate
-> feature -> score -> filter loop (stub). Currently: - Scores existing finals via
score_new_ads.py - Selects Top-K by p_win with LCB>0, writes slate Outputs:
AELP2/reports/topk_slate.json """
```

AELP2/tools/hourly_multipliers.py

```
#!/usr/bin/env python3 """ Fetch account-level hourly CVR multipliers (normalized
to 1.0 mean). Writes JSON: AELP2/reports/hourly_multipliers.json Note: Useful for
future hourly simulators; day-level sim remains neutral. """ from __future__ import
annotations import os, json from datetime import date, timedelta import numpy as np
```

AELP2/tools/import_proof_clips.py

```
#!/usr/bin/env python3 from __future__ import annotations """ Import real screen
recordings and cut proof clips for slot B. Inputs: - AELP2/assets/proof_raw/*.mp4
(full screen recordings) - AELP2/assets/proof_manifest.json (maps named clips to
in/out times) Outputs:
```

AELP2/tools/import_vendor_meta_creatives.py

```
#!/usr/bin/env python3 from __future__ import annotations """ Normalize vendor
exports (BigSpy/PowerAdSpy/SocialPeta) into GAELP creative_objects. Input: CSV/JSON
files in AELP2/vendor_imports Output:
AELP2/reports/creative_objects/vendor_<source>_<id>.json We only populate the
fields required by build_features_from_creative_objects.py: creative.asset_feed_spe
c.{titles,bodies,link_urls,call_to_action_types,videos,asset_customization_rules}
```

AELP2/tools/join_dna_to_creatives.py

```
#!/usr/bin/env python3 """ Join heuristic CreativeDNA-style features to historical
ad items using ad_name and placement mix. Input: AELP2/reports/creative/*.json
Output: AELP2/reports/creative_with_dna/*.json """ from __future__ import
annotations import json, re from pathlib import Path
```

AELP2/tools/join_finals_features.py

```
#!/usr/bin/env python3 from __future__ import annotations """ Join per-final
features into one JSONL for analysis. Inputs: motion_features.jsonl,
aesthetic_features.jsonl, legibility_features.jsonl, object_counts.jsonl,
audio_lufs.jsonl Output: AELP2/reports/creative_enriched/finals_features.jsonl """
import json from pathlib import Path
```

AELP2/tools/journey_states_from_bq.py

```
#!/usr/bin/env python3 """ Optional: Pull journey states from BigQuery if available. Reads env: GOOGLE_CLOUD_PROJECT, BIGQUERY_USERS_DATASET Looks for tables: user_journeys, journey_touchpoints, conversion_events Writes a tiny sample JSON with row counts to AELP2/reports/journey_states_probe.json """ from __future__ import annotations import os, json from google.cloud import bigquery
```

AELP2/tools/log_creative_dna.py

```
#!/usr/bin/env python3 from __future__ import annotations """ Write CreativeDNA + cost placeholders for produced finals; load to BigQuery if configured. """ import json, os, time from pathlib import Path ROOT = Path(__file__).resolve().parents[2] FIN = ROOT / 'AELP2' / 'outputs' / 'finals'
```

AELP2/tools/make_meta_account_audit.py

```
#!/usr/bin/env python3 """ Generate a Macro Account Audit checklist for Meta (offline). Reads recent simulator fidelity (if present) to suggest priority areas. Writes AELP2/reports/meta_account_audit.md. """ from __future__ import annotations import json from pathlib import Path
```

AELP2/tools/merge_copy_banks.py

```
#!/usr/bin/env python3 from __future__ import annotations """ Merge copy from Meta (copy_bank.json), Google Ads (google_ads_copy.json), Impact (impact_copy.json), and YouTube (youtube_copy.json). Outputs AELP2/reports/copy_bank_merged.json with deduped lines and simple tags per source. """ import json, re from pathlib import Path ROOT = Path(__file__).resolve().parents[2]
```

AELP2/tools/meta_campaign_bayes.py

```
#!/usr/bin/env python3 """ Bayesian per-campaign MMM (Poisson with geometric adstock) for Meta campaigns. Inputs: - META_ACCESS_TOKEN, META_ACCOUNT_ID in environment or .env (export VAR=... lines) Outputs (stdout JSON): { "created_at": "...",
```

AELP2/tools/mine_meta_copy.py

```
#!/usr/bin/env python3 from __future__ import annotations import json, re from pathlib import Path from collections import Counter ROOT = Path(__file__).resolve().parents[2] COBJ = ROOT / 'AELP2' / 'reports' / 'creative_objects' OUT = ROOT / 'AELP2' / 'reports' / 'copy_bank.json'
```

AELP2/tools/mj_ingest.py

```
#!/usr/bin/env python3 from __future__ import annotations """ Ingest Midjourney backplates from a folder, normalize to 9:16, lightly denoise, write a manifest, and optionally animate via Runway image_to_video. Usage: python3 AELP2/tools/mj_ingest.py --src AELP2/assets/backplates/mj/2025-09-24 --animate Outputs:
```

AELP2/tools/normalize_audio.py

```
#!/usr/bin/env python3 from __future__ import annotations """ Normalize audio to -14 LUFS using ffmpeg loudnorm. If no audio, adds a silent track. Usage: python3 AELP2/tools/normalize_audio.py in.mp4 out.mp4 """ import sys, shutil, subprocess, json, tempfile, os FFMPEG = shutil.which('ffmpeg') or 'ffmpeg' def has_audio(src:
```

str) -> bool:

AELP2/tools/offline_creative_search.py

#!/usr/bin/env python3 """ Offline creative generation + simulator scoring scaffold. This tool: 1) Enumerates product × message × format hypotheses 2) Generates CreativeDNA JSON specs (copy-first; assets referenced) 3) Writes candidates to AELP2/outputs/creative_candidates/ 4) (Optional) Calls a simulator scoring hook to produce sim_score and CIs

AELP2/tools/offline_rl_stub.py

#!/usr/bin/env python3 from __future__ import annotations """ Offline RL stub (bandit/IQL-lite): trains a logistic policy to predict P(win) from features and applies a conservative penalty. Outputs: AELP2/reports/offline_policy.json """ import json from pathlib import Path import numpy as np from sklearn.linear_model import LogisticRegression

AELP2/tools/ope_config_weighting.py

#!/usr/bin/env python3 from __future__ import annotations """ OPE with config-based propensity reweighting (diagnostic). Reads weekly_creatives_policy/*.json and computes a Top-K utility estimate where each item is reweighted by its policy compliance propensity (approximate): p_conf = (policy_score + eps). Outputs: AELP2/reports/ope_config_weighted.json """

AELP2/tools/ope_topk.py

#!/usr/bin/env python3 from __future__ import annotations import json from pathlib import Path import numpy as np ROOT = Path(__file__).resolve().parents[2] CRE_DIR = ROOT / 'AELP2' / 'reports' / 'creative_enriched' OUT = ROOT / 'AELP2' / 'reports'

AELP2/tools/ope_upgrades.py

#!/usr/bin/env python3 from __future__ import annotations """ OPE upgrades: DM, IPS, SNIPS, DR, SWITCH, CAB; plus simple bandit-FQE. Inputs: creative_enriched/*.json with fields: sim_score, actual_score (purchases), test_spend, placement_stats. We evaluate a Top-K policy that ranks by sim_score (or a provided score) and estimate expected utility U = purchases - spend/target_CAC per campaign using multiple OPE estimators. Outputs: AELP2/reports/ope_upgrades_topk.json

AELP2/tools/ope_uplift_topk.py

#!/usr/bin/env python3 from __future__ import annotations import json, math from pathlib import Path import numpy as np ROOT = Path(__file__).resolve().parents[2] ENR = ROOT / 'AELP2' / 'reports' / 'creative_enriched' OUT = ROOT / 'AELP2' / 'reports'

AELP2/tools/parse_brand_guide.py

#!/usr/bin/env python3 from __future__ import annotations """ Parse brand guide PDFs to extract a usable brand_config.json for the creative engine. Heuristics: - Extract text via `pdftotext` if available; otherwise fall back to a naive binary scan. - Find hex colors (#RRGGBB) and frequent RGB triplets; dedupe and keep top 8. - Detect font family names by matching common tokens near words like Typography/Font/Typeface. - Save AELP2/creative/brand_config.json with palette, primary/secondary, CTA defaults,

AELP2/tools/policy_audit_live.py

```
#!/usr/bin/env python3 from __future__ import annotations """ Policy audit of current live adsets/campaigns vs META_POLICY_SETUP.md Outputs: AELP2/reports/policy_audit.json with counts per flag and list of non-compliant adsets. """ import json from pathlib import Path ROOT = Path(__file__).resolve().parents[2]
```

AELP2/tools/portfolio_selector.py

```
#!/usr/bin/env python3 from __future__ import annotations """ Portfolio selector: choose a per-campaign slate that maximizes predicted purchases under CAC caps and test budget share. Inputs - weekly_creatives_cpc/*.json (preferred) or weekly_creatives/*.json - target_cac_locked.json (if present) - cac_calibrator.json (optional, applied if present)
```

AELP2/tools/propose_kb_non_claims.py

```
#!/usr/bin/env python3 from __future__ import annotations """ Propose non-claim benefit lines for each product KB from merged copy bank (Meta/Google/Impact/YouTube). Writes AELP2/reports/kb_non_claim_suggestions.json for HITL review. """ import json, re from pathlib import Path ROOT = Path(__file__).resolve().parents[2]
```

AELP2/tools/pull_google_ads_copy.py

```
#!/usr/bin/env python3 from __future__ import annotations """ Pull Google Ads headlines/descriptions via the Google Ads API (read-only) using env credentials. Writes AELP2/reports/google_ads_copy.json with top texts and counts. Env required: - GOOGLE_ADS_DEVELOPER_TOKEN - GOOGLE_ADS_CLIENT_ID - GOOGLE_ADS_CLIENT_SECRET
```

AELP2/tools/pull_google_ads_copy_from_bq.py

```
#!/usr/bin/env python3 from __future__ import annotations """ Pull Google Ads ad names/headlines from BigQuery (preferred over API on this box). Tables used (if present): - `<project>.<dataset>.ads_ad_performance` → uses `ad_name` when not redacted - `<project>.<dataset>.ads_assets` → uses `text` for asset records (if ingested) Writes AELP2/reports/google_ads_copy.json with fields: {headlines:[], descriptions:[]}
```

AELP2/tools/pull_google_ads_copy_rest.py

```
#!/usr/bin/env python3 from __future__ import annotations """ Pull Google Ads headlines/descriptions via REST (searchStream) using OAuth refresh token. Avoids gRPC compatibility issues. Env required: - GOOGLE_ADS_DEVELOPER_TOKEN - GOOGLE_ADS_CLIENT_ID - GOOGLE_ADS_CLIENT_SECRET
```

AELP2/tools/pull_impact_copy.py

```
#!/usr/bin/env python3 from __future__ import annotations """ Pull top affiliate (Impact) copy snippets via Impact API. Writes AELP2/reports/impact_copy.json Env: - IMPACT_ACCOUNT_SID - IMPACT_AUTH_TOKEN (basic auth)
```

AELP2/tools/pull_impact_copy_from_bq.py

```
#!/usr/bin/env python3 from __future__ import annotations """ Pull affiliate (Impact) copy-like fields from BigQuery tables populated by AELP2 pipelines. Tables considered (if exist): - impact_ads (entity dump) - impact_media_partners (entity dump) - impact_partner_performance (report rows; name fields only)
```

AELP2/tools/qc_gates.py

```
#!/usr/bin/env python3 from __future__ import annotations """ Hard QC gates for candidate clips and finals. Gates implemented (lightweight): - resolution/aspect check - blur check (Laplacian variance) - basic text presence heuristic via MSER (if OpenCV available) - loudness check using ffmpeg loudnorm (for files with audio)
```

AELP2/tools/render_bayes_report.py

```
#!/usr/bin/env python3 import os, json, math, csv, datetime from pathlib import Path from typing import Dict, List, Optional, Tuple REPORT_DIR = Path('AELP2/reports') JSON_PATH = Path('/tmp/meta_campaign_bayes_out.json') CSV_PATH = REPORT_DIR / 'meta_bayes_summary.csv' PDF_PATH = REPORT_DIR / 'meta_bayes_onepager.pdf'
```

AELP2/tools/render_quiz_overlay.py

```
#!/usr/bin/env python3 from __future__ import annotations """ Render timed quiz-style overlay cards on a 9:16 video using ffmpeg drawbox/drawtext. Usage: python3 AELP2/tools/render_quiz_overlay.py \ --video AELP2/outputs/renders/runway/clip.mp4 \ --out AELP2/outputs/finals/clip_quiz.mp4 \ [--pack AELP2/creative/overlays/quiz_pack.json]
```

AELP2/tools/render_sim_onepager.py

```
#!/usr/bin/env python3 """ Render a simple, marketer-friendly one-pager (Markdown) summarizing simulator fidelity. Reads: - AELP2/reports/sim_fidelity_campaigns.json (Phase 1) - AELP2/reports/sim_fidelity_campaigns_temporal.json (Phase 2) - AELP2/reports/sim_fidelity_campaigns_journey.json (Phase 3) - AELP2/reports/sim_fidelity_roll.json (rolling-origin) - AELP2/reports/sim_forward_forecast.json (forward forecast)
```

AELP2/tools/report_policy_vs_mixed.py

```
#!/usr/bin/env python3 from __future__ import annotations """ Compare WBUA and dual-gate metrics on mixed vs policy-annotated data. Writes AELP2/reports/policy_vs_mixed_summary.json """ import json, os, math, random from pathlib import Path ROOT = Path(__file__).resolve().parents[2]
```

AELP2/tools/rl_shadow_score.py

```
#!/usr/bin/env python3 """ RL shadow scorer using temporal v2 simulator parameters. Inputs: - Optional: proposals JSON AELP2/reports/rl_proposals.json of the form [{"campaign_id":"...","spend_next_day":1234.56}, ...] If not present, uses last-7 median spend per campaign. Output:
```

AELP2/tools/score_ad_items.py

```
#!/usr/bin/env python3 from __future__ import annotations """ Score ad items using success_config proxies (best-effort on available fields). Inputs: AELP2/competitive/ad_items_raw.json, AELP2/competitive/success_config.json Outputs: AELP2/competitive/ad_items_scored.json """ import json, datetime as dt from pathlib import Path
```

AELP2/tools/score_new_ads.py

```
#!/usr/bin/env python3 from __future__ import annotations """ Score local final videos with the Meta-only new-ad ranker (light features). Features are proxied from
```

filename since we don't have copy/meta for new assets. Outputs: AELP2/reports/new_ad_scores.json """ import json, joblib from pathlib import Path

AELP2/tools/score_vendor_creatives.py

#!/usr/bin/env python3 from __future__ import annotations """ Score all creative_objects (including vendor-imported) with the trained new-ad ranker. Steps: 1) Ensure features are built (build_features_from_creative_objects.py) 2) Load model, calibrator, feature map, and reference vector 3) Score each creative_id and write vendor_scores.json

AELP2/tools/selector_utility_baseline.py

#!/usr/bin/env python3 from __future__ import annotations """ Baseline utility selector: rank creatives by expected utility proxy and export Top-N per campaign (offline only). Utility: U = purchases - spend/target_CAC, with target from target_cac.json or campaign median. Outputs: AELP2/reports/utility_topn.json """ import json, os from pathlib import Path

AELP2/tools/self_judge.py

#!/usr/bin/env python3 from __future__ import annotations """ Self-judging evaluator for creative quality. Scores: - interestingness (0-1): motion energy in first ~2s, cut density proxy - creativity (0-1): color entropy + hue variance (novelty proxy) - relevance (0-1): phone/hand proxies via edge density near center + skin-tone ratio (very lightweight) - legibility (0-1): contrast of caption region + font size proxy (if detect text via QC heuristic)

AELP2/tools/serve_previews.py

#!/usr/bin/env python3 from __future__ import annotations """ Serve AELP2/outputs/finals on http://127.0.0.1:8080 for local/SSH-tunnel preview. """ import http.server, socketserver from pathlib import Path import os ROOT = Path(__file__).resolve().parents[2]

AELP2/tools/sim_fidelity_campaigns.py

#!/usr/bin/env python3 """ Phase 1 fidelity: per-campaign heterogeneity What it does - Pulls Meta daily insights (by campaign, last 28 days, daily rows) - Splits into train (first 14 days) and test (next 7 days) - Fits per-campaign CPC distributions (lognormal, clamped to p5–p95) - Fits per-campaign CVR priors (Beta with partial pooling to account-level) - Simulates purchases for test days given actual spend, aggregates to account/day

AELP2/tools/sim_fidelity_campaigns_journey.py

#!/usr/bin/env python3 """ Phase 3 fidelity: journey stages + recency proxy Extends Phase 2 with a simple stage model: - Stage shares inferred from frequency: higher freq -> more 'decision' stage - Stage multipliers: awareness 0.35x, consideration 0.7x, decision 1.0x (clamped) - Combine with weekday and frequency fatigue effects Outputs

AELP2/tools/sim_fidelity_campaigns_temporal.py

#!/usr/bin/env python3 """ Phase 2 fidelity: temporal patterns (weekday seasonality + frequency fatigue) Builds on Phase 1 per-campaign heterogeneity, and adds: - Weekday multipliers for CVR per campaign with shrinkage to account-level - Frequency fatigue: log(CVR) ~ k*(freq - median_freq_train), clamp effect Outputs - JSON: AELP2/reports/sim_fidelity_campaigns_temporal.json

AELP2/tools/sim_fidelity_campaigns_temporal_v2.py

#!/usr/bin/env python3 """ Temporal simulator v2: time-decayed CVR priors + auto window (per-campaign) Features in this step: - Auto-select train window per campaign (7/14/21) based on drift & support - Time-decayed counts in Beta CVR prior (half-life configurable) Outputs - JSON: AELP2/reports/sim_fidelity_campaigns_temporal_v2.json

AELP2/tools/sim_fidelity_campaigns_temporal_v3.py

#!/usr/bin/env python3 """ Temporal simulator v3: per-campaign hourly effect + fast-drift window rule Adds to v2: - Per-campaign hourly CVR multiplier (shrink to account-level if sparse) - Fast-drift train window: force 7-day if CVR drift > 0.2 or median creative age < 7 Outputs - JSON: AELP2/reports/sim_fidelity_campaigns_temporal_v3.json

AELP2/tools/sim_fidelity_eval.py

#!/usr/bin/env python3 """ Simulation Fidelity Check (AELP ↔ AELP2) Uses existing AELP2 ProductionFortifiedEnvironment (auction via AuctionGymWrapper) and LegacyEnvAdapter (bid calibration) to simulate day-level outcomes and compare them to real Meta outcomes over a recent window. No live changes are made. What it does: - Calibrates the auction (saves calibration to AELP2 .auction_calibration.pkl)

AELP2/tools/sim_fidelity_eval_empirical.py

#!/usr/bin/env python3 import os, json, random, requests from dataclasses import dataclass from typing import List @dataclass class Row: date: str spend: float impr: int

AELP2/tools/sim_fidelity_journey_criteo.py

#!/usr/bin/env python3 """ Journey + Criteo fidelity evaluation (no BigQuery writes). Train→test split (time-based): - Train on first K days (estimate CPC distribution and base CVR) - Test on next days: for each day, simulate spend → clicks using CPC samples, then per-click conversion probability adjusted by Criteo CTR vs train CTR. Outputs JSON with purchases/day MAPE and CAC/day MAPE on held-out days.

AELP2/tools/simulate_bandit_from_forecasts.py

#!/usr/bin/env python3 from __future__ import annotations """ Offline bandit simulation using the forecast distributions for Top-20 blueprints. - Thompson Sampling with Beta-Bernoulli proxy on "success" = signups rate per impression, calibrated from forecast p50 CTR/CVR and CPM for each blueprint. - Simulates allocation over T days for a chosen daily budget. Outputs: AELP2/reports/rl_offline_simulation.json """

AELP2/tools/sort_uploaded_assets.py

#!/usr/bin/env python3 from __future__ import annotations """ Sort a big "drop" folder of mixed digital assets into the repo structure. Default source: ~/AELP/uploads (override with --src). Rules (by extension and simple heuristics): - Video (.mp4,.mov,.m4v,.webm) → AELP2/assets/broll/raw/ - Audio (.mp3,.wav,.aac) → AELP2/assets/audio/raw/

AELP2/tools/still_to_motion.py

```
#!/usr/bin/env python3 from __future__ import annotations """ Turn a still image (1080x1920 recommended) into a subtle-motion MP4. Effects: gentle pan/zoom, optional vignette. Usage: python3 AELP2/tools/still_to_motion.py --image <path> --seconds 3.0 --out <out.mp4> """ import subprocess, argparse
```

AELP2/tools/tag_ad_patterns.py

```
#!/usr/bin/env python3 from __future__ import annotations """ Tag ads with hook_type, emotion, proof_device, captions_present (heuristics on text), format/length (unknown -> defaults), brand/claim/CTA (from text cues). Inputs: AELP2/competitive/ad_items_scored.json Outputs: AELP2/competitive/ad_items_tagged.json and pattern freqs """ import json, re from pathlib import Path
```

AELP2/tools/train_new_ad_ranker.py

```
#!/usr/bin/env python3 from __future__ import annotations """ Train a lightweight pairwise ranker on weekly pairs (feature diffs). Model: LogisticRegression on X (diff vector) -> P(win vs baseline) Calibration: Isotonic on held-out fold; Conformal: absolute residual quantile. Artifacts saved to AELP2/models/new_ad_ranker/ - model.pkl (sklearn pipeline) - calib.pkl (IsotonicRegression)
```

AELP2/tools/tune_generator_priors.py

```
#!/usr/bin/env python3 from __future__ import annotations import json from pathlib import Path ROOT = Path(__file__).resolve().parents[2] ENR = ROOT / 'AELP2' / 'reports' / 'creative_enriched' OUTD = ROOT / 'AELP2' / 'priors'; OUTD.mkdir(parents=True, exist_ok=True) def main():
```

AELP2/tools/uplift_baseline_test.py

```
#!/usr/bin/env python3 from __future__ import annotations """ Historic baseline uplift test. For each campaign (creative_enriched/*.json): - Choose a baseline ad: older (age_days >= MIN_BASE_AGE) and adequate volume - For each later variant (age_days < baseline.age_days), compare: sim_winner = sim_score(variant) > sim_score(baseline) actual_winner = beats by purchases & CAC: purch_v >= purch_b and cac_v <= 1.0 * cac_b
```

AELP2/tools/validate_candidates.py

```
#!/usr/bin/env python3 from __future__ import annotations import json, sys from pathlib import Path ROOT = Path(__file__).resolve().parents[2] KB_DIR = ROOT / 'AELP2' / 'knowledge' / 'products' CAND = ROOT / 'AELP2' / 'outputs' / 'creative_candidates' OUT = ROOT / 'AELP2' / 'reports' / 'candidates_validation.json'
```

AELP2/tools/validate_kb.py

```
#!/usr/bin/env python3 from __future__ import annotations import json, sys from pathlib import Path ROOT = Path(__file__).resolve().parents[2] SCHEMA = ROOT / 'AELP2' / 'knowledge' / 'schema' / 'product_kb.schema.json' DIR = ROOT / 'AELP2' / 'knowledge' / 'products' def load_schema():
```

AELP2/tools/voice_library_add_prompt.py

```
#!/usr/bin/env python3 from __future__ import annotations import json, sys from pathlib import Path ROOT = Path(__file__).resolve().parents[2] LIB = ROOT / 'AELP2' / 'branding' / 'voice_library.json' def main(): prompt = sys.argv[1] if
```

```
len(sys.argv)>1 else None
```

AELP2/tools/weekly_fidelity_from_v3.py

```
#!/usr/bin/env python3 from __future__ import annotations """ Compute weekly (7-day aggregated) fidelity from sim_fidelity_campaigns_temporal_v3.json. Aggregates actual and predicted purchases across the last 7 days and reports weekly relative error and interval coverage. Output: AELP2/reports/weekly_fidelity.json """ import json from pathlib import Path
```

AELP2/tools/weekly_relabel.py

```
#!/usr/bin/env python3 from __future__ import annotations """ Recompute weekly labels from creative_enriched/*.json and write to creative_weekly/*.json Labels use the same quantile rules but apply to the weekly-aggregated fields already present. """ import json, os, math from pathlib import Path ROOT = Path(__file__).resolve().parents[2]
```

AELP2/tools/weekly_topN_dual.py

```
#!/usr/bin/env python3 from __future__ import annotations """ Export weekly Top-N creatives that pass the dual gate with a conservative lower-bound utility. Inputs: - weekly_creatives/*.json - dual_gate_weekly.json (to pick operating point) - target_cac.json (optional; else median CAC per week)
```

AELP2/validation/acceptance_criteria.py

```
"""Acceptance criteria validation for AELP2 production system. This module validates that ALL production requirements are met: - All required components exist and function - No fallbacks or simplifications - Minimum performance requirements met - Production safety gates operational - BigQuery integration working - Attribution system functional
```

AELP2/validation/production_checklist.py

```
#!/usr/bin/env python3 """ Production Readiness Validation for AELP2 This module performs comprehensive validation of AELP2 system against production requirements and acceptance criteria. NO SHORTCUTS - validates actual functionality. Validation Categories: 1. Configuration Requirements 2. Component Integration
```