It worked in the notebook

..now what?

Migrating scoring from sklearn to Pytorch

whoami

- Data Scientist / Developer @ Celtra's Analytics team
- Used to working in notebooks, locally, in Snowflake – not "fully *in*" the AWS stack
- For this task I had to wear the DevOps / MLOps hat too

Ivan Nikolov

Celtra

We’re Celtra.

Driving media effectiveness through exceptional creative execution.

400+
enterprise-level customers

5M+
creative assets produced annually

120B+
served ad impressions annually

100+
Integrated DSPs & SSPs (TTD, DV360, Xandr, etc)

adidas NBCUniversal CNN HELLO FRESH SIEMENS Unilever HEARST

Spotify CONDÉ NAST lyft T··Mobile· YETI Mail Online

The real challenge isn’t generating creatives or variant volume—it’s knowing which ones truly deserve your budget.

Celtra

Make confident, data-driven decisions

The Core Challenge

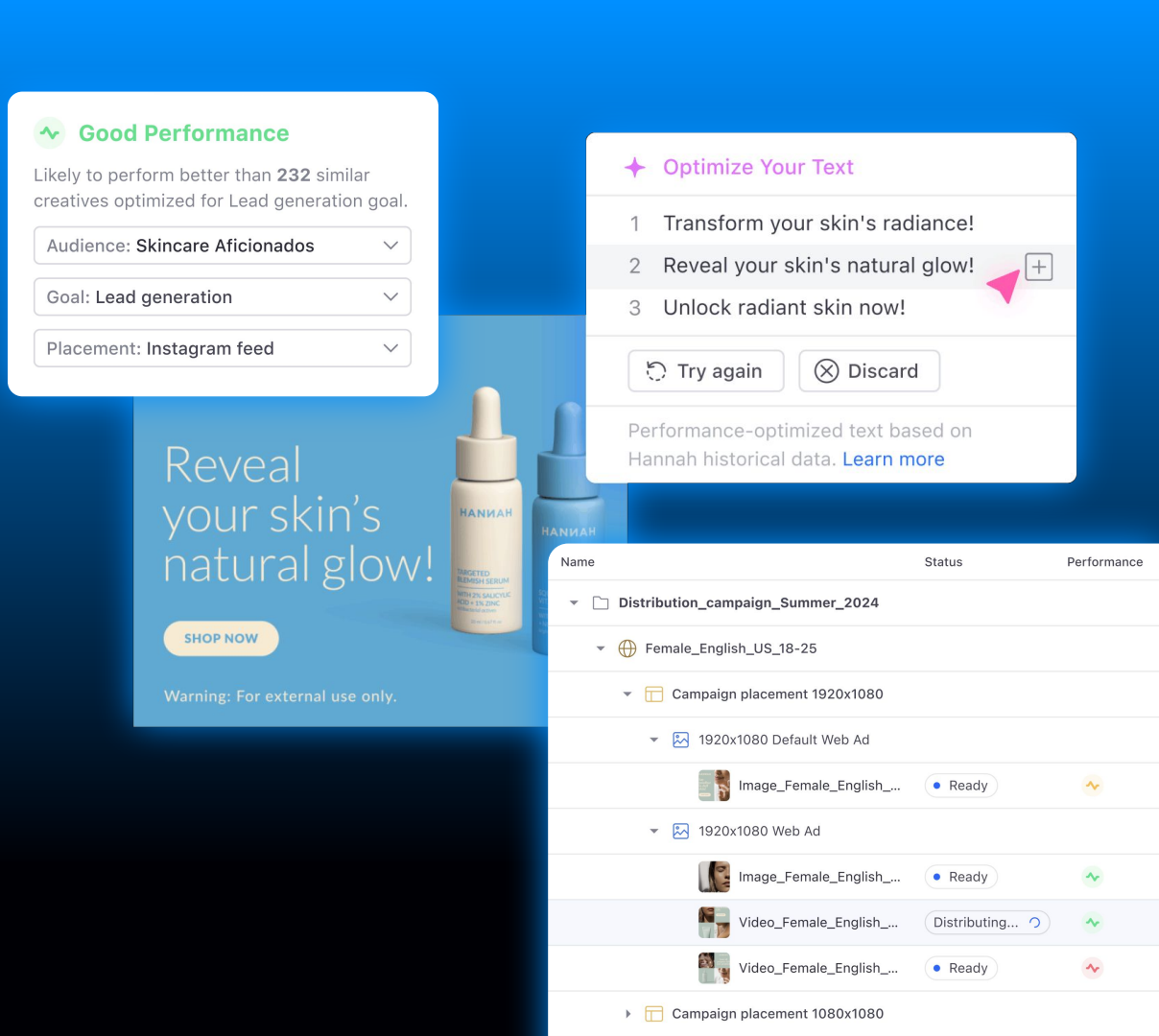
Can we reliably score and predict the performance of a creative ad asset **before** actually trafficking it?

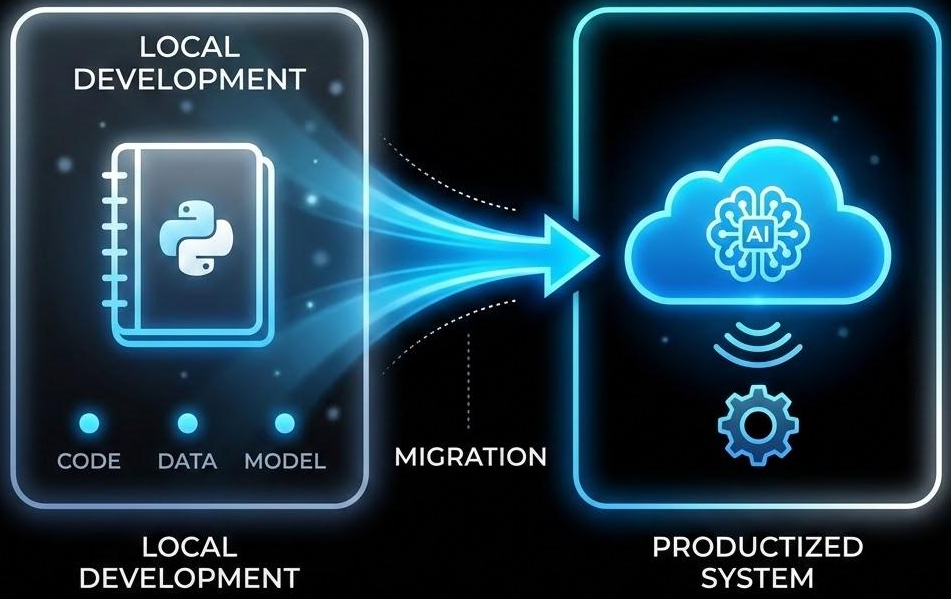
Available Data Assets

We possess a vast library of rich ad assets (images and videos) and have the ability to directly link them to their historical performance data.

The ML Opportunity

By leveraging this historical data, can we train a robust **Machine Learning model** to automate and scale our creative pre-scoring process?





Today's talk is about migration to production, and the problems we faced when iterating upon our work.

What we had

What we had

Predictive Scoring

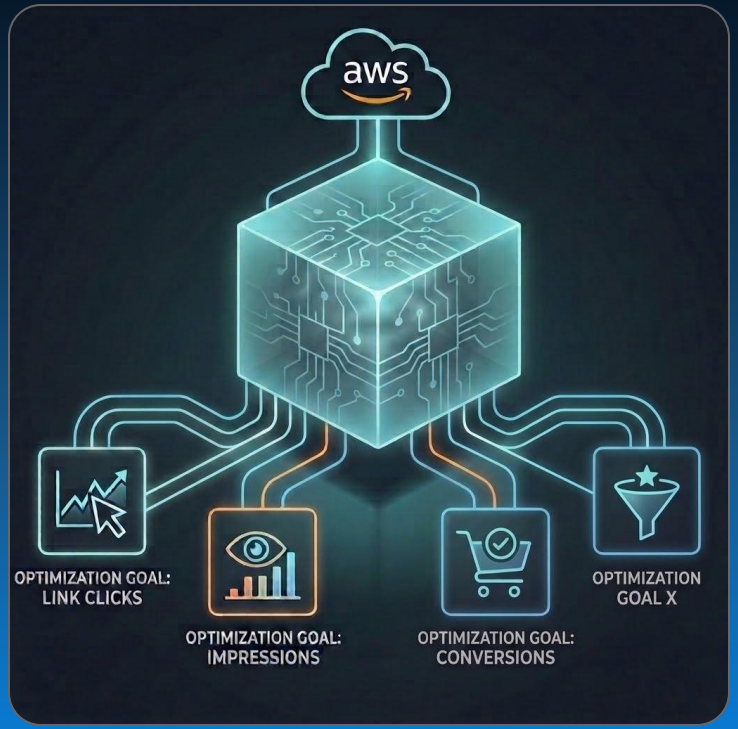
Classification task based on historical performance data. Each ad image is dynamically categorized into performance tiers: **bad**, **good**, or **great**.

Backbone Architecture

Multi-Layer Perceptron (MLP) trained on top of a powerful foundation model backbone, utilizing **AWS Bedrock's Titan** for deep feature representation.

Model Complexity

A separate, dedicated MLP model was deployed for each individual optimization goal, leading to a multi-model pipeline.



Celtra

How was it done?

Training

Training of the models locally in a Jupyter notebook.

Simple setup

Utilizing scikit-learn for lightweight classification heads.

Exporting

Exporting optimized joblib artifacts for production deployment.

Serving

Robust inference and model serving on AWS SageMaker.

Celtra

What is SageMaker?

Build, Train & Deploy

Amazon SageMaker is a fully managed AWS machine-learning platform for building, training, and deploying ML models.

MLOps

It includes tools for model monitoring, versioning, security, and performance optimization.

Managed Inference

It provides managed inference endpoints for serving real-time predictions without managing the underlying servers.

Celtra

Scikit-learn SageMaker MME (Multi model endpoint)

Model Artifacts

S3 Storage
Models are packaged as tar.gz files containing joblib artifacts (model weights + model class).

Dynamic Loading
SageMaker dynamically pulls these model archives from S3 on demand.

Inference Code

inference.tar.gz
Custom inference code overrides default python handlers:

`input_fn` - parses json
`model_fn` - loads joblib
`predict_fn` - predicts
`output_fn` - JSON response

Orchestration

CDK Deployment
Deploys package archives and automates the infrastructure wiring.

SageMaker MME
Hosts multiple models efficiently on a single shared endpoint.

On-Demand Invoke
TargetModel header determines which model tar is loaded dynamically.

```
export class SageMaker extends Construct {
    readonly model: sagemaker.CfnModel
    readonly endpointConfig: sagemaker.CfnEndpointConfig
    readonly endpoint: sagemaker.CfnEndpoint

    constructor(scope: Construct, id: string, props: SageMakerProps) {
        super(scope, id)

        // create tar files from the inference.py and model directories.
        const archives = prepareModelArchives()

        /**
         * Upload inference code to S3. Upload as asset since we do not care about the name of the
         * Asset upload will only trigger if there are changes to the inference script.
         */
        const inference = new S3Assets.Asset(this, 'inference', {
            path: archives.inferenceArchiveLocation,      [DEV-2089] Introduce Sagemaker to infrast
        })

        /**
         *  Uploading model files must be done as a bucket deployment since we need to create a dir
         *  Models are uploaded as a tar.gz file which is versioned by the version file in the mode
         */
        const destinationKeyPrefix = 'models'
        const models = new S3Deployment.BucketDeployment(this, 'sagemakerModels', {
            sources: [S3Deployment.Source.asset(archives.models.modelArchivesLocation)],
            destinationBucket: props.bucket,
            destinationKeyPrefix,
            prune: false,
            memoryLimit: 2048,
        })
```

E2E flow

01

Client Request

POST /api/score

Triggers the creative-score service endpoint to start processing.

02

Embedding

Bedrock Titan

Converts the input image URL into a high-dimensional 1024-d embedding.

03

Model Inference

SageMaker Invoke

Invokes the dynamic endpoint passing TargetModel as model_name.tar.gz.

04

Response

Classification

Returns the predicted classification results back to the client.

This is great but…

scikit-learn limitations

Limited Flexibility

Hard to customize the training loop, loss functions, and network architecture.

Scalability Constraints

Offers highly limited GPU support and struggles to scale with massive datasets.

PyTorch: The Real Tool for DL stuff

Extensible & Modern

Easier to extend with new deep learning architectures and custom workflows.

Production-Ready Power

Provides faster training on GPU, better debugging visibility, and acts as a foundation for future models.

E2E flow

01

Client Request

POST /api/score

Triggers the creative-score service endpoint to start processing.

02

Embedding

Bedrock Titan

Converts the input image URL into a high-dimensional 1024-d embedding.

03

Model Inference

SageMaker Invoke

Invokes the dynamic endpoint passing TargetModel as model_name.tar.gz.

04

Response

Classification

Returns the predicted classification results back to the client.

Migration

It should be easy right? We just rewrite the code from sklearn to Pytorch and change the inference image

Well rewriting training was easy

Local Training

We don’t train using SageMaker, everything is done locally, so this was easy

PyTorch Rewrite

Just rewrite the data loading and training using pytorch lightning,

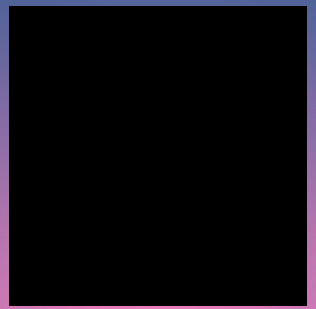
we know how to do this and agentic development helps a lot

Model Weights

Instead of the whole serialized model class we just output the serialized model weights

… and everything worked in the notebook, now what?

… now we need to figure out how to migrate the model serving



We want something that

Quick Setup

That is implemented quickly

Low Friction

Doesn’t require too many changes

Multi-Model

Supports serving of multiple models

CPU Powered

Does inference on CPU

There are multiple paths

1	**Quick & Dirty PyTorch Integration** Add PyTorch to existing sagemaker-scikit-learn image	🐍 Dependency hell Far from an elegant solution
2	**Official SageMaker v2 PyTorch Image** Utilize the pre-built AWS inference container	PyTorch is already installed Uses TorchServe (limited maintenance)
3	**Custom Multi-Model Server** Build custom inference container via awslabs/multi-model-server	Customizable We own image maintenance and security updates. •More CI/CD and infrastructure work.
4	**NVIDIA Triton Inference Server** Adopt high-performance production-ready server	Highly modern & optimized Excessive overhead for current use-case
5	**SageMaker V3 Pipeline Migration** Transition entire orchestration ecosystem	Cleanest architecture if training co-exists High effort; outside current scope

There are multiple paths

1	**Quick & Dirty PyTorch Integration** Add PyTorch to existing sagemaker-scikit-learn image	🐍 Dependency hell Far from an elegant solution
2	**Official SageMaker PyTorch Image** Utilize the pre-built AWS inference container `pytorch-inference:2.5.1-cpu-py311-ubuntu22.04-sagemaker`	PyTorch is already installed Uses TorchServe (limited maintenance)
3	**Custom Multi-Model Server** Build custom inference container via awslabs/multi-model-server	DevOps team disliked this :(TODO: ping Mitja
4	**NVIDIA Triton Inference Server** Adopt high-performance production-ready server	Highly modern & optimized Excessive overhead for current use-case
5	**SageMaker V3 Pipeline Migration** Transition entire orchestration ecosystem	Cleanest architecture if training co-exists High effort; outside current scope

Torchserve

API Endpoints

Serves PyTorch models through REST and gRPC endpoints, making them accessible to applications.

Production Ready

Handles production concerns such as model loading, worker management, batching, logging, metrics, and scaling.

Multi-Model Hosting

Supports multiple models and model versioning, enabling deployment of several models from a single server.

Custom Workflows

Custom handlers allow you to implement preprocessing, inference, and postprocessing logic for your model.

Adjust to TorchServe

Remap Handlers

Translate previous scikit-learn pipeline handlers to the native TorchServe handler interface.

```python
import json      Vid Stropnik, 15 months ago • Prepare model training notebook/script (#277
import os                                                                          #1 202
from io import StringIO                                                            #2 202

import joblib                                                                      #1 202
from celtra_scorer import CeltraScoreClassifier                                    #2 202
from inference_constants import REQUEST_HEADER_CONTENT_TYPE
from inference_types import ModelOutput


def model_fn(model_dir: str) -> CeltraScoreClassifier:

    return joblib.load(os.path.join(model_dir, 'model.joblib'))


def input_fn(request_body: str, request_content_type: str) -> dict:
                                                                                   #1 202
    if request_content_type == REQUEST_HEADER_CONTENT_TYPE:                        #2 202
        input_data = json.loads(StringIO(request_body).read())
        return input_data
```

TorchServe Interface

Implement initialization, preprocessing, and custom model inference within a unified handler class.

```python
class CeltraScoreHandler(BaseHandler):      Ivan Nikolov, 3 weeks ago • [DEV-5829] Migrate Celtra Scorer to
    """Loads ``CeltraScoreClassifier`` from ``model.pt`` in the MAR extract directory."""

    def __init__(self):
        super().__init__()
        self.model: CeltraScoreClassifier | None = None
        self.initialized = False

    def initialize(self, context: Any) -> None:
        if self.initialized:
            return
        properties = context.system_properties
        model_dir = properties.get("model_dir")
        if not model_dir:
            raise RuntimeError("TorchServe context missing model_dir")
        self.model = CeltraScoreClassifier.load_for_inference(Path(model_dir) / "model.pt")
        self.initialized = True

    def preprocess(self, data: list[dict]) -> dict:
        request_content_type: str | None = None
        if self.context is not None:
            headers = self.context.get_all_request_header(0)
            request_content_type = _request_header_value(headers, "Content-Type")
```

Adjust to TorchServe

Model Archive (MAR) Components

A TorchServe Model Archive (.mar) bundles all necessary artifacts for deployment. Archived used a CLI
Training script packages everything

- **Inference Code:** Custom handler and supporting Python files to process requests.
- **Model Weights:** The serialized PyTorch model state (`model.pt`).
- **Model Configuration:** Definition of workers, batch sizes, and resource limits.
- **Manifest:** Generated metadata file detailing model name, runtime …
- **Requirements**

model_archive_structure

```
├── celtra_scorer.py
├── inference_constants.py
├── inference_types.py
├── MAR-INF/
│   └── MANIFEST.json
├── model_fields.py
├── model-config.yaml
├── model.pt
├── requirements.txt
└── torchserve_handler.py
```

What changed in the SageMaker runtime?

Model Dependencies

Dependencies are isolated and installed per model dynamically.

- **Isolated venvs:** Each model gets its own virtual environment and specific dependencies.
- **Cold starts:** Requires additional time to install dependencies during the first invocation. (~17s)

Local vs. Production

Local development closely mirrors but does not perfectly match production.

- **Registration Script:** Local development requires an extra script to register models (same in scikit-learn too)
- **Discrepancy Risks:** Subtle bugs can go unnoticed; always dev-test on a dev stack too.

What about the deployment?

S3 Upload & Versioning

CDK automates the deployment pipeline for Model Archives.

- Takes generated MARs
- Versions each release
- Uploads assets to S3

CI State Validation

Automated checks to ensure consistency between repo state and MAR state.

- Detects unsaved logic changes
- Validates git state
- **Prevents out-of-sync inference**

Runtime Configuration

Environment configuration specifically tailored for production serving.

- Slightly different ENV vars

What did we learn?

Lessons learned

MME & Container

Same MME pattern, different Deep Learning Container—Multi-Model stayed. Switching sklearn → PyTorch was mostly image + artifact format + env vars.

MAR Deploy Unit

MAR as the unit of deploy — model + handler + deps travel together. No separate “submit directory” for inference code.

Cold Start Delay

17s first load occurs when TorchServe installs per-model requirements. We did not address this issue.

Local ≈ Cloud

docker-compose uses same PyTorch inference image and TS_* flags as CDK.

⚠️ Bugs can go unnoticed; always test on dev AWS before prod.

Pragmatism Over Purity

The official PyTorch image met our needs without the complexity of Triton, a custom server, or a full migration.

Documentation

Code with limited/contradicting docs is NPA-hard (not solvable by Agents in polynomial time :D), so check the source code directly.