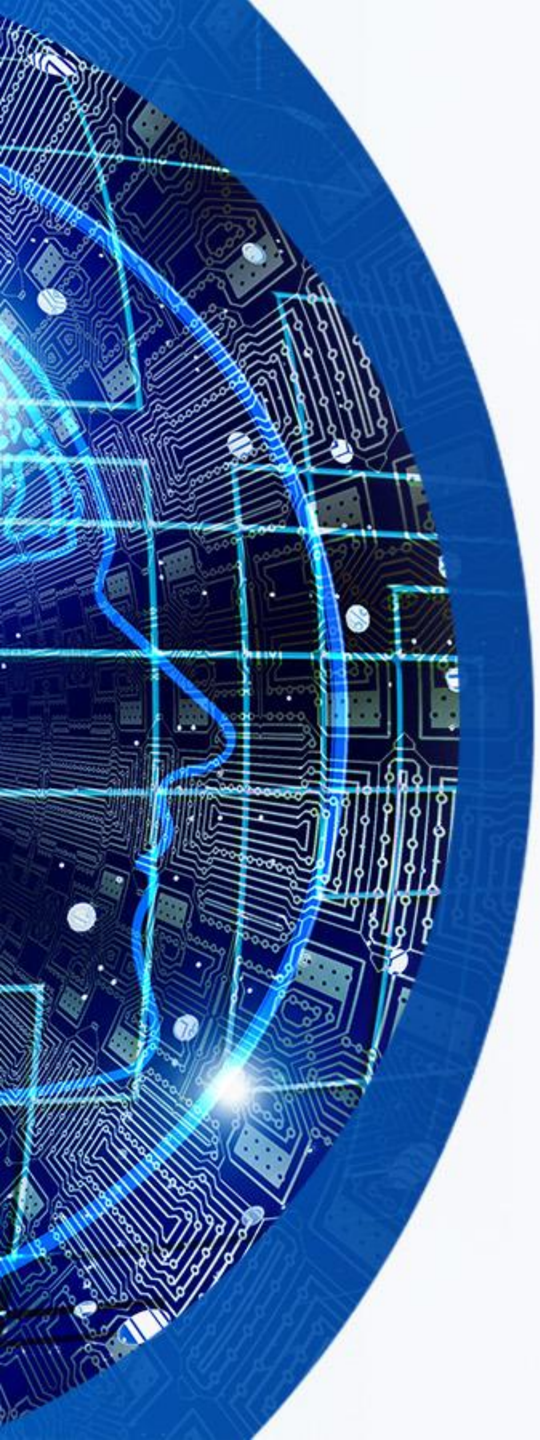
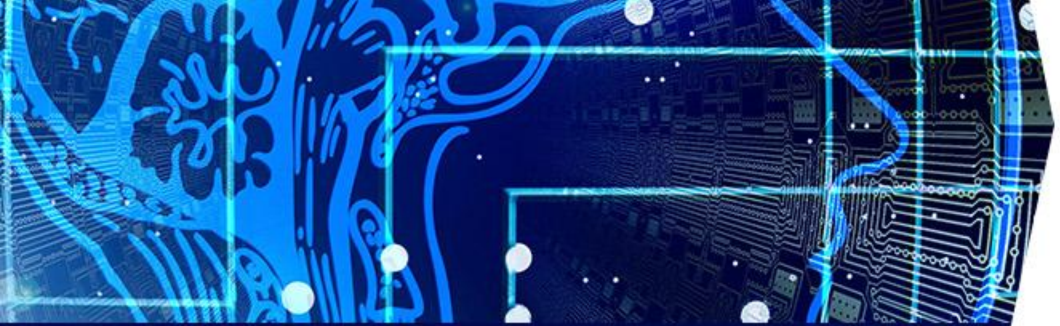
Digital Advertising

To auction “keywords”
or not to auction
that is the question
Gruppe 2



Agenda

- Einführung / «Business Case» (MAC)
- Data (PK)
  - Generierung
  - Normalisierung
- Code
  - Übersicht / Reward / Learning (RB)
  - Optimierung (Tensorboard / Optuna) (PZ)
- Analyse
  - Hyperparameter_tuning script (EO)
  - Tensorboard-analyzer / Visualization-ad_performance scripts (EO)
  - Data comparisons / Conclusions (EO)
- Lessons-Learned / Herausforderungen (all)



Einführung – Business Case (MAC)

Einführung

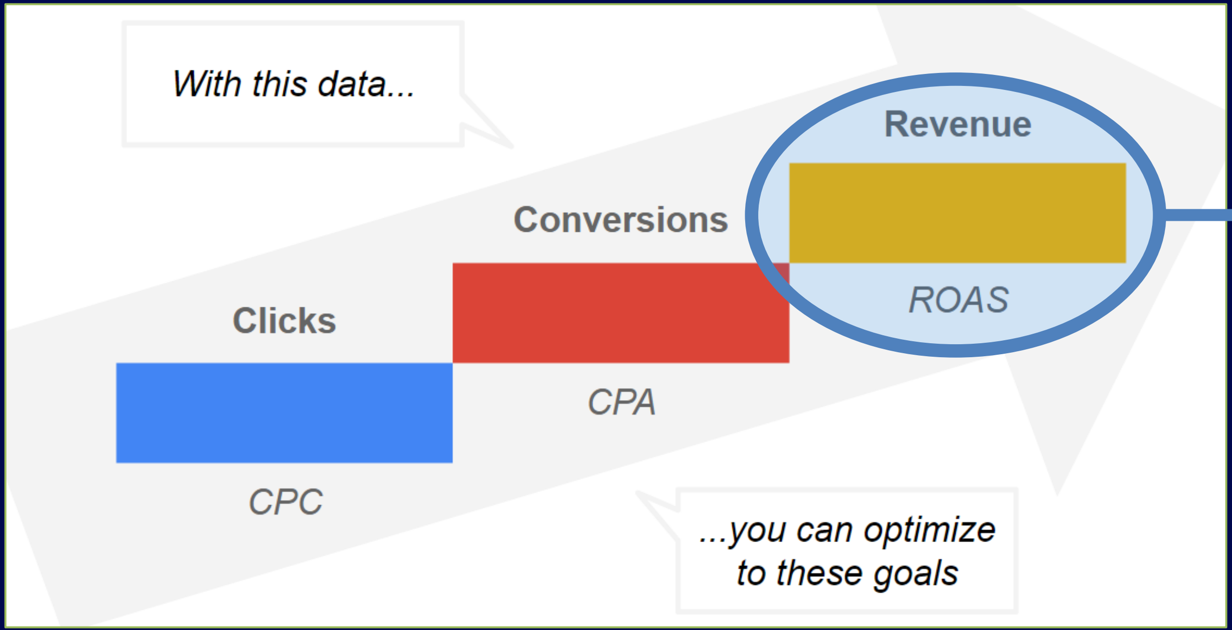
- Datenbasierte Budgetallokation (10% des vorhandenen Bargeldes) dank Einsatz von KI bzw. Reinforcement Learning
- Von der Datenaufbereitung via Modell-Training zur fundierten Analyse und Verständnis

Business Case

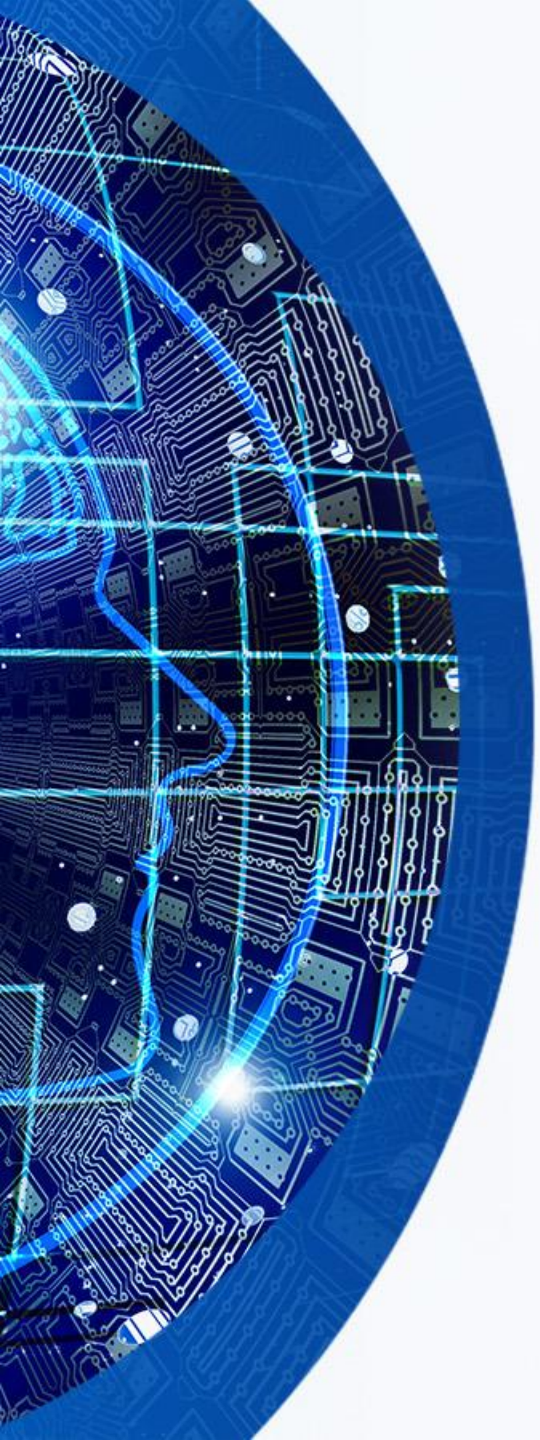
- Unter der Annahme, dass (brauchbare) Daten vorliegen, rechnen wir mit einer Umsetzungszeit von 3-4 Monaten
- Danke einer Effektivitätssteigerung des ROAS um 20% erhalten wir einen positiven ROI nach 12 Monaten

- Mögliche Herausforderungen / Abhängigkeiten
  - Messbarkeit / Zuordnung der Effektivitätssteigerung
  - Produktions- / Lieferzeiten
  - Sonstiges (z.B. Landingpage-Qualität, andere Keywords …)

Optimizing is all we need



Wir optimieren den Umsatz, der durch Werbeausgaben generiert wird, im Verhältnis zu den Kosten dieser Werbung.



Agent

Zustand (state (S_t))

- Vorhandenes «Bargeld» (cash)
- «Holdings» - aktive keywords

- keyword
- competitiveness
- difficulty_score
- organic_rank
- organic_clicks
- organic_ctr
- paid_clicks
- paid_ctr
- ad_spend
- ad_conversions
- ad_roas
- conversion_rate
- cost_per_click
- cost_per_acquisition
- previous_recommendation
- impression_share
- conversion_value

- done, terminated, truncated

Belohnung (reward (R_t))

- ROAS-basiert: ad_roas
→ maximiere Return on Ad Spend

- Kosteneffizienz: ad_conversions / ad_spend
  → maximieren Conversions pro Ausgabeeinheit

- Kombination: 0.6 * ad_roas + 0.4 * organic_clicks
  → Vergangene Investition zahlt sich aus

- Nur «Organisches» Ranking: organic_ctr
  → Vergangene Investition zahlt sich aus

Aktion (action (A_t))

- 1 Keyword kaufen/bieten oder nichts tun

reward (R_{t+1})

action (A_{t+1})

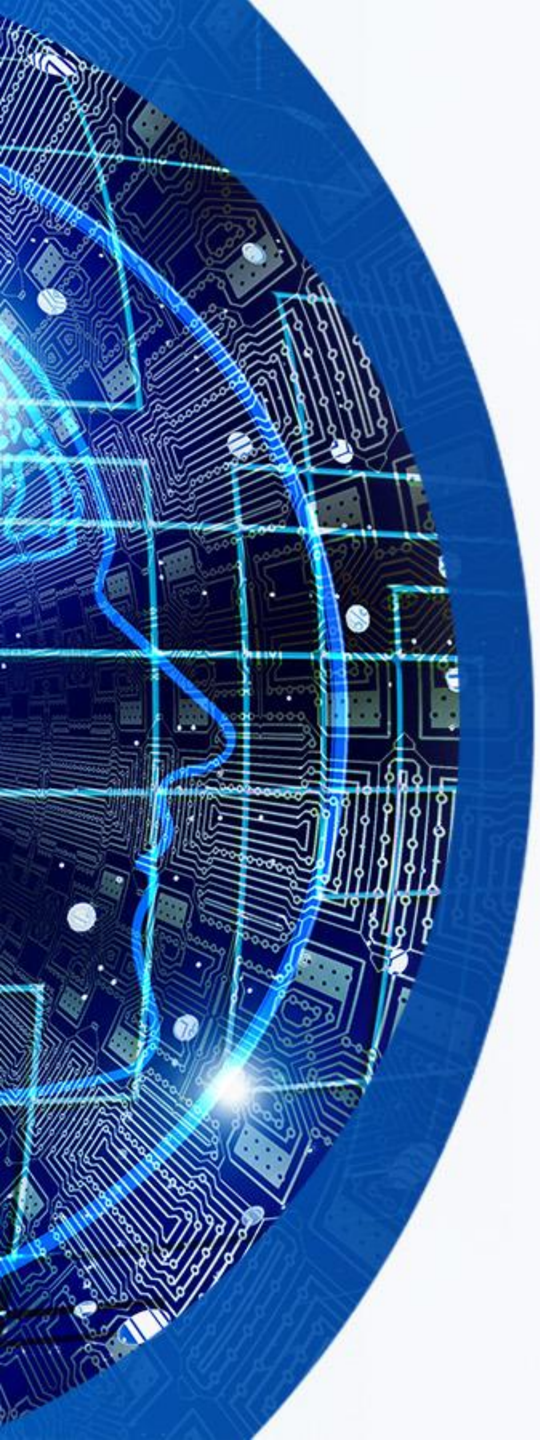
Umgebung

state (S_{t+1})

Daten (PK)

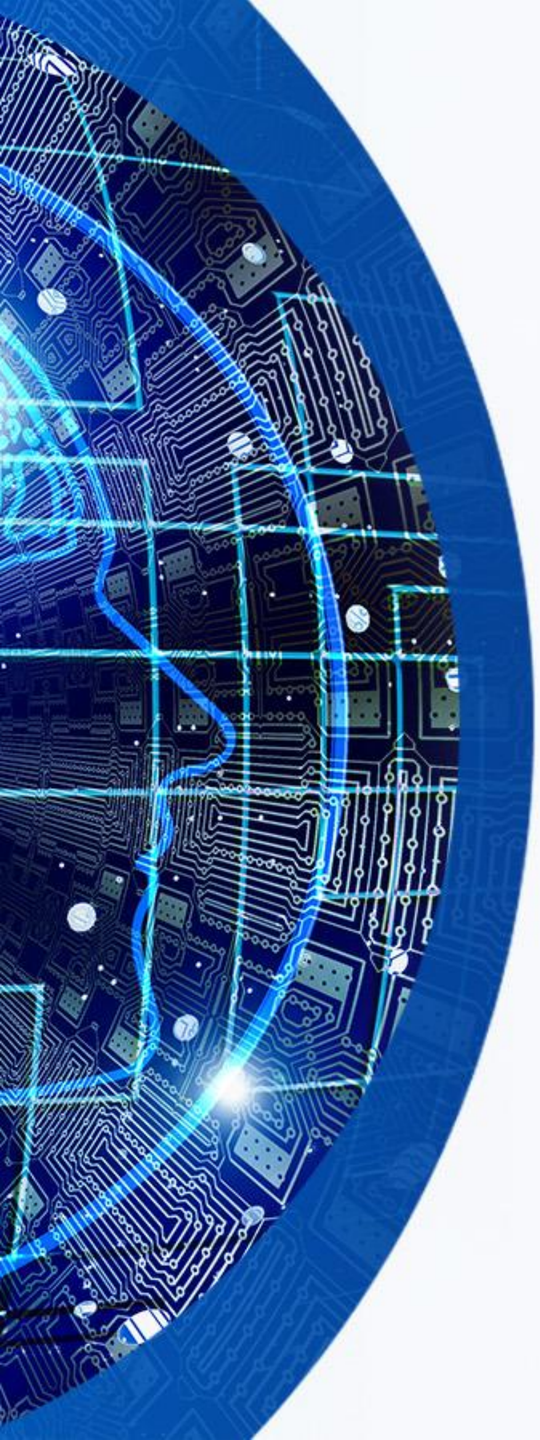
- Suche nicht zielführend
  - Shopping Mall Paid Search Campaign Dataset
  - PPC Campaign Performance Data

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R
1	keyword	competitivene	difficulty_scor	organic_rank	organic_clicks	organic_ctr	paid_clicks	paid_ctr	ad_spend	ad_conversior	ad_roas	conversion_ra	cost_per_click	cost_per_acqu	previous_reco	impression_sh	conversion_value	
2	Autoversicher	0.56	0.866	80	131	0.15703732	746	0.1274197	1350.26	39	1.43688913	0.05227882	1.81	34.6220513	1	0.27564632	1940.17391	
3	Kreditvergleicl	0.72	0.309	95	34	0.01	465	0.09809572	1116	35	5.18030778	0.07526882	2.4	31.8857143	0	0.29563762	5781.22349	
4	Hotel buchen	0.34	0.806	75	117	0.07054571	107	0.03618353	131.61	4	4.39885891	0.03738318	1.23	32.9025	0	0.21776979	578.933821	
5	Flug buchen	0.8	0.214	34	308	0.21758421	485	0.09075523	1096.1	32	1.03442398	0.06597938	2.26	34.253125	1	0.5141958	1133.83212	
6	Mietwagen	0.88	0.349	86	59	0.01271818	1048	0.10847737	3028.72	57	0.71031707	0.05438931	2.89	53.1354386	0	0.44077783	2151.35151	
7	Online Bankin	0.43	0.413	23	157	0.13066824	402	0.12775712	590.94	16	4.93368431	0.039801	1.47	36.93375	0	0.24457658	2915.51141	
8	Sparplan	0.6	0.404	72	103	0.02719325	1202	0.13462178	372.62	66	22.8909404	0.05490849	0.31	5.64575758	0	0.31918163	8529.6222	
9	Reiseversiche	0.4	0.405	54	661	0.13385591	708	0.08440347	885	27	4.33914756	0.03813559	1.25	32.7777778	0	0.1	3840.14559	
10	Smartphone k	0.62	0.75	64	184	0.05797873	797	0.10085611	2255.51	33	2.35678364	0.04140527	2.83	68.3487879	0	0.23926687	5315.74906	
11	Laptop kaufen	0.66	0.615	32	51	0.08935746	203	0.10652401	491.26	13	4.32124211	0.06403941	2.42	37.7892308	0	0.21208059	2122.8534	
12	GÃ¼nstige Ho	0.73	0.319	77	27	0.01	657	0.097323	361.35	40	14.1922005	0.0608828	0.55	9.03375	0	0.24584845	5128.35165	



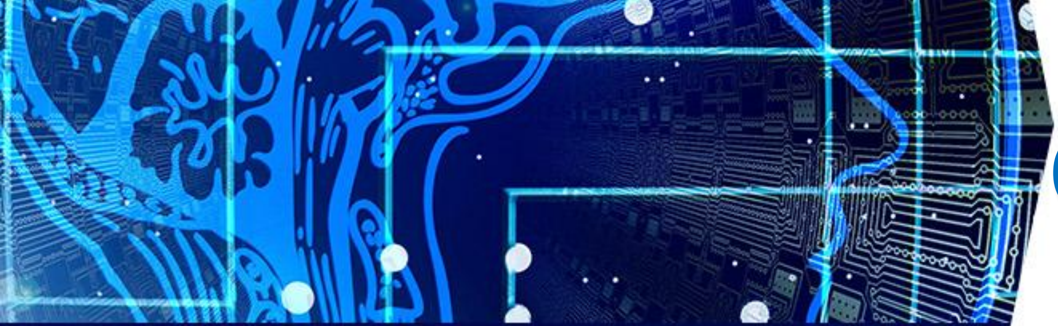
Daten

Feature	In diesem Datensatz	Reiner `np.random.uniform`	Realitätsgrad
CTR	`np.random.beta(2, 5)`	Gleichverteilung	Mittel – Beta-Verteilungen modellieren oft Klickwahrscheinlichkeiten besser, da sie asymmetrisch sein können (z. B. viele niedrige CTRs, wenige hohe).
Bid	`np.random.uniform(0.1, 5.0)`	Gleichverteilung	Niedrig – realistischer wären Cluster (z. B. viele kleine Bids, wenige hohe).
Competition	`np.random.uniform(0.1, 1.0)`	Gleichverteilung	Niedrig – könnte auch von Keyword-Typ abhängen.
Paid Clicks	Binomial (Impressions, CTR)	-	Relativ gut – Binomial passt hier gut, da Klicks von Impressionen & CTR abhängen.
Revenue	Zufällig pro Click	Gleichverteilung	Mittel – echte Einnahmen pro Klick hängen stark vom Keyword & Intent ab.
CPC	Bid × Competition	-	Okay – besser wäre ein dynamischeres Auktionsmodell.



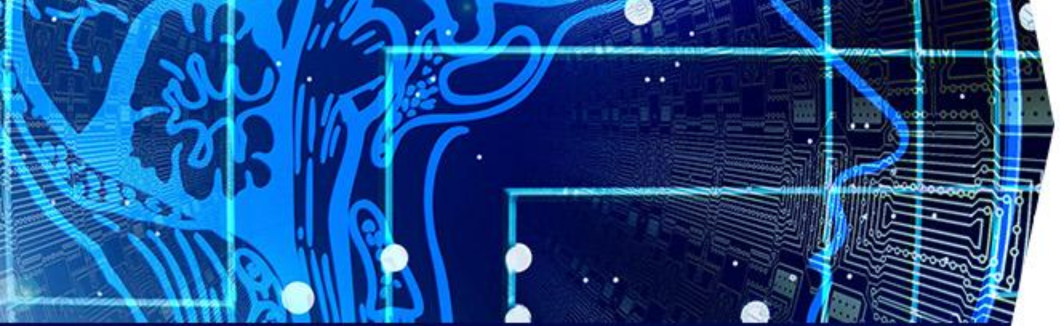
Code –Dateien (RB)

- digital_advertising.py - Deep Reinforcement Learning Module
- hyperparameter_tuning.py - Hyperparameter tuning mit Optuna
- visualize_ad_performance.py - Leistungsvisualisierung
- tensorboard-analyzer.py - Trainingsanalyse
- analyze_raw_data.py - Interaktiver Rohdatenexplorer



Code – digital_advertising.py

- Daten laden und in Training und Test aufteilen
- Custom Environment erstellen
  - specs
  - _reset
  - _step
  - _compute_reward
- Policy erstellen
- Trainingsloop mit integrierter Verifikation mit Test und speichern des besten Models
- Vorhersagen (Inference)
- Speichern der Werte in Tensorboard



Code
Custom Environment Specs

- Unsere Environment hat folgende Specs:

```
self.num_features = len(feature_columns)
self.num_keywords = get_entry_from_dataset(self.dataset, 0).shape[0]
self.action_spec = OneHot(n=self.num_keywords + 1) # select which one to buy or the last one to buy
nothing
self.reward_spec = Unbounded(shape=(1,), dtype=torch.float32)
self.observation_spec = Composite(
    observation = Composite(
        keyword_features=Unbounded(shape=(self.num_keywords, self.num_features), dtype=torch.float32),
        cash=Unbounded(shape=(1,), dtype=torch.float32),
        holdings=Bounded(low=0, high=1, shape=(self.num_keywords,), dtype=torch.int, domain="discrete")
    ),
    step_count=Unbounded(shape=(1,), dtype=torch.int64)
)
self.done_spec = Composite(
    done=Binary(shape=(1,), dtype=torch.bool),
    terminated=Binary(shape=(1,), dtype=torch.bool),
    truncated=Binary(shape=(1,), dtype=torch.bool)
)
```

Code
Custom Environment - _step

- Holt die gewählte Action aus dem TensorDict
- Aktualisiert den State (Geld, welches Keyword wir kaufen)
- Berechnet den Reward
- Prüft, ob wir fertig sind
- Erhöht den aktuellen Schritt Index
- Aktualisert TensorDict mit den aktuellen Werten und gibt den nächsten State gemäss TorchRL vorgaben zurück

Code
Custom Environment – _step

- TorchRL: Aktueller State in SensorDict, neuer State als Rückgabewert

```
# tensordict is used from EnvBase later on, so we add the current state here
tensordict["done"] = torch.as_tensor(bool(terminated or truncated), dtype=torch.bool, device=self.device)
tensordict["observation"] = self.obs
tensordict["reward"] = torch.tensor(reward, dtype=torch.float32, device=self.device)
tensordict["step_count"] = torch.tensor(self.current_step-1, dtype=torch.int64, device=self.device)
tensordict["terminated"] = torch.tensor(bool(terminated), dtype=torch.bool, device=self.device)
tensordict["truncated"] = torch.tensor(bool(truncated), dtype=torch.bool, device=self.device)

# next as return value is also used by EnvBase and later added to tensordict by EnvBase
next_obs = TensorDict({
    "keyword_features": next_keyword_features,  # next pki for each keyword
    "cash": cash_normalized.clone().detach(),  # Current cash balance
    "holdings": self.holdings.clone()
}, batch_size=[])

next = TensorDict({
    "done": torch.tensor(bool(terminated or truncated), dtype=torch.bool, device=self.device),
    "observation": next_obs,
    "reward": torch.tensor(reward, dtype=torch.float32, device=self.device),
    "step_count": torch.tensor(self.current_step, dtype=torch.int64, device=self.device),
    "terminated": torch.tensor(bool(terminated), dtype=torch.bool, device=self.device),
    "truncated": torch.tensor(bool(truncated), dtype=torch.bool, device=self.device)
}, batch_size=tensordict.batch_size)


return next
```

Code – Custom Environment _compute_reward

- Ziel: Bester "Return on Ad Spend (ROAS)"

```
def _compute_reward(self, action, current_pki, action_idx, ad_roas):
    """Compute reward based on the selected keyword's metrics"""
    adjusted_reward = 0 if action_idx < self.num_keywords else 1 # encourage the agent to buy something
    if ad_roas > 0: # log(0) is undefined
        adjusted_reward = np.log(ad_roas)  ## Adjust reward based on ad_roas performance, scale it with log
    missing_rewards = []
    # Calculate the ad_roas we did not get because we chose another keyword
    for i in range(self.num_keywords):
        sample = current_pki.iloc[i]
        if action[i] == False:
            missing_rewards.append(sample["ad_roas"])
    # Adjust reward based on missing rewards to penalize the agent when not selecting keywords with
    # high(er) ROAS
    # clipping reduces the variance of the rewards
    return np.clip(adjusted_reward - np.mean(missing_rewards) * 0.2, -2, 2)
```

Code Policy

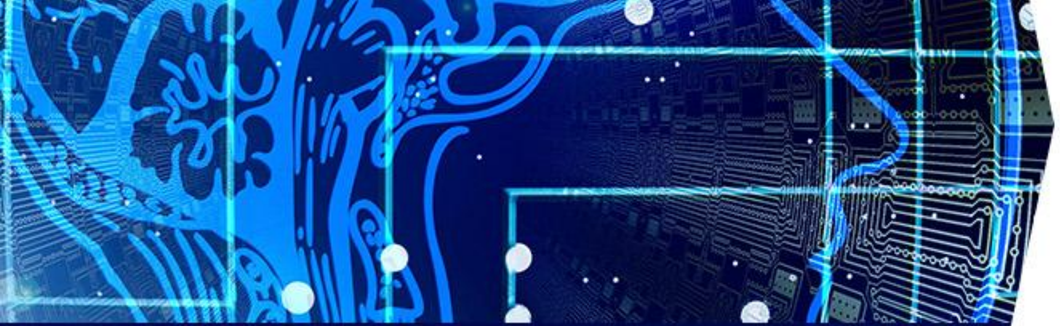
- Wir brauchen die Policy für Training, Test und Inference

```python
def create_policy(env, feature_dim, num_keywords, device):
    action_dim = env.action_spec.shape[-1]
    total_input_dim = feature_dim * num_keywords + 1 + num_keywords  # features per keyword + cash + holdings

    # Create the flattening module to combine our observation into one vector
    flatten_module = TensorDictModule(
        FlattenInputs(),
        in_keys=[("observation", "keyword_features"), ("observation", "cash"), ("observation", "holdings")],
        out_keys=["flattened_input"]
    )

    value_mlp = MLP(     # Create the value network
        in_features=total_input_dim,
        out_features=action_dim,
        num_cells=[256, 256, 128, 64],  # Deeper and wider architecture
        activation_class=nn.ReLU  # ReLU often performs better than Tanh
    )

    value_net = TensorDictModule(value_mlp, in_keys=["flattened_input"], out_keys=["action_value"])

    # Combine into the complete policy
    policy = TensorDictSequential(flatten_module, value_net, QValueModule(spec=env.action_spec))

    return policy.to(device)
```

Code
Training Loop

- Normaler TorchRL Trainingsloop

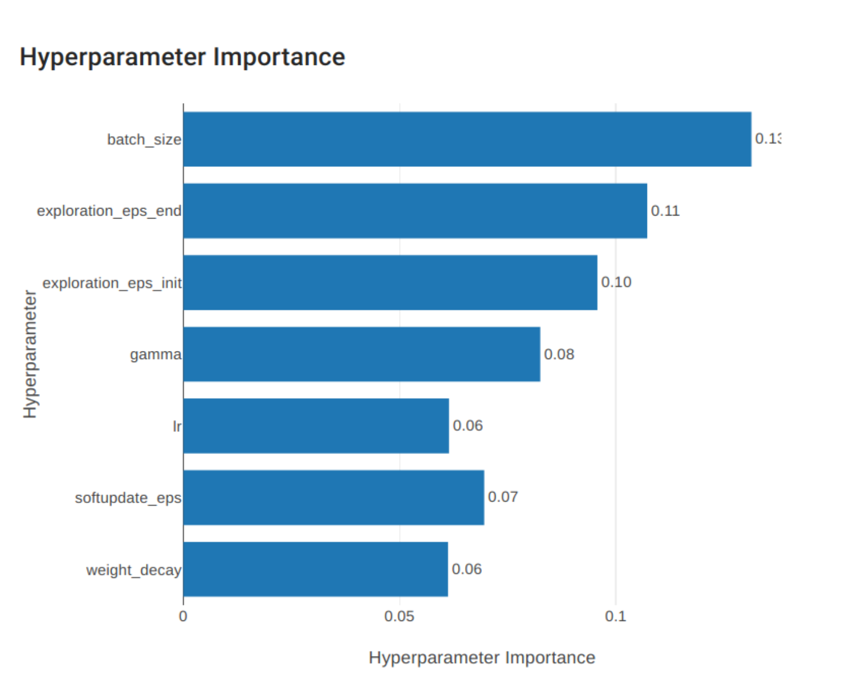
```
exploration_module = EGreedyModule(
    env.action_spec, annealing_num_steps=100_000, eps_init=exploration_eps_init, eps_end=exploration_eps_end
)
exploration_module = exploration_module.to(device)
policy_explore = TensorDictSequential(policy, exploration_module).to(device)


collector = SyncDataCollector(
    env,
    policy_explore,
    frames_per_batch=frames_per_batch,
    total_frames=-1,
    init_random_frames=init_rand_steps,
)
replay_buffer_size = 100_000
rb = ReplayBuffer(storage=LazyTensorStorage(replay_buffer_size))


loss = DQNLoss(value_network=policy, action_space=env.action_spec, delay_value=True).to(device)


optim = Adam(loss.parameters(), lr=lr, weight_decay=weight_decay)  # Add weight decay for regularization
updater = SoftUpdate(loss, eps=softupdate_eps)
```



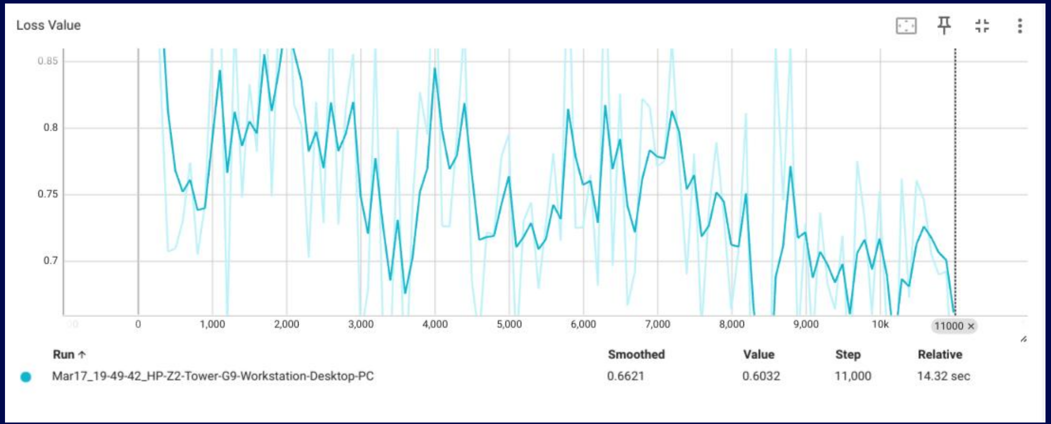
Optuna

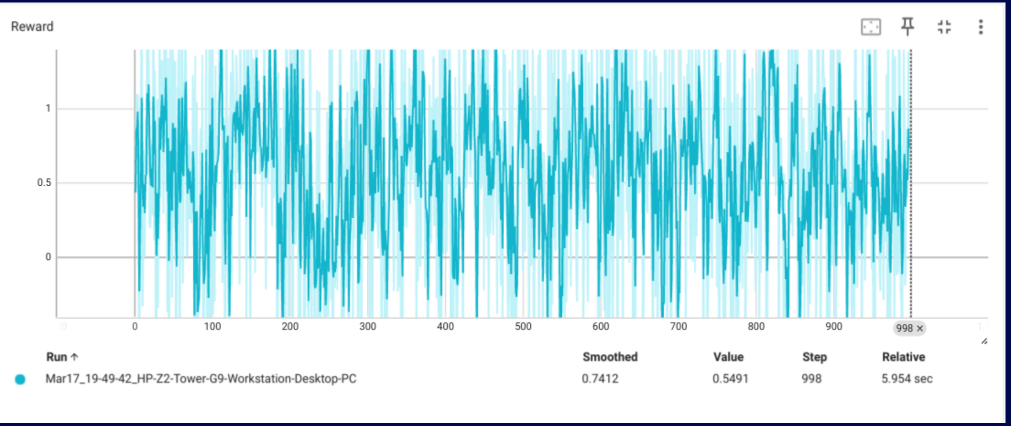
```
    # Learning rate for the optimizer
    # Batch size for training
),  # Initial value for epsilon in epsilon-greedy exploration
,   # Final value for epsilon in epsilon-greedy exploration
    # Soft update rate for target network
    # Discount factor for future rewards
    # Weight decay for regularization
```

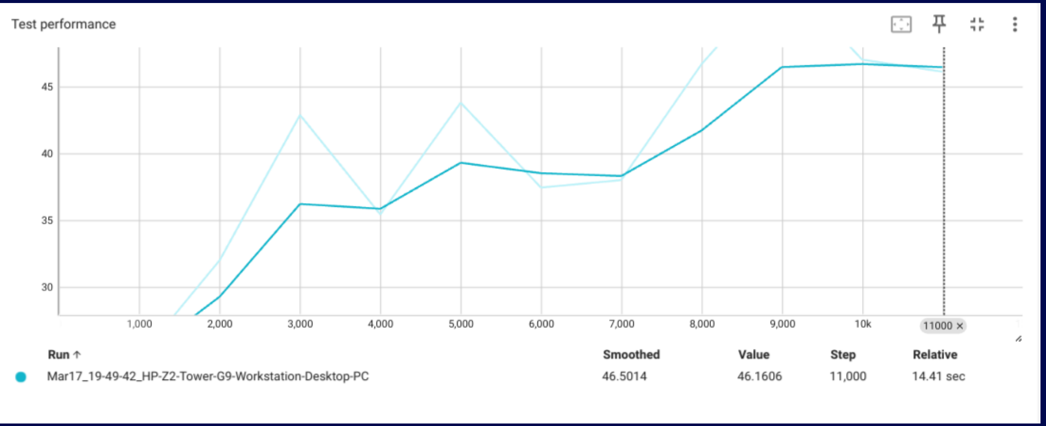
```python
# Sample hyperparameters
params = {
    'lr': trial.suggest_float('lr', 1e-4, 1e-2, log=True),
    'batch_size': trial.suggest_categorical('batch_size', [32, 64, 128, 256]),
    'exploration_eps_init': trial.suggest_float('exploration_eps_init', 0.5, 1.0),
    'exploration_eps_end': trial.suggest_float('exploration_eps_end', 0.01, 0.1),
    'softupdate_eps': trial.suggest_float('softupdate_eps', 0.9, 0.99),
    'gamma': trial.suggest_float('gamma', 0.9, 0.99),
    'weight_decay':  trial.suggest_float('weight_decay', 1e-6, 1e-4)
}
```



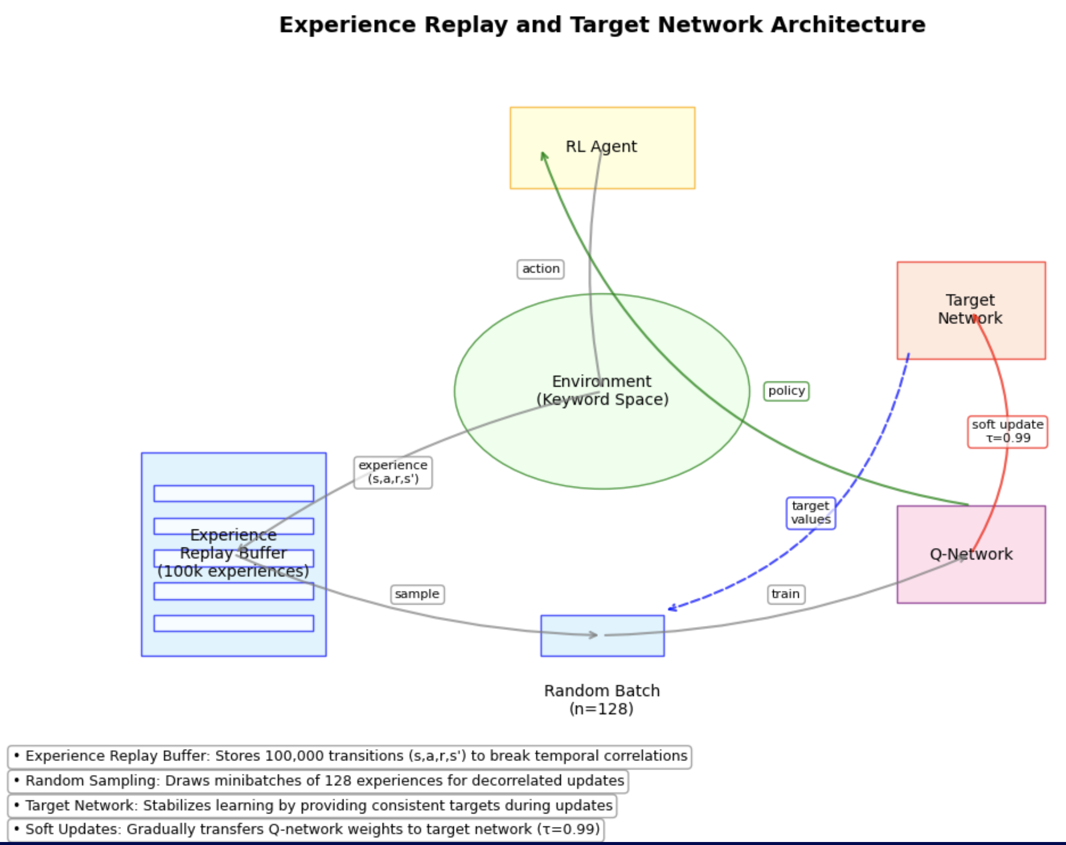
Tensorboard

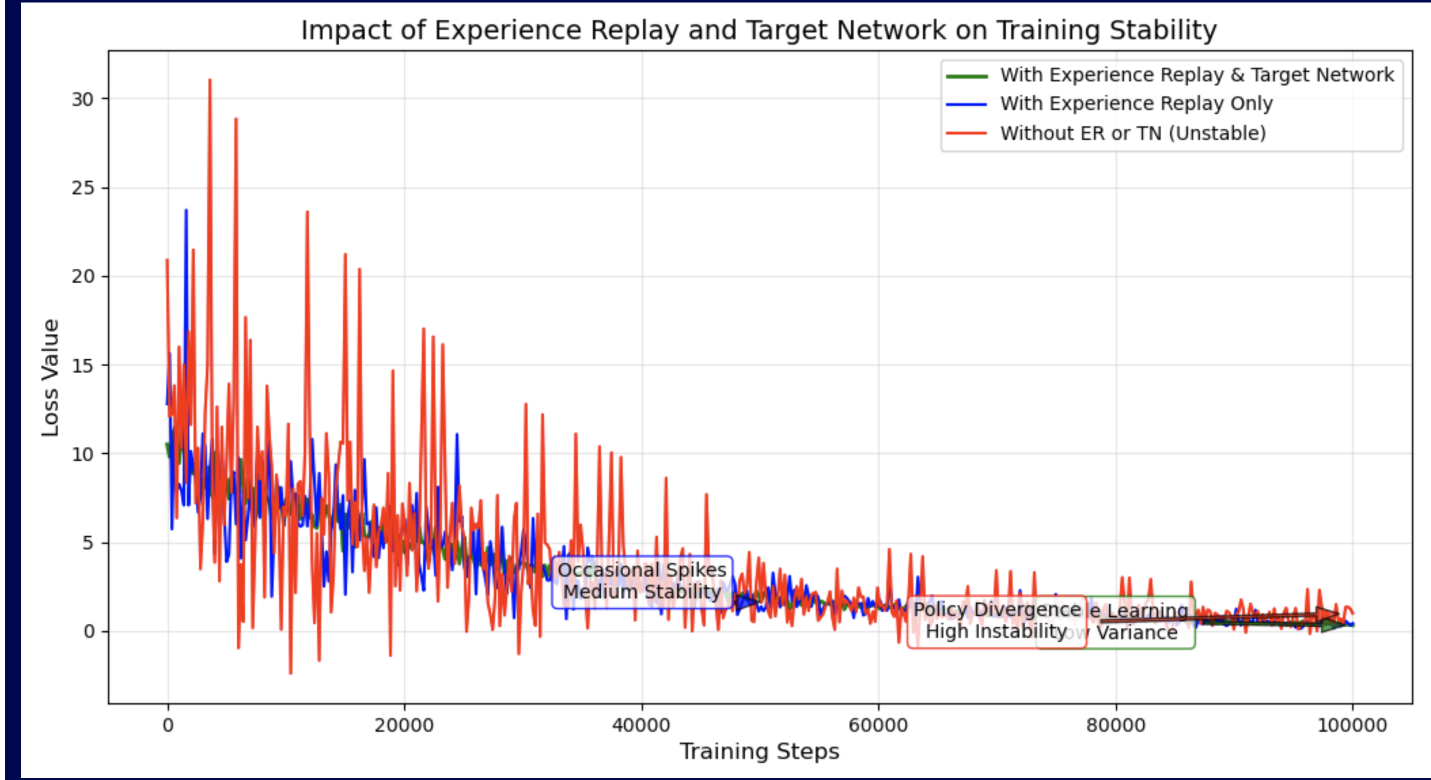




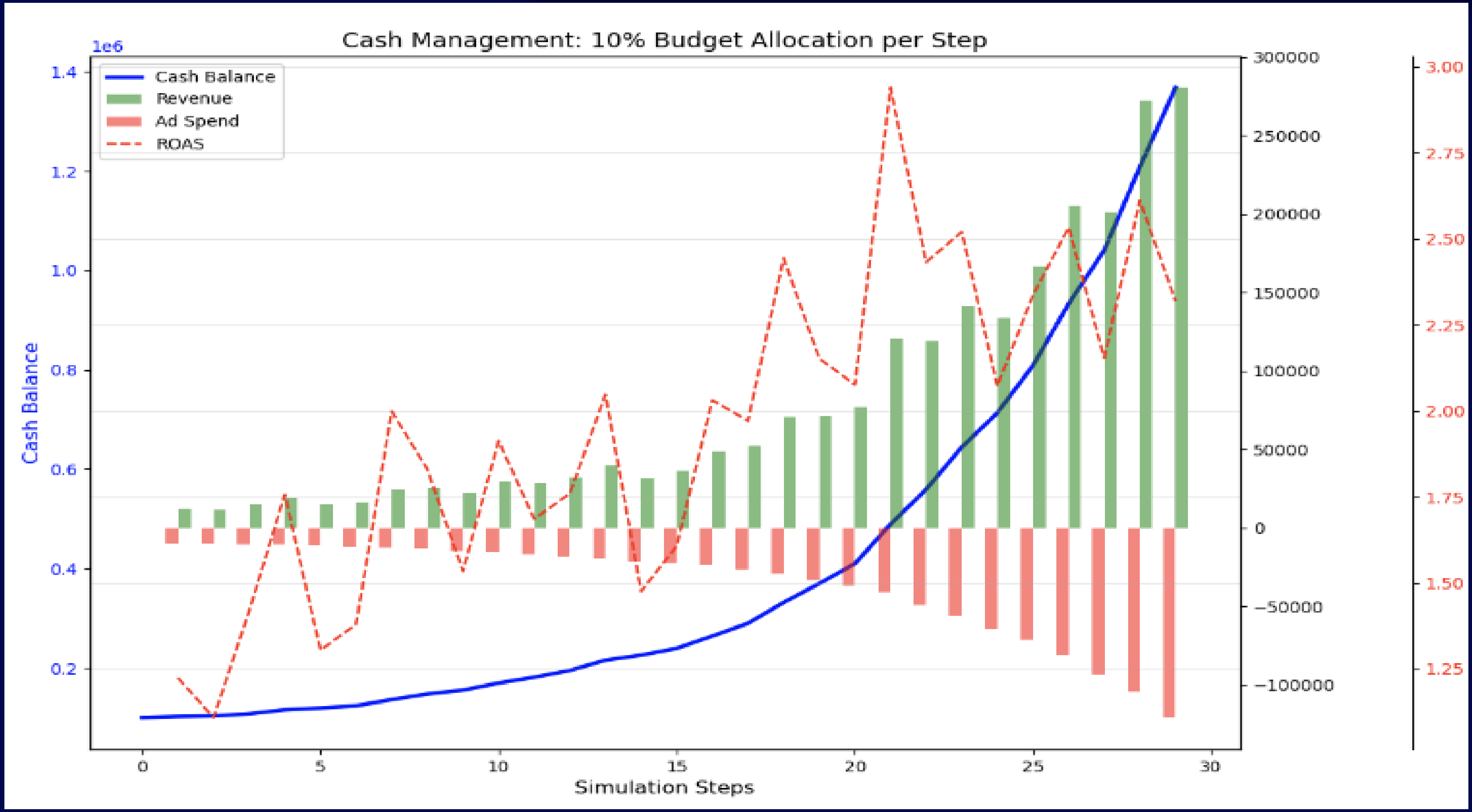


Data analysis (EO)

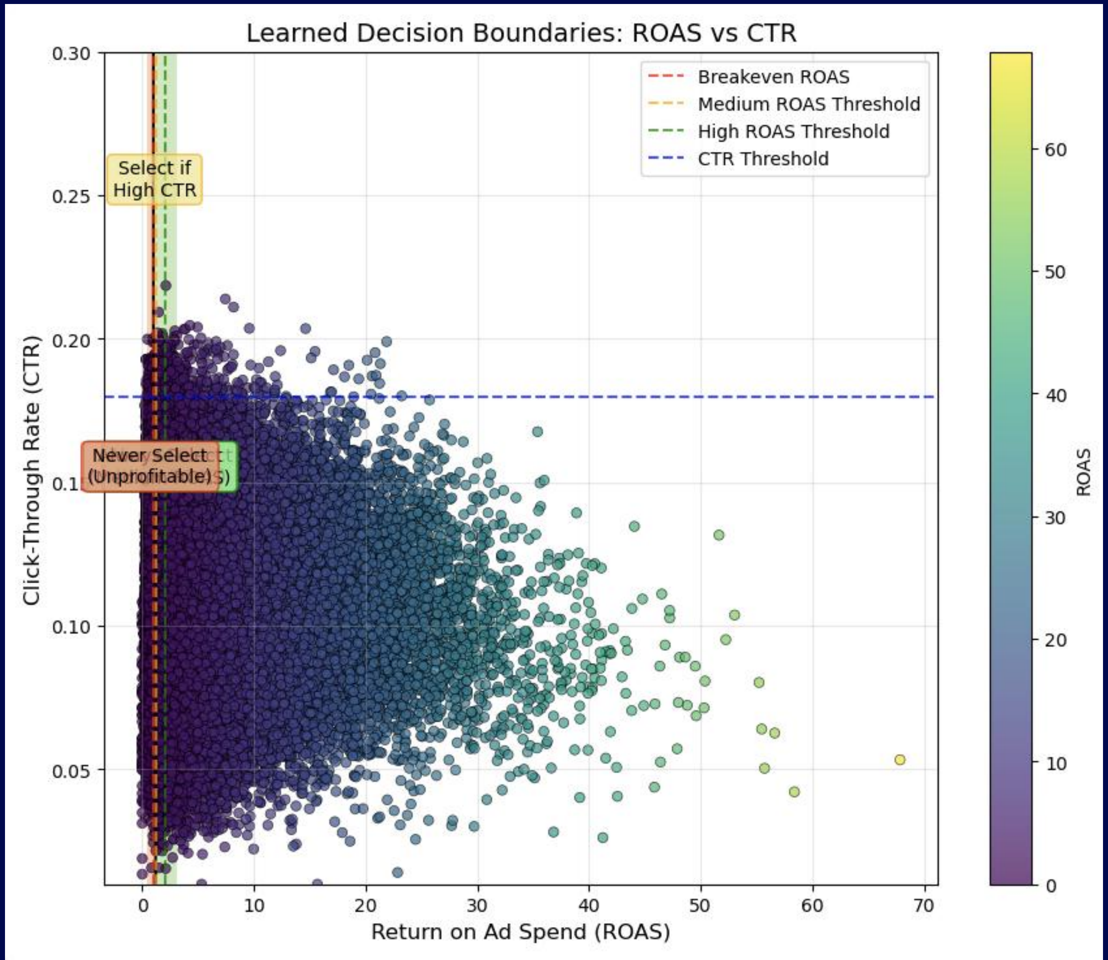


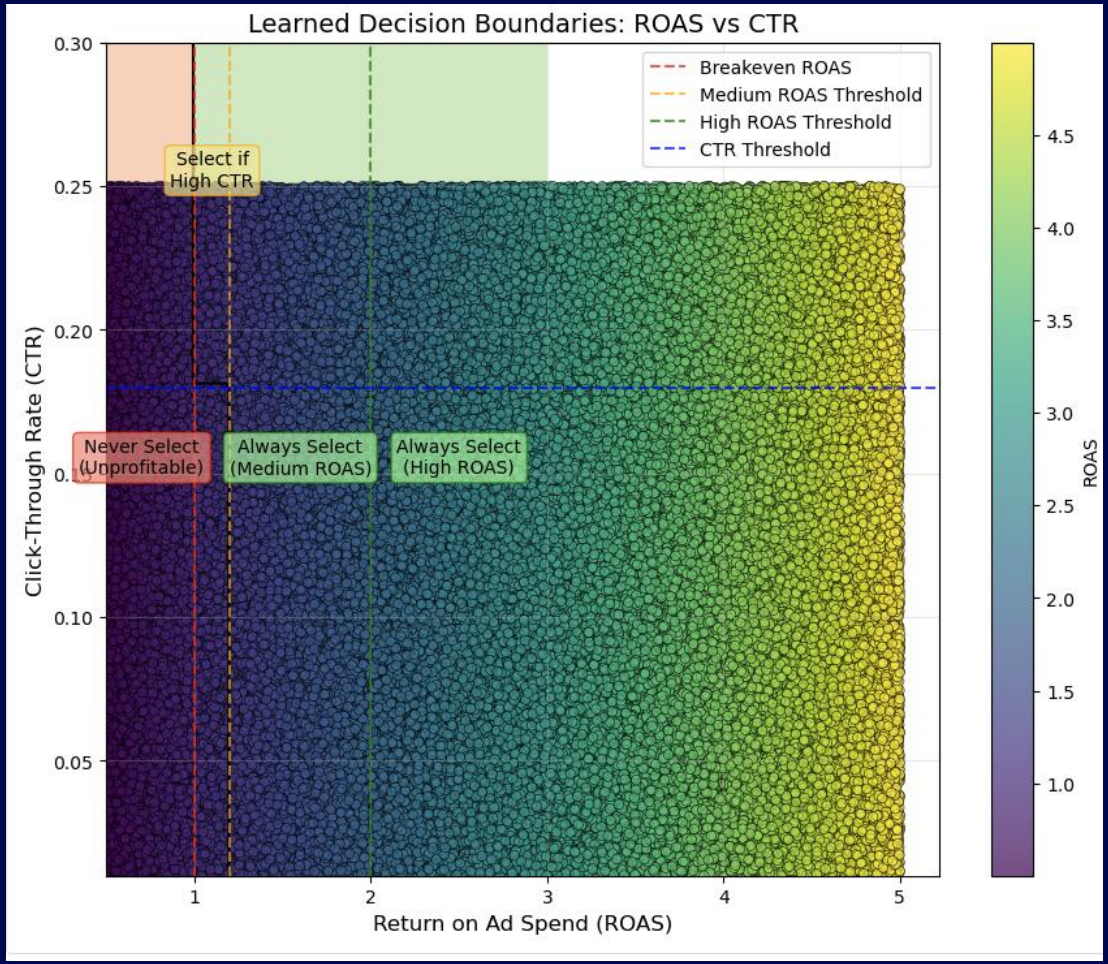


Cash Management Overview

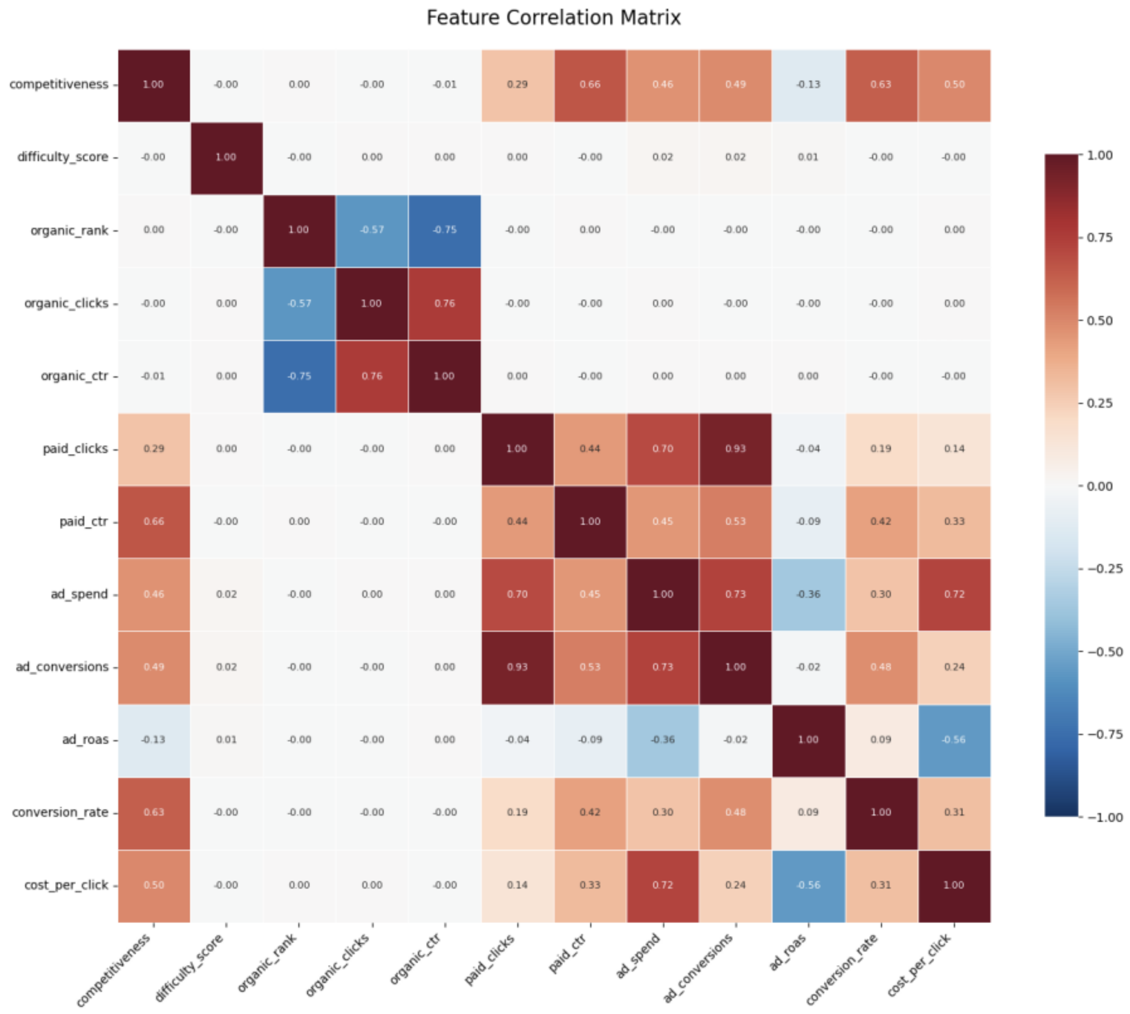


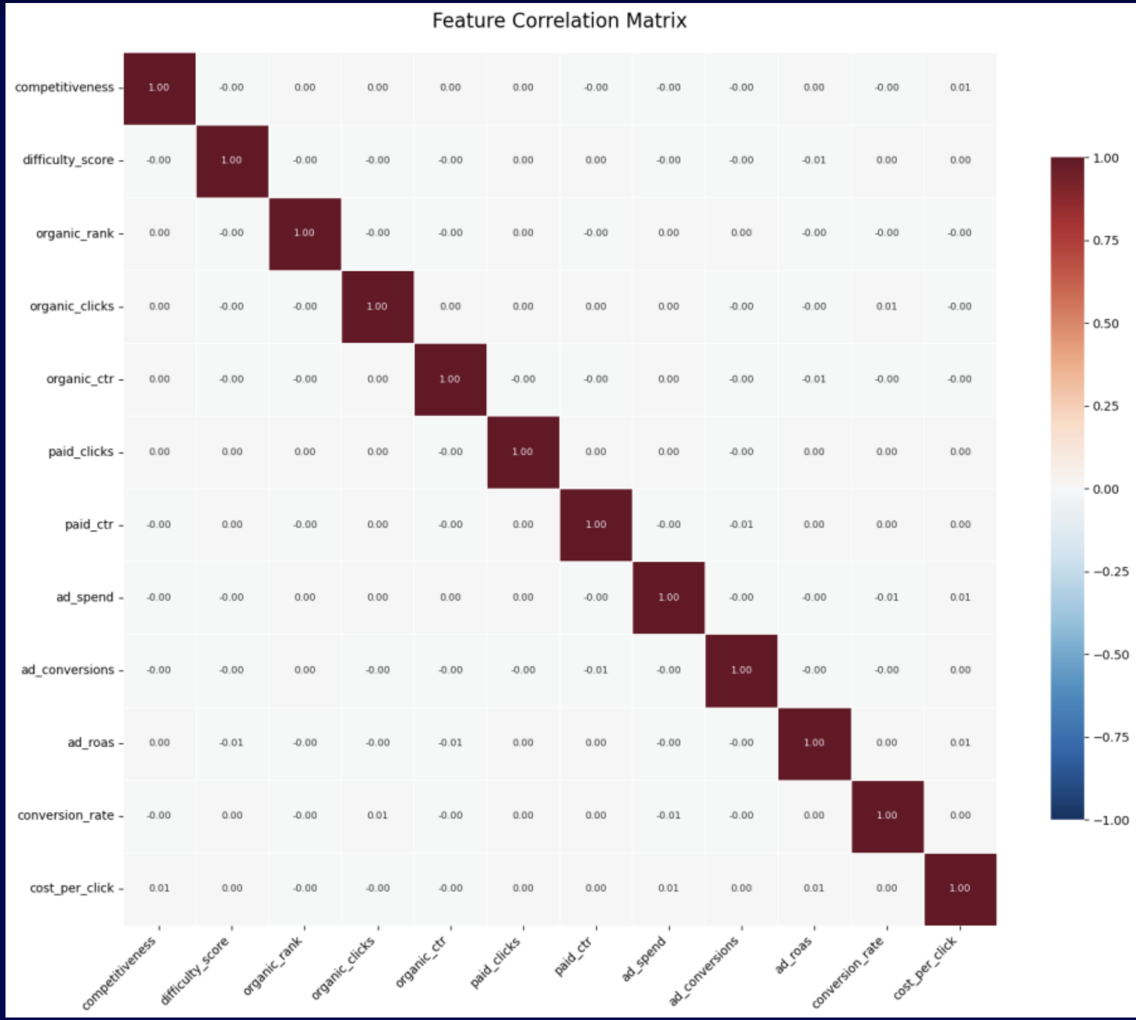
Comparison:
Our data vs "Ilja simple data"



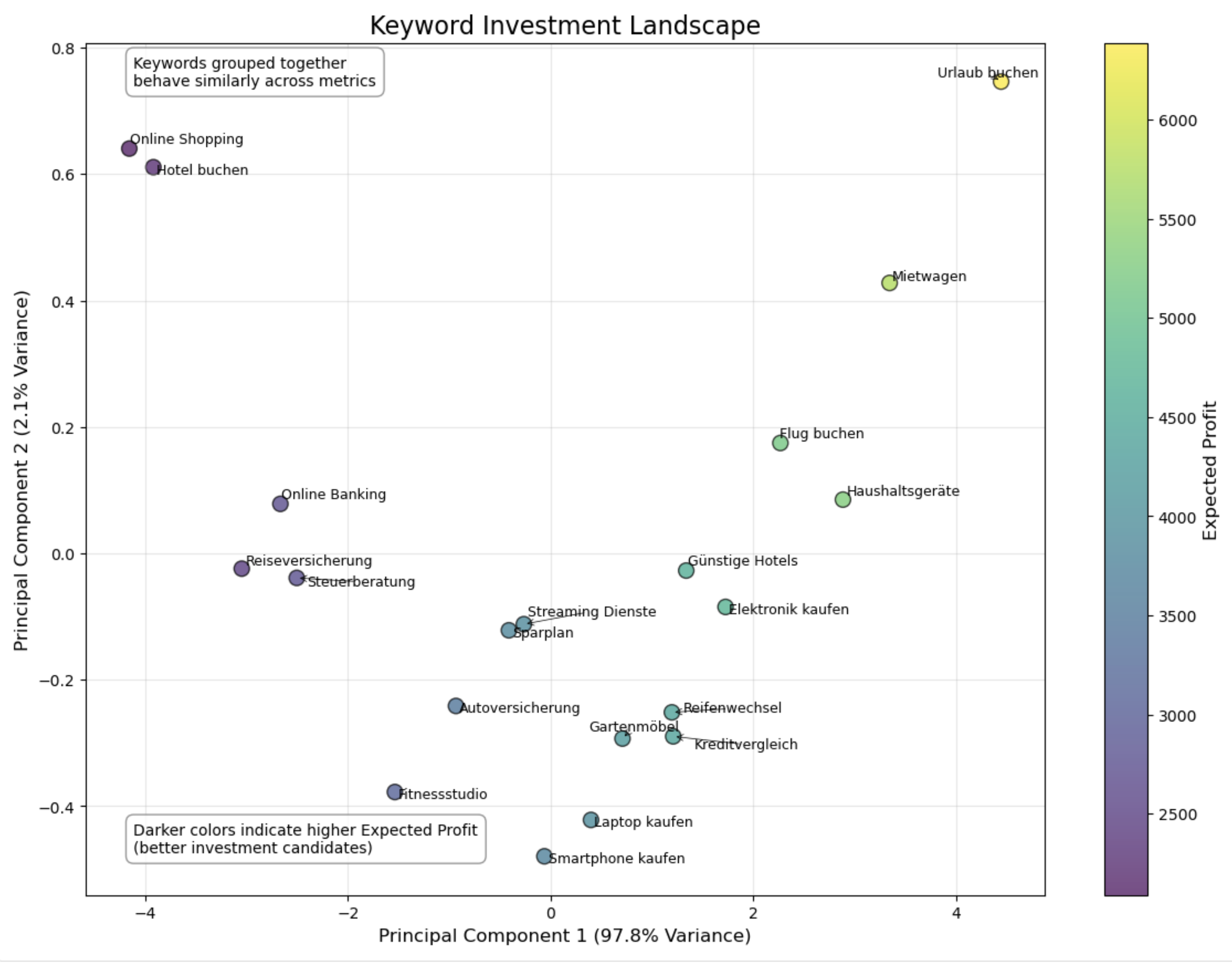


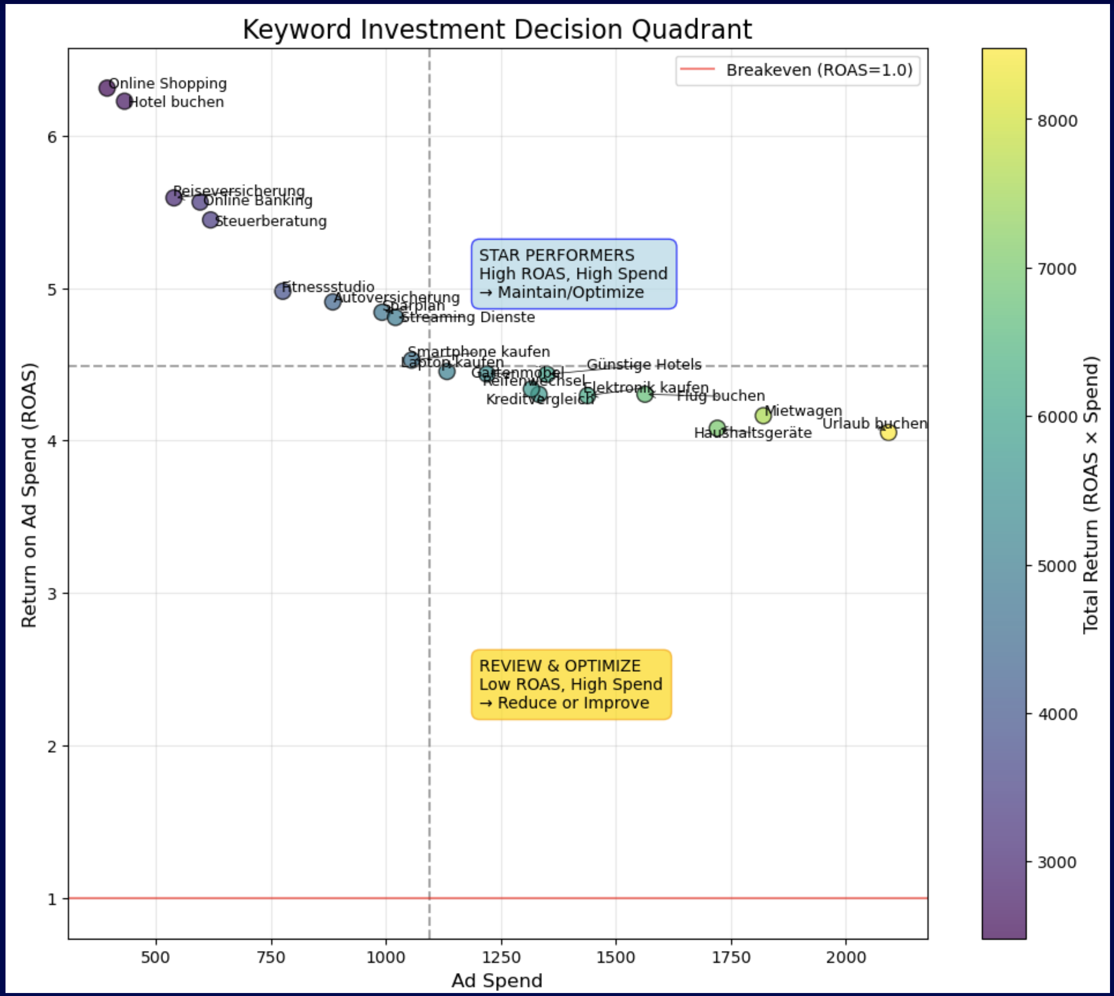
Comparison:
Our data vs "Ilja simple data"

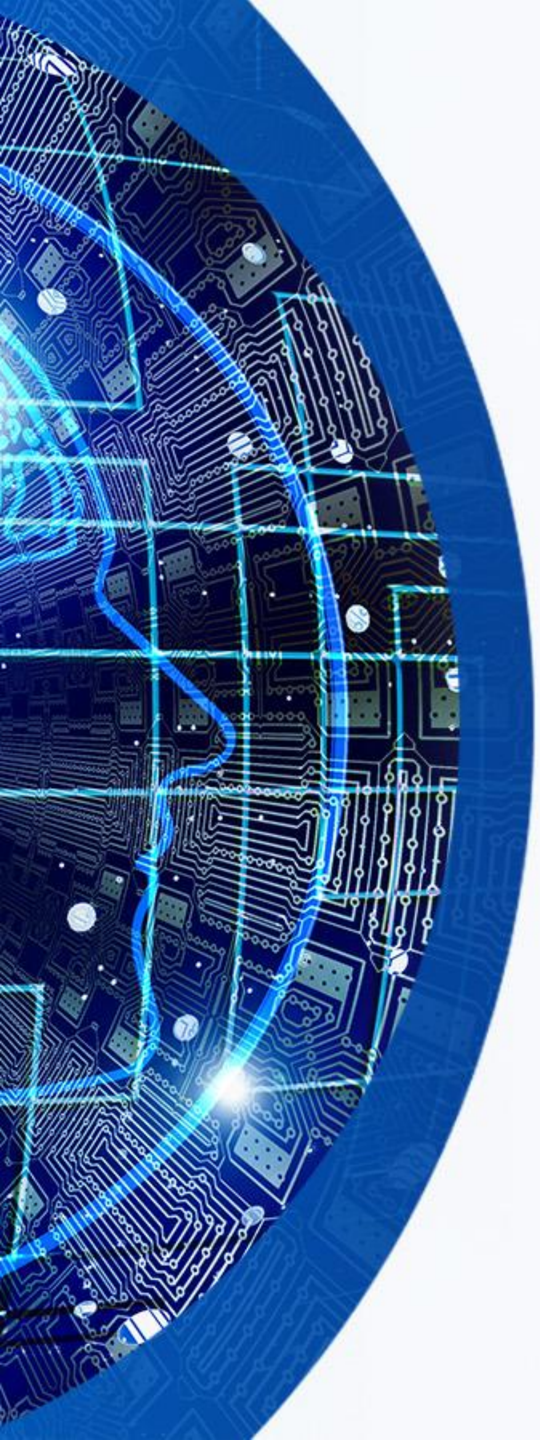




Which keywords to invest in?







Lessons Learned / Herausforderungen (MAC)

- Erfahrungen in einem „realen“ Business-Projekt
  → von der Idee via Daten und Training zur Analyse
- Daten: Gewisse offene Fragen hinsichtlich Datenqualität
- TorchRL Library
- Fehlendes Knowhow über "Digital Advertising" aufgebaut

GitHub- / Colab-Link

- Digital Advertising «Gruppe 2» on GitHub
- Colab-Notebook