In [1]:
```
import matplotlib.pyplot as plt
import numpy as np
import sklearn

plt.rcParams["font.family"] = "Times New Roman"
plt.rcParams["figure.dpi"] = 100.0
```

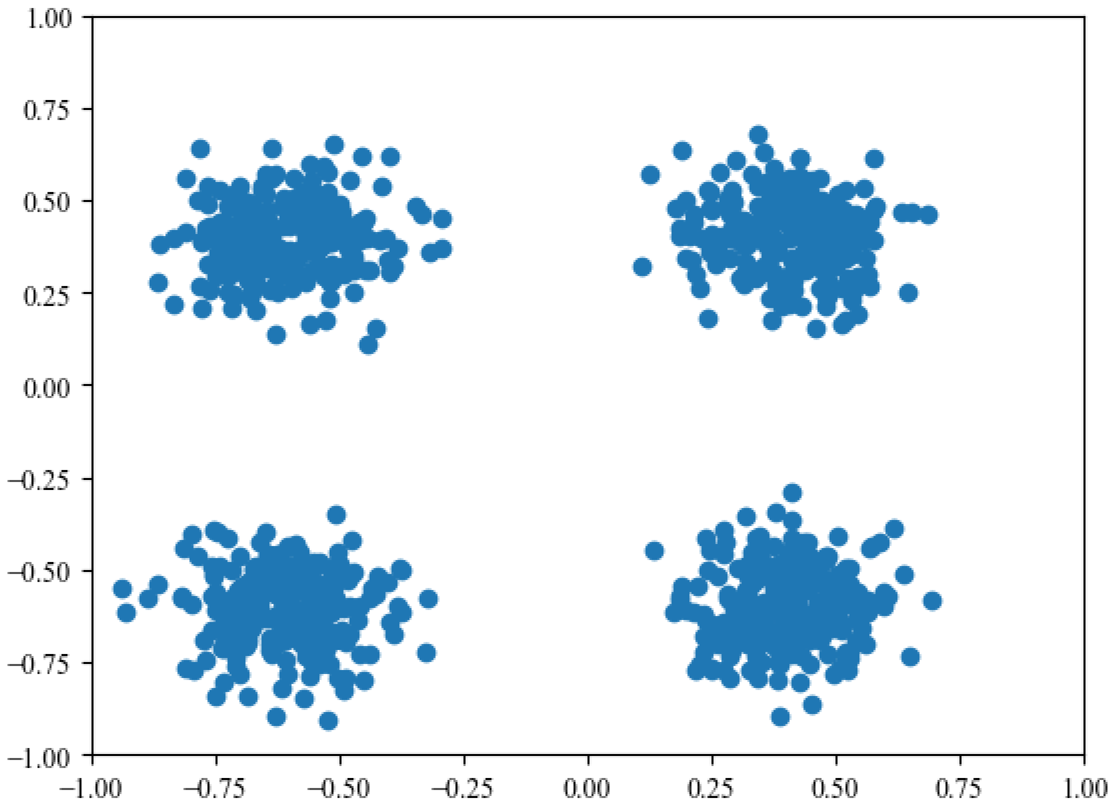
Environment

Mixture of Truncated Gaussians

In [2]:
```
from numpy.random import RandomState
from scipy.stats import truncnorm


def sample_mixture_truncnorm(n: int, random_state: RandomState = None) -> np
    if random_state is None:
        random_state = RandomState()

    scale = 0.1

    loc = -0.6
    a, b = (-1.0 - loc) / scale, (1.0 - loc) / scale
    pos_samples = truncnorm.rvs(
        a, b, loc=loc, scale=scale, size=2 * n, random_state=random_state
    )

    loc = 0.4
    a, b = (-1.0 - loc) / scale, (1.0 - loc) / scale
    neg_samples = truncnorm.rvs(
        a, b, loc=loc, scale=scale, size=2 * n, random_state=random_state
    )

    all_samples = np.concatenate([pos_samples, neg_samples], axis=None)
    random_state.shuffle(all_samples)

    return all_samples.reshape((-1, 2))[:n]
```

In [3]:
```
truncnorm_data = sample_mixture_truncnorm(1000)
plt.scatter(truncnorm_data[:, 0], truncnorm_data[:, 1])
# plt.scatter([0.4, 0.4, -0.6, -0.6], [0.4, -0.6, 0.4, -0.6])

ax = plt.gca()
ax.set_xlim(-1, 1)
ax.set_ylim(-1, 1)
```

Out[3]: (-1.0, 1.0)

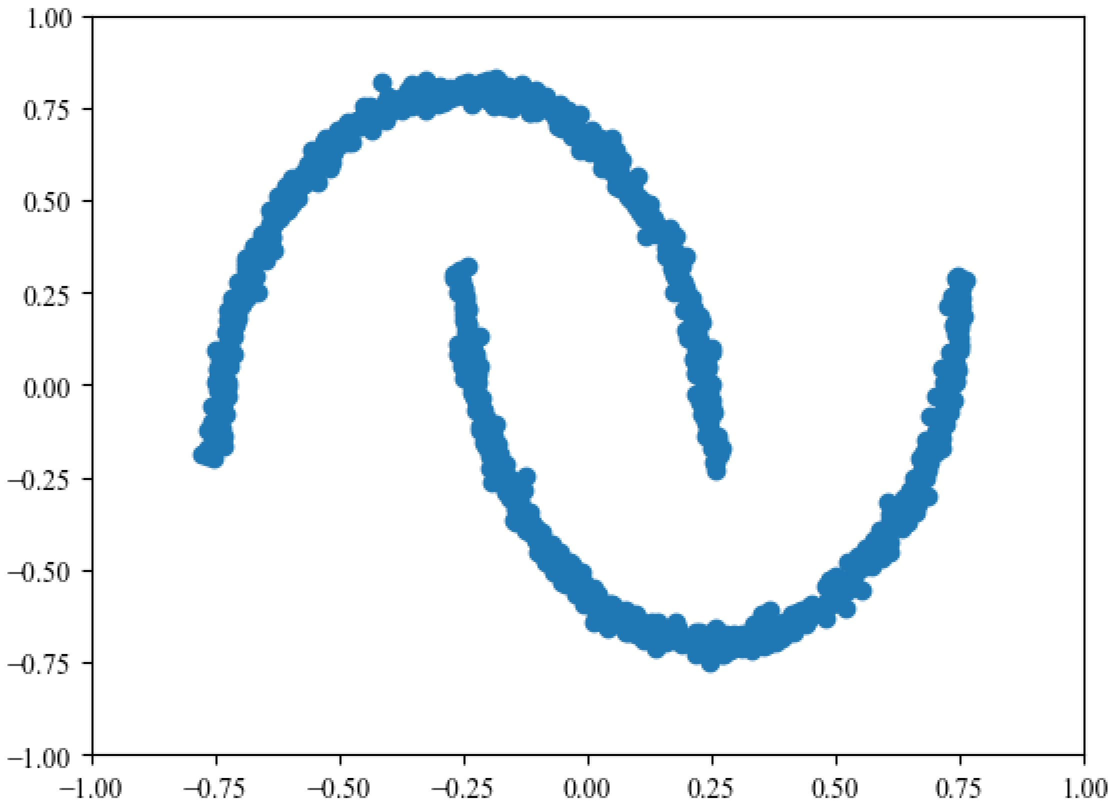


Two Moons Distribution

In [4]:
```
from sklearn import datasets


def sample_moons(size: int):
    samples, _ = datasets.make_moons(n_samples=size, shuffle=True, noise=0.0
    samples[:, 0] -= 0.5
    samples[:, 1] -= 0.2
    samples[:, 0] /= 2.0

    return samples
```

In [5]:
```
moons_data = sample_moons(1000)
plt.scatter(moons_data[:, 0], moons_data[:, 1])

ax = plt.gca()
ax.set_xlim(-1, 1)
ax.set_ylim(-1, 1)
```

Out[5]:
```
(-1.0, 1.0)
```

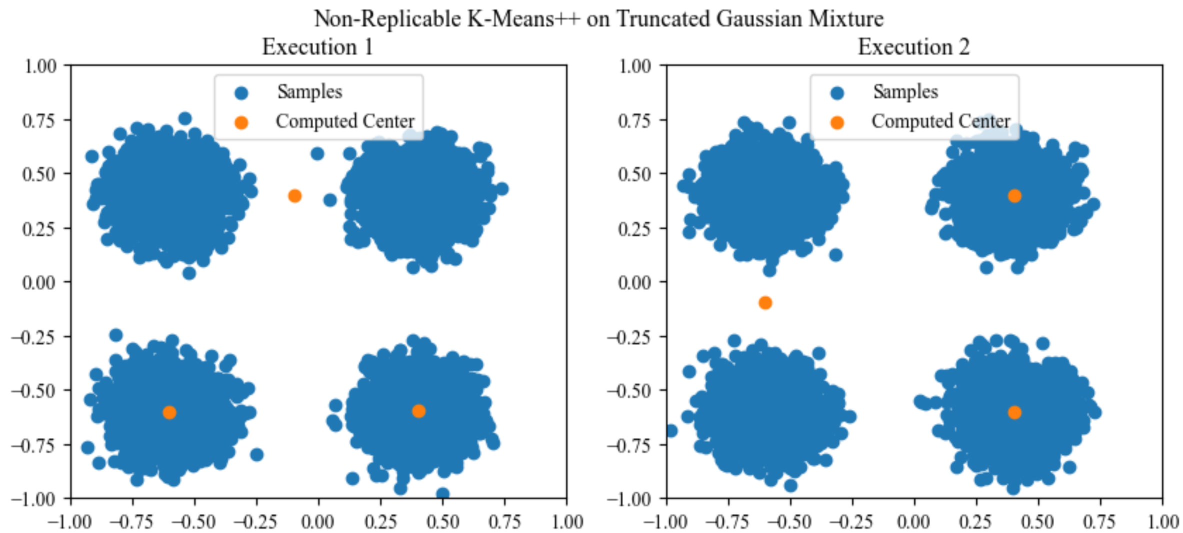


Naive K-Means++

In [6]:
```
from sklearn.cluster import KMeans

truncnorm_data1 = sample_mixture_truncnorm(500_00)
truncnorm_data2 = sample_mixture_truncnorm(500_00)
truncnorm_kmeans1 = KMeans(n_clusters=3).fit(truncnorm_data1)
truncnorm_kmeans2 = KMeans(n_clusters=3).fit(truncnorm_data2)

fig, ax = plt.subplots(1, 2, figsize=(10, 4))

truncnorm_data_indices1 = np.random.choice(
    truncnorm_data1.shape[0], 10000, replace=False
)

ax[0].scatter(
    truncnorm_data1[truncnorm_data_indices1, 0],
    truncnorm_data1[truncnorm_data_indices1, 1],
    label="Samples",
)
# ax[0].scatter([0.4, 0.4, -0.6, -0.6], [0.4, -0.6, 0.4, -0.6])
ax[0].scatter(
    truncnorm_kmeans1.cluster_centers_[:, 0],
    truncnorm_kmeans1.cluster_centers_[:, 1],
    label="Computed Center",
)

ax[0].set_xlim(-1, 1)
```

```
ax[0].set_ylim(-1, 1)
ax[0].legend()
ax[0].set_title("Execution 1")

truncnorm_data_indices2 = np.random.choice(
    truncnorm_data2.shape[0], 10000, replace=False
)

ax[1].scatter(
    truncnorm_data2[truncnorm_data_indices2, 0],
    truncnorm_data2[truncnorm_data_indices2, 1],
    label="Samples",
)
# ax[1].scatter([0.4, 0.4, -0.6, -0.6], [0.4, -0.6, 0.4, -0.6])
ax[1].scatter(
    truncnorm_kmeans2.cluster_centers_[:, 0],
    truncnorm_kmeans2.cluster_centers_[:, 1],
    label="Computed Center",
)

ax[1].set_xlim(-1, 1)
ax[1].set_ylim(-1, 1)
ax[1].legend()
ax[1].set_title("Execution 2")

fig.suptitle("Non-Replicable K-Means++ on Truncated Gaussian Mixture")

plt.savefig("truncnorm.pdf", format="pdf", bbox_inches="tight")

plt.show()
```

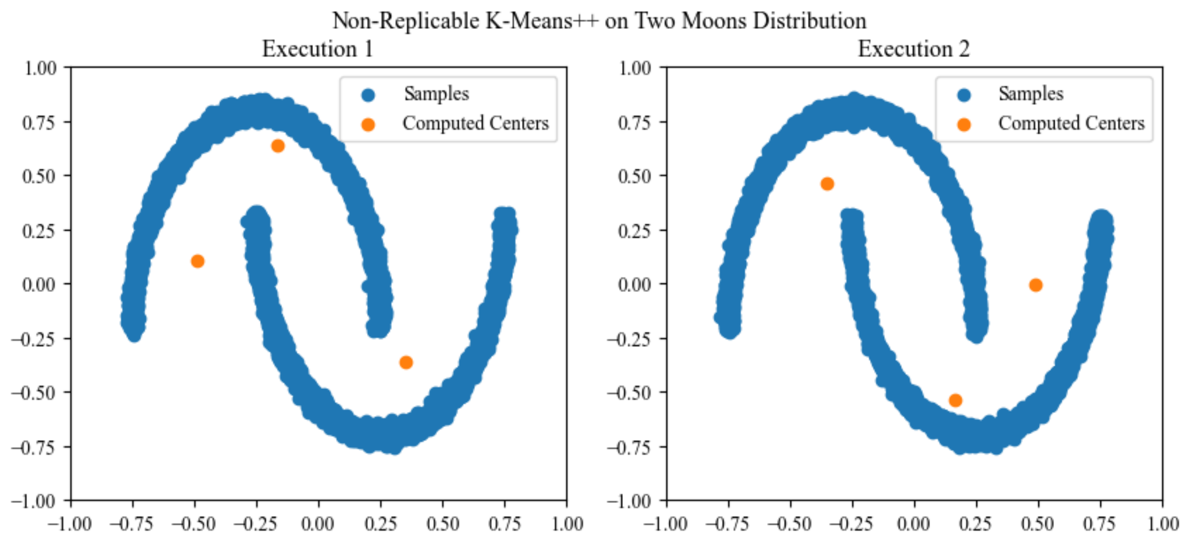
```
/Users/cz397/.mambaforge/envs/approx_matmul/lib/python3.10/site-packages/sk
learn/cluster/_kmeans.py:870: FutureWarning: The default value of `n_init`
will change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly
to suppress the warning
  warnings.warn(
/Users/cz397/.mambaforge/envs/approx_matmul/lib/python3.10/site-packages/sk
learn/cluster/_kmeans.py:870: FutureWarning: The default value of `n_init`
will change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly
to suppress the warning
  warnings.warn(
```



In [7]:
```
moons_data1 = sample_moons(500_00)
moons_data2 = sample_moons(500_00)
moons_kmeans1 = KMeans(n_clusters=3).fit(moons_data1)
moons_kmeans2 = KMeans(n_clusters=3).fit(moons_data2)

fig, ax = plt.subplots(1, 2, figsize=(10, 4))

moons_data_indices1 = np.random.choice(moons_data1.shape[0], 10000, replace=

ax[0].scatter(
    moons_data1[moons_data_indices1, 0],
    moons_data1[moons_data_indices1, 1],
    label="Samples",
)
ax[0].scatter(
    moons_kmeans1.cluster_centers_[:, 0],
    moons_kmeans1.cluster_centers_[:, 1],
    label="Computed Centers",
)

ax[0].set_xlim(-1, 1)
ax[0].set_ylim(-1, 1)
ax[0].legend()
ax[0].set_title("Execution 1")

moons_data_indices2 = np.random.choice(moons_data2.shape[0], 10000, replace=

ax[1].scatter(
    moons_data2[moons_data_indices2, 0],
    moons_data2[moons_data_indices2, 1],
    label="Samples",
)
ax[1].scatter(
    moons_kmeans2.cluster_centers_[:, 0],
    moons_kmeans2.cluster_centers_[:, 1],
    label="Computed Centers",
)

ax[1].set_xlim(-1, 1)
```

```
ax[1].set_ylim(-1, 1)
ax[1].legend()
ax[1].set_title("Execution 2")

fig.suptitle("Non-Replicable K-Means++ on Two Moons Distribution")

plt.savefig("moons.pdf", format="pdf", bbox_inches="tight")

plt.show()
```

```
/Users/cz397/.mambaforge/envs/approx_matmul/lib/python3.10/site-packages/sk
learn/cluster/_kmeans.py:870: FutureWarning: The default value of `n_init`
will change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly
to suppress the warning
  warnings.warn(
/Users/cz397/.mambaforge/envs/approx_matmul/lib/python3.10/site-packages/sk
learn/cluster/_kmeans.py:870: FutureWarning: The default value of `n_init`
will change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly
to suppress the warning
  warnings.warn(
```



Replicable K-Means++

Replicable Heavy Hitters

In [8]:
```
from numpy.random import RandomState


def get_idx_intersect(A, B):
    return (A[:, None] == B).all(-1).any(1)


# 1-dimensional
def r_heavy_hitters(
    sampler, thres: float, eps: float, rho: float, delta: float, random_stat
):
```

```
    print("r_heavy_hitters...")
    assert 0 < thres < 1
    assert 0 < eps < thres
    assert 0 < rho < 1
    assert 0 < delta <= rho/3

    if random_state is None:
        random_state = RandomState()

    n1 = int(np.ceil(np.log(2 / (delta * (thres - eps))) / (thres - eps)))
    print(n1)
    candidates = sampler(size=n1)
    candidates = np.unique(candidates, axis=0)

    n2 = int(
        np.ceil(
            (np.log(2 / delta) + (np.sqrt(n1) + 1) * np.log(2))  # * 648
            / (rho**2 * eps**2)
        )
    )
    print(n2)
    samples = sampler(size=n2)

    unique_samples, count = np.unique(samples, axis=0, return_counts=True)
    count = count.astype(float) / n2

    rand_thres = random_state.uniform(thres - 2*eps/3, thres-eps/3)
    print(count, rand_thres, n1 + n2)

    # magic intersection https://stackoverflow.com/a/67113105
    # get_idx_intersect(unique_samples, candidates)
    _, idx_intersect, _ = np.intersect1d(
        unique_samples, candidates, return_indices=True
    )

    unique_samples_intersect = unique_samples[idx_intersect]
    count_intersect = count[idx_intersect]
    return unique_samples_intersect[count_intersect >= rand_thres]
```

In [9]:

```
def toy_sampler(size: int, random_state: RandomState = None):
    if random_state is None:
        random_state = RandomState()

    return np.random.choice([0, 1, 2], p=[0.3, 0.3, 0.4], size=size).reshape


print(
    r_heavy_hitters(
        toy_sampler, thres=0.4, eps=0.1, rho=0.4, delta=0.1, random_state=Ra
    )
)
print(
    r_heavy_hitters(
        toy_sampler, thres=0.4, eps=0.1, rho=0.4, delta=0.1, random_state=Ra
```

```
    )
)
```

```
r_heavy_hitters...
14
3927
[0.29182582 0.31703591 0.39113827] 0.3478664967380668 3941
[[2]]
r_heavy_hitters...
14
3927
[0.29666412 0.29717341 0.40616246] 0.3478664967380668 3941
[[2]]
```

Replicable Quad Tree

In [10]:

```
from itertools import product


class QuadTreeNode:
    offsets = [np.array([dx, dy]) for dx, dy in product([-1.0, 1.0], repeat=

    def __init__(
        self, point: np.array, radius: float, is_heavy: bool = False, parent
    ):
        self.point = point
        self.radius = radius
        self.is_heavy = is_heavy
        self.children = [None] * 4
        self.parent = parent

    def get_heavy_nodes(self):
        heavy_nodes = []

        def _explore(node):
            if not node.is_heavy:
                return

            heavy_nodes.append(node.point.reshape((1, -1)))
            for child in node.children:
                _explore(child)

        _explore(self)
        return np.concatenate(heavy_nodes, axis=0)

    def get_leaves(self):
        leaves = []

        # return true if found node
        def _explore(node) -> bool:
            has_heavy_child = False
            for child in node.children:
                if child is not None and child.is_heavy:
                    _explore(child)
                    has_heavy_child = True
```

```
            if not has_heavy_child:
                leaves.append(node.point.reshape((1, -1)))

            return has_heavy_child

        _explore(self)
        return np.concatenate(leaves, axis=0)
```

In [11]:
```
def make_children(nodes):
    child_nodes = []
    for node in nodes:
        radius = node.radius
        for idx, d in enumerate(QuadTreeNode.offsets):
            next_point = node.point + d * radius / 2
            child_node = QuadTreeNode(next_point, radius / 2, parent=node)

            node.children[idx] = child_node
            child_nodes.append(child_node)

    return child_nodes


def get_idx_sampler(sampler, nodes):
    def _idx_sampler(size: int):
        samples = sampler(size)
        idx_samples = [len(nodes)] * size
        for i in range(size):
            for j in range(len(nodes)):
                if (
                    np.linalg.norm(samples[i] - nodes[j].point, ord=np.inf)
                    <= nodes[j].radius
                ):
                    idx_samples[i] = j
        return idx_samples

    return _idx_sampler
```

In [12]:
```
from numpy.random import RandomState


def r_quad_tree(
    sampler,
    k: int,
    eps: float,
    rho: float,
    delta: float,
    Gamma: float,
    beta: float,
    Delta: float = np.sqrt(2),
    skip_layers: int = 1,
    random_state: RandomState = None,
):
    assert 0 < eps < 1
    assert 0 < rho < 1
```

```
    t = 3  # int(np.ceil(1 / 2 * np.log(5 * Delta**2 / (eps * Gamma)) + 1))
    M = (Delta / eps) ** 2  # * 2**10
    gamma = eps / (t * k * M * Delta**2)  # / 20
    print(t, M, gamma)

    # build quad-tree
    root = QuadTreeNode(point=np.array([0.0, 0.0]), radius=1.0, is_heavy=Tru
    H = [root]
    i = 1
    while H:
        print(i)
        if (2 ** (-i + 1) * Delta) ** 2 <= eps * Gamma / 5:
            break

        child_nodes = make_children(H)

        if i <= skip_layers:  # skip first few layers
            heavy_hitters = range(len(child_nodes))
        else:
            idx_sampler = get_idx_sampler(sampler, child_nodes)
            thres = gamma * Gamma * 2 ** (2 * i)
            heavy_hitters = r_heavy_hitters(
                idx_sampler,
                thres=thres,
                eps=thres / 2,
                rho=rho / t,
                delta=delta / t,
                random_state=random_state,
            )

        H = []
        for idx in heavy_hitters:
            if idx < len(child_nodes):
                child_nodes[idx].is_heavy = True
                H.append(child_nodes[idx])

        i += 1

    return root
```

In [13]:

```
root1 = r_quad_tree(
    sample_mixture_truncnorm,
    k=3,
    eps=0.99,
    rho=0.4,
    delta=0.1,
    Gamma=0.5,
    beta=1.0,
    Delta=np.sqrt(2),
    random_state=RandomState(2),
)

root2 = r_quad_tree(
    sample_mixture_truncnorm,
    k=3,
```

```
    eps=0.99,
    rho=0.4,
    delta=0.1,
    Gamma=0.5,
    beta=1.0,
    Delta=np.sqrt(2),
    random_state=RandomState(2),
)
```

```
3 2.040608101214162 0.026952749999999999
1
2
r_heavy_hitters...
59
48935
[0.17533463 0.03373863 0.03523041 0.00651885 0.17631552 0.03416777
 0.03300296 0.00676407 0.17654031 0.0330234  0.03261469 0.00606928
 0.1796056  0.03132727 0.03355472 0.00619189] 0.15941634879827712 48994
3
r_heavy_hitters...
12
2175
[0.00137931 0.01195402 0.01149425 0.15770115 0.00137931 0.00965517
 0.01471264 0.14942529 0.00183908 0.01655172 0.00873563 0.15402299
 0.00137931 0.01241379 0.01103448 0.15586207 0.28045977] 0.5787188439727955
2187
3 2.040608101214162 0.026952749999999999
1
2
r_heavy_hitters...
59
48935
[0.17654031 0.03306427 0.03365689 0.00604884 0.1792582  0.03292122
 0.03455604 0.00692756 0.17976908 0.03288035 0.03304383 0.0063758
 0.17390416 0.03275774 0.0323899  0.00590579] 0.15941634879827712 48994
3
r_heavy_hitters...
12
2175
[0.00091954 0.01747126 0.01011494 0.15172414 0.00183908 0.01149425
 0.01471264 0.14252874 0.00137931 0.01103448 0.01333333 0.1554023
 0.00091954 0.01471264 0.01793103 0.14206897 0.29241379] 0.5787188439727955
2187
```

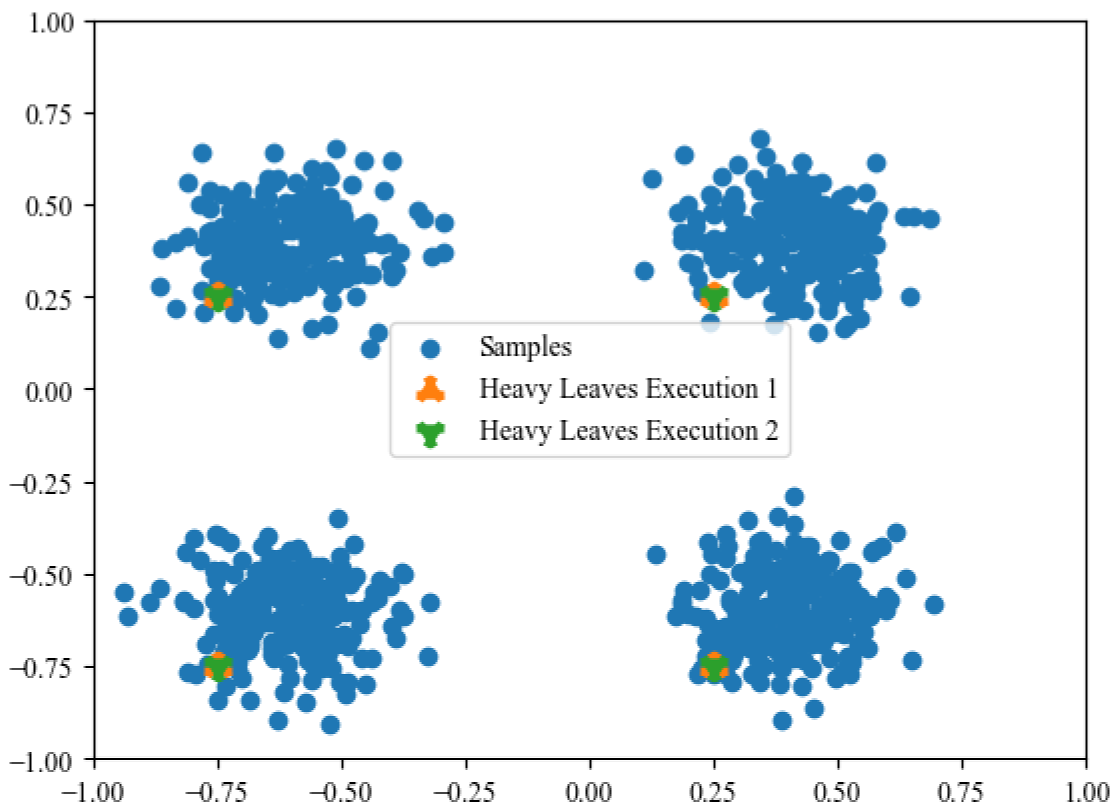
In [14]:

```
heavy_nodes1 = root1.get_leaves()
heavy_nodes2 = root2.get_leaves()
```

In [15]:

```
plt.scatter(truncnorm_data[:, 0], truncnorm_data[:, 1], label="Samples")
# plt.scatter([0.4, 0.4, -0.6, -0.6], [0.4, -0.6, 0.4, -0.6], label="Mixture
plt.scatter(
    heavy_nodes1[:, 0],
    heavy_nodes1[:, 1],
    marker="1",
    linewidths=10,
    label="Heavy Leaves Execution 1",
)
```

```
plt.scatter(
    heavy_nodes1[:, 0],
    heavy_nodes1[:, 1],
    marker="2",
    linewidths=10,
    label="Heavy Leaves Execution 2",
)

ax = plt.gca()
ax.set_xlim(-1, 1)
ax.set_ylim(-1, 1)

plt.legend()
plt.show()
```



Replicable Probability Mass Estimation

In [16]:
```
# eps should be a power of 10 e.g. 1e-3
def r_prob_mass(sampler, N: int, rho: float, eps: float, delta: float, rando
    assert 0 < rho < 1
    assert 0 < eps < 1
    assert 0 < delta < rho/3

    alpha = 2 * eps / (rho - 2*delta + 1)
    eps_prime = eps * (rho - 2*delta) / (rho + 1 - 2*delta)
    n = int(np.ceil((np.log(1/delta) + N * np.log(2)) / (eps**2 * (rho - 2*c
    decimals = int(np.log10(1 / eps))
    print(alpha, eps_prime, n, decimals)
```

```
    samples = sampler(size=n)
    unique_samples, count = np.unique(samples, axis=0, return_counts=True)
    len(unique_samples) <= N
    count = count.astype(float) / n

    offset = random_state.uniform(low=0.0, high=alpha, size=len(unique_sampl
    rounded_count = np.around(count - offset, decimals=decimals) + offset

    # normalize estimates
    return unique_samples, rounded_count - (rounded_count.sum() - 1.0) / ler
        unique_samples
    )
```

In [17]:
```
samples1, mass1 = r_prob_mass(
    toy_sampler, N=3, rho=0.4, eps=0.1, delta=0.1, random_state=RandomState(
)
samples2, mass2 = r_prob_mass(
    toy_sampler, N=3, rho=0.4, eps=0.1, delta=0.1, random_state=RandomState(
)

assert np.isclose(mass1.sum(), 1.0)
assert np.isclose(mass2.sum(), 1.0)

print(samples1, mass1)
print(samples2, mass2)
```

```
0.16666666666666669 0.016666666666666667 21912 1
0.16666666666666669 0.016666666666666667 21912 1
[[0]
 [1]
 [2]] [0.28313339 0.31478862 0.40207799]
[[0]
 [1]
 [2]] [0.28313339 0.31478862 0.40207799]
```

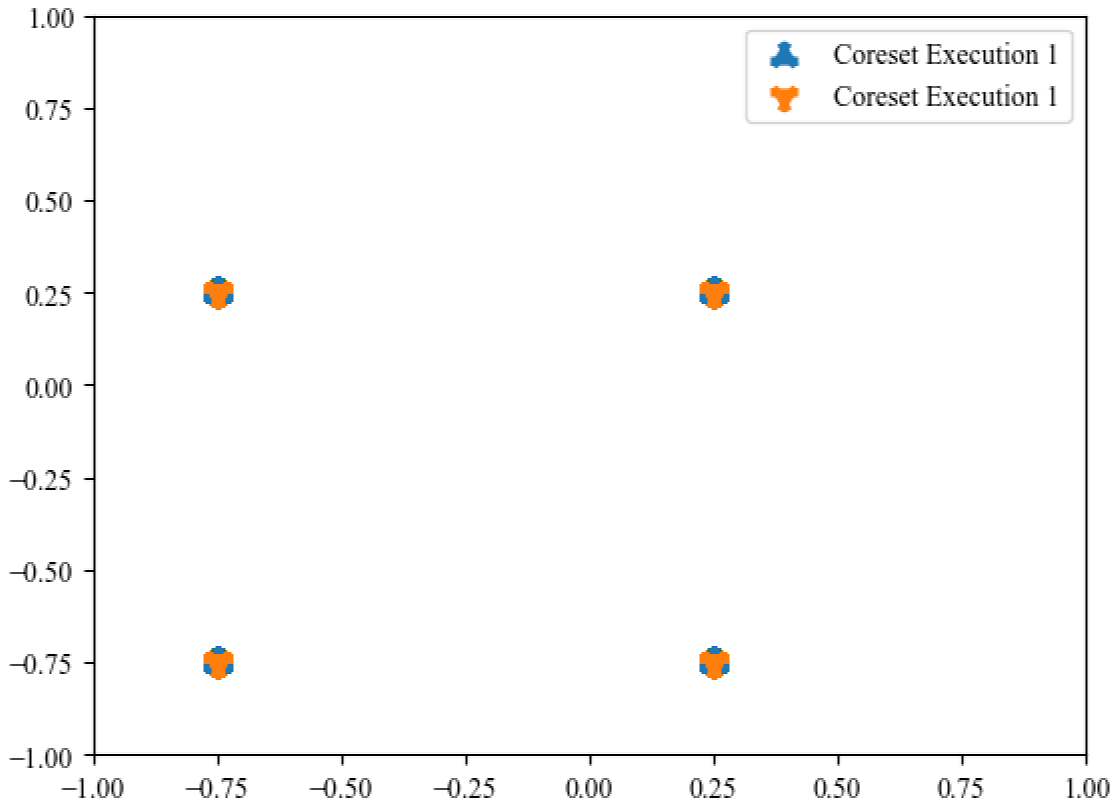
Replicable Coreset

In [18]:
```
def get_child_idx(node, point):
    if point[0] < node.point[0]:
        if point[1] < node.point[1]:
            return 0
        else:
            return 1
    else:
        if point[1] < node.point[1]:
            return 2
        else:
            return 3


def quad_tree_round(point, root: QuadTreeNode):
    output = np.array([0.0, 0.0])

    node = root
    while node is not None:
```

```
        child_idx = get_child_idx(node, point)

        if node.children[child_idx] is not None and node.children[child_idx]
            node = node.children[child_idx]
            output = node.point
            continue

        new_node = None
        for idx in range(len(QuadTreeNode.offsets)):
            if node.children[idx] is not None and node.children[idx].is_heav
                new_node = node.children[idx]
                output = new_node.point
                break

        node = new_node

    return output


def make_quad_tree_sampler(sampler, root: QuadTreeNode):
    def _quad_tree_sampler(size: int):
        samples = sampler(size)
        for i in range(len(samples)):
            samples[i] = quad_tree_round(samples[i], root)

        return samples

    return _quad_tree_sampler
```

In [19]:

```
quad_tree_sampler1 = make_quad_tree_sampler(sample_mixture_truncnorm, root1)
quad_tree_sampler2 = make_quad_tree_sampler(sample_mixture_truncnorm, root2)

samples1 = quad_tree_sampler1(1000)
samples2 = quad_tree_sampler2(1000)

plt.scatter(
    samples1[:, 0],
    samples1[:, 1],
    marker="1",
    linewidths=10,
    label="Coreset Execution 1",
)
plt.scatter(
    samples2[:, 0],
    samples2[:, 1],
    marker="2",
    linewidths=10,
    label="Coreset Execution 1",
)

ax = plt.gca()
ax.set_xlim(-1, 1)
ax.set_ylim(-1, 1)
plt.legend()
plt.show()
```



In [20]:
```
def r_coreset(
    sampler,
    k: int,
    eps: float,
    rho: float,
    delta: float,
    Gamma: float,
    beta: float,
    Delta: float = np.sqrt(2),
    skip_layers: int = 1,
    random_state: RandomState = None,
):
    assert 0 < eps < 1
    assert 0 < rho < 1
    assert 0 < delta < rho/3

    if random_state is None:
        random_state = RandomState()

    root = r_quad_tree(
        sampler,
        k=k,
        eps=eps,
        rho=rho,
        delta=delta,
        Gamma=Gamma,
        beta=beta,
        Delta=Delta,
        skip_layers=skip_layers,
        random_state=random_state,
```

```
    )

    N = len(root.get_leaves())

    quad_tree_sampler = make_quad_tree_sampler(sampler, root)
    coreset, mass = r_prob_mass(
        quad_tree_sampler, N=N, rho=0.1, eps=0.1, delta=0.01, random_state=r
    )

    return coreset, mass
```

Replicable K-Means++

In [21]:
```
class MemorizedSampler:
    def __init__(self, sampler):
        self.sampler = sampler
        self.samples = []

    def __call__(self, size: int):
        samples = self.sampler(size)
        self.samples.append(samples.copy())
        return samples

    def get_samples(self):
        return np.concatenate(self.samples, axis=0)
```

In [26]:
```
memorized_truncnorm_sampler1 = MemorizedSampler(sample_mixture_truncnorm)
truncnorm_coreset1, truncnorm_mass1 = r_coreset(
    memorized_truncnorm_sampler1,
    k=3,
    eps=0.99,
    rho=0.3,
    delta=0.01,
    Gamma=0.5,
    beta=1.0,
    Delta=np.sqrt(2),
    random_state=RandomState(2),
)

memorized_truncnorm_sampler2 = MemorizedSampler(sample_mixture_truncnorm)
truncnorm_coreset2, truncnorm_mass2 = r_coreset(
    memorized_truncnorm_sampler2,
    k=3,
    eps=0.99,
    rho=0.3,
    delta=0.01,
    Gamma=0.5,
    beta=1.0,
    Delta=np.sqrt(2),
    random_state=RandomState(2),
)
```

```
3 2.040608101214162 0.026952749999999999
1
2
r_heavy_hitters...
80
114339
[0.17847803 0.03245612 0.0329284  0.00581604 0.17475227 0.0331383
 0.03358434 0.00641951 0.17852176 0.03444144 0.03376801 0.00627083
 0.17741978 0.03226371 0.03354061 0.00620086] 0.15941634879827712 114419
3
r_heavy_hitters...
17
5350
[0.00168224 0.01495327 0.01495327 0.14242991 0.0011215  0.01495327
 0.01028037 0.15121495 0.00149533 0.0128972  0.01570093 0.15196262
 0.0011215  0.01308411 0.01663551 0.14093458 0.29457944] 0.5787188439727955
5367
0.18518518518518517 0.007407407407407407 230556 1
3 2.040608101214162 0.026952749999999999
1
2
r_heavy_hitters...
80
114339
[0.17720113 0.03354061 0.03407411 0.00625333 0.17516333 0.03349688
 0.03364556 0.00619211 0.17658017 0.03307708 0.03339193 0.00624459
 0.17776087 0.03326074 0.0333482  0.00676934] 0.15941634879827712 114419
3
r_heavy_hitters...
17
5350
[0.00093458 0.01196262 0.01364486 0.14859813 0.00093458 0.01364486
 0.01196262 0.15158879 0.00074766 0.01196262 0.01719626 0.16766355
 0.00093458 0.01364486 0.01439252 0.14598131 0.27420561] 0.5787188439727955
5367
0.18518518518518517 0.007407407407407407 230556 1
```

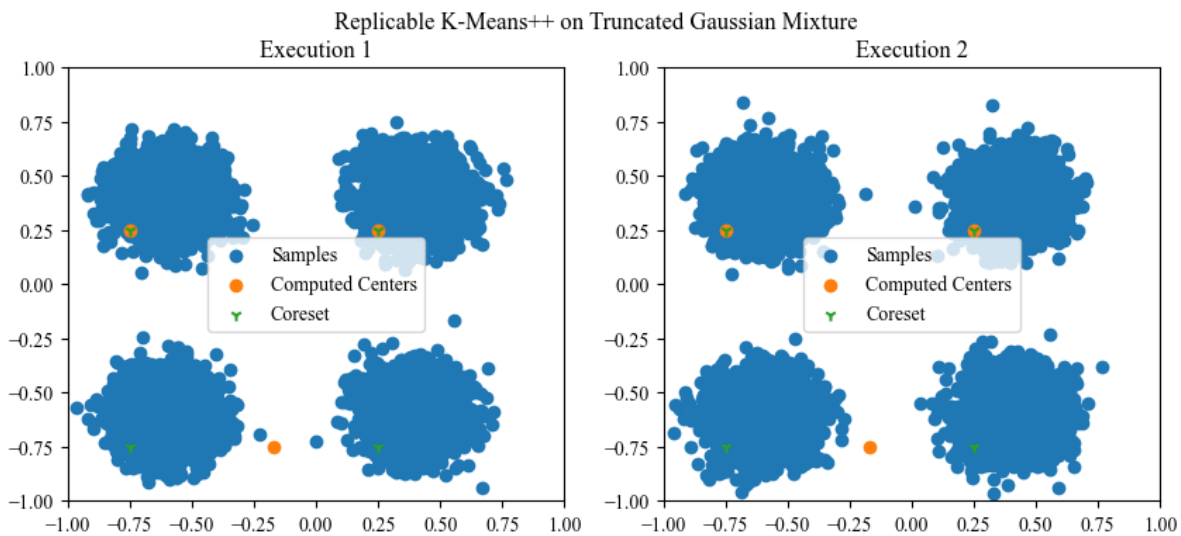
In [27]:

```
from sklearn.cluster import KMeans

r_truncnorm_kmeans1 = KMeans(n_clusters=3).fit(
    truncnorm_coreset1, sample_weight=truncnorm_mass1
)
r_truncnorm_kmeans2 = KMeans(n_clusters=3).fit(
    truncnorm_coreset2, sample_weight=truncnorm_mass2
)

fig, ax = plt.subplots(1, 2, figsize=(10, 4))

memorized_truncnorm_data1 = memorized_truncnorm_sampler1.get_samples()
print(f"used {len(memorized_truncnorm_data1)} samples")
memorized_truncnorm_data_indices1 = np.random.choice(
    memorized_truncnorm_data1.shape[0], 10000, replace=False
)

ax[0].scatter(
    memorized_truncnorm_data1[memorized_truncnorm_data_indices1, 0],
```

```
    memorized_truncnorm_data1[memorized_truncnorm_data_indices1, 1],
    label="Samples",
)
# ax[0].scatter([0.4, 0.4, -0.6, -0.6], [0.4, -0.6, 0.4, -0.6])
ax[0].scatter(
    r_truncnorm_kmeans1.cluster_centers_[:, 0],
    r_truncnorm_kmeans1.cluster_centers_[:, 1],
    label="Computed Centers",
)

ax[0].scatter(
    truncnorm_coreset1[:, 0],
    truncnorm_coreset1[:, 1],
    marker="1",
    # linewidths=10,
    label="Coreset",
)

ax[0].set_xlim(-1, 1)
ax[0].set_ylim(-1, 1)
ax[0].legend()
ax[0].set_title("Execution 1")

memorized_truncnorm_data2 = memorized_truncnorm_sampler2.get_samples()
print(f"used {len(memorized_truncnorm_data2)} samples")
memorized_truncnorm_data_indices2 = np.random.choice(
    memorized_truncnorm_data2.shape[0], 10000, replace=False
)

ax[1].scatter(
    memorized_truncnorm_data2[memorized_truncnorm_data_indices2, 0],
    memorized_truncnorm_data2[memorized_truncnorm_data_indices2, 1],
    label="Samples",
)
# ax[1].scatter([0.4, 0.4, -0.6, -0.6], [0.4, -0.6, 0.4, -0.6])
ax[1].scatter(
    r_truncnorm_kmeans2.cluster_centers_[:, 0],
    r_truncnorm_kmeans2.cluster_centers_[:, 1],
    label="Computed Centers",
)

plt.scatter(
    truncnorm_coreset2[:, 0],
    truncnorm_coreset2[:, 1],
    marker="1",
    # linewidths=10,
    label="Coreset",
)

ax[1].set_xlim(-1, 1)
ax[1].set_ylim(-1, 1)
ax[1].legend()
ax[1].set_title("Execution 2")

fig.suptitle("Replicable K-Means++ on Truncated Gaussian Mixture")
```

```
plt.savefig("r_truncnorm.pdf", format="pdf", bbox_inches="tight")

plt.show()
```

```
/Users/cz397/.mambaforge/envs/approx_matmul/lib/python3.10/site-packages/sk
learn/cluster/_kmeans.py:870: FutureWarning: The default value of `n_init`
will change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly
to suppress the warning
  warnings.warn(
/Users/cz397/.mambaforge/envs/approx_matmul/lib/python3.10/site-packages/sk
learn/cluster/_kmeans.py:870: FutureWarning: The default value of `n_init`
will change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly
to suppress the warning
  warnings.warn(
used 350342 samples
used 350342 samples
```



In [24]:
```
memorized_moons_sampler1 = MemorizedSampler(sample_moons)
moons_coreset1, moons_mass1 = r_coreset(
    memorized_moons_sampler1,
    k=3,
    eps=0.99,
    rho=0.4,
    delta=0.1,
    Gamma=0.5,
    beta=1.0,
    Delta=np.sqrt(2),
    random_state=RandomState(2),
)

memorized_moons_sampler2 = MemorizedSampler(sample_moons)
moons_coreset2, moons_mass2 = r_coreset(
    memorized_moons_sampler2,
    k=3,
    eps=0.99,
    rho=0.4,
    delta=0.1,
    Gamma=0.5,
    beta=1.0,
    Delta=np.sqrt(2),
```

```
    random_state=RandomState(2),
)
```

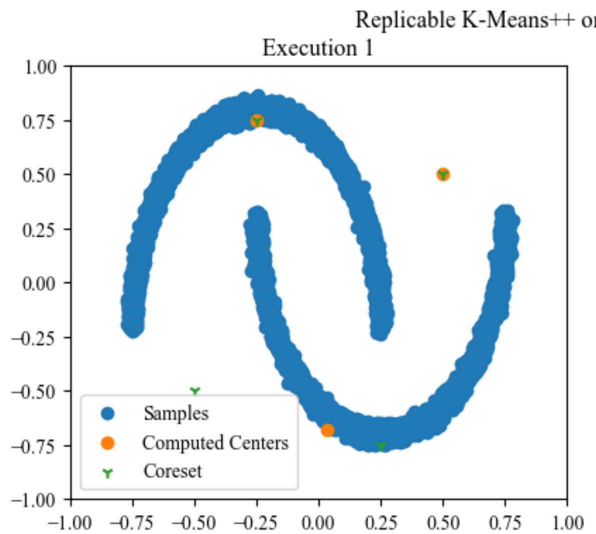
```
3 2.040608101214162 0.02695274999999999
1
2
r_heavy_hitters...
59
48935
[0.03177685 0.01910698 0.09902932 0.09169306 0.04273015 0.04871769
 0.16720139 0.16673138 0.0319812  0.01871871 0.09892715 0.0914274
 0.04317973 0.04877899] 0.15941634879827712 48994
3
r_heavy_hitters...
12
2175
[0.03172414 0.05011494 0.03448276 0.04781609 0.08275862 0.0845977
 0.66850575] 0.5787188439727955 2187
0.18518518518518517 0.007407407407407407 230556 1
3 2.040608101214162 0.02695274999999999
1
2
r_heavy_hitters...
59
48935
[0.03194033 0.01886176 0.09960151 0.0914274  0.04350669 0.04845203
 0.16644528 0.16652703 0.03222642 0.01873914 0.09941759 0.09146827
 0.04301625 0.04837029] 0.15941634879827712 48994
3
r_heavy_hitters...
12
2175
[3.12643678e-02 4.91954023e-02 3.21839080e-02 5.19540230e-02
 8.45977011e-02 4.59770115e-04 8.36781609e-02 6.66666667e-01] 0.57871884397
27955 2187
0.18518518518518517 0.007407407407407407 230556 1
```

In [25]:

```
r_moons_kmeans1 = KMeans(n_clusters=3).fit(moons_coreset1, sample_weight=moo
r_moons_kmeans2 = KMeans(n_clusters=3).fit(moons_coreset2, sample_weight=moo

fig, ax = plt.subplots(1, 2, figsize=(10, 4))

memorized_moons_data1 = memorized_moons_sampler1.get_samples()
print(f"used {len(memorized_moons_data1)} samples")
memorized_moons_data_indices1 = np.random.choice(
    memorized_moons_data1.shape[0], 10000, replace=False
)

ax[0].scatter(
    memorized_moons_data1[memorized_moons_data_indices1, 0],
    memorized_moons_data1[memorized_moons_data_indices1, 1],
    label="Samples",
)
ax[0].scatter(
    r_moons_kmeans1.cluster_centers_[:, 0],
    r_moons_kmeans1.cluster_centers_[:, 1],
```

```
    label="Computed Centers",
)

ax[0].scatter(
    moons_coreset1[:, 0],
    moons_coreset1[:, 1],
    marker="1",
    # linewidths=10,
    label="Coreset",
)

ax[0].set_xlim(-1, 1)
ax[0].set_ylim(-1, 1)
ax[0].legend()
ax[0].set_title("Execution 1")

memorized_moons_data2 = memorized_moons_sampler2.get_samples()
print(f"used {len(memorized_moons_data2)} samples")
memorized_moons_data_indices2 = np.random.choice(
    memorized_moons_data2.shape[0], 10000, replace=False
)

ax[1].scatter(
    memorized_moons_data2[memorized_moons_data_indices2, 0],
    memorized_moons_data2[memorized_moons_data_indices2, 1],
    label="Samples",
)
ax[1].scatter(
    r_moons_kmeans2.cluster_centers_[:, 0],
    r_moons_kmeans2.cluster_centers_[:, 1],
    label="Computed Centers",
)

plt.scatter(
    moons_coreset2[:, 0],
    moons_coreset2[:, 1],
    marker="1",
    # linewidths=10,
    label="Coreset",
)

ax[1].set_xlim(-1, 1)
ax[1].set_ylim(-1, 1)
ax[1].legend()
ax[1].set_title("Execution 2")

fig.suptitle("Replicable K-Means++ on Two Moons Distribution")

plt.savefig("r_moons.pdf", format="pdf", bbox_inches="tight")

plt.show()
```

```
/Users/cz397/.mambaforge/envs/approx_matmul/lib/python3.10/site-packages/sk
learn/cluster/_kmeans.py:870: FutureWarning: The default value of `n_init`
will change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly
to suppress the warning
  warnings.warn(
/Users/cz397/.mambaforge/envs/approx_matmul/lib/python3.10/site-packages/sk
learn/cluster/_kmeans.py:870: FutureWarning: The default value of `n_init`
will change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly
to suppress the warning
  warnings.warn(
used 281737 samples
used 281737 samples
```




```
In [ ]:
```