In [ ]:
```
# !pip install git+https://github.com/openai/CLIP.git
# !pip install torch torchvision
```

In [2]:
```
import ssl
import os

# Désactive la vérification SSL pour le téléchargement
ssl._create_default_https_context = ssl._create_unverified_context
print("Certificats SSL contournés pour le téléchargement.")
```

```
Certificats SSL contournés pour le téléchargement.
```

Initialisation

In [3]:
```
import torch
import clip
from torchvision.datasets import Food101

# Configuration du device pour MacBook (M1, M2, M3)
if torch.backends.mps.is_available():
    device = torch.device("mps")
else:
    device = torch.device("cpu")

print(f"Device utilisé : {device}")

# C'est ici qu'on définit 'preprocess'
model, preprocess = clip.load("ViT-B/32", device=device)
```

```
Device utilisé : mps
100%|███████████████████████████████████████| 338M/338M [00:11<00:00, 30.
6MiB/s]
```

Téléchargement et Filtrage de Food-101

In [ ]:
```
from torchvision.datasets import Food101
import torch

# # 1. Téléchargement (seulement si nécessaire)
# full_train_ds = Food101(root="./data", split="train", download=True, tr
# full_test_ds = Food101(root="./data", split="test", download=True, tran
```

In [5]:
```python
import os
import shutil
from torchvision.datasets import Food101, ImageFolder
from tqdm import tqdm

# Configuration des chemins
source_path = "./data"
filtered_path = "./data_filtered"
target_classes = ['pizza', 'hamburger', 'sushi', 'tacos', 'apple_pie',
                  'ice_cream', 'omelette', 'french_fries', 'guacamole', '

def create_filtered_dataset(split, max_samples=200):
    split_path = os.path.join(filtered_path, split)

    if os.path.exists(split_path):
        print(f"✅ Dossier {split} déjà existant dans {filtered_path}")
        return

    print(f"📂 Création du dataset filtré ({split})...")
    # On charge le dataset original sans transformation pour copier les f
    full_ds = Food101(root=source_path, split=split, download=True)

    class_counts = {c: 0 for c in target_classes}
    target_indices = {full_ds.class_to_idx[c]: c for c in target_classes}

    for i in tqdm(range(len(full_ds))):
        _, label = full_ds[i]

        if label in target_indices:
            class_name = target_indices[label]
            if class_counts[class_name] < max_samples:
                # Créer le dossier de la classe
                os.makedirs(os.path.join(split_path, class_name), exist_o

                # Récupérer le chemin de l'image originale
                img_path = full_ds._image_files[i]
                dest_path = os.path.join(split_path, class_name, os.path.

                # Copier le fichier
                shutil.copy(img_path, dest_path)
                class_counts[class_name] += 1

        if all(count >= max_samples for count in class_counts.values()):
            break

# Exécution pour Train et Test
create_filtered_dataset("train", max_samples=200)
create_filtered_dataset("test", max_samples=50)
```

```
📂 Création du dataset filtré (train)...
 95%|█████████▌| 72199/75750 [01:29<00:04, 810.92it/s]
📂 Création du dataset filtré (test)...
 95%|█████████▌| 24049/25250 [00:31<00:01, 766.64it/s]
```

Chargement du Dataset filtré

In [6]:
```
from torchvision import transforms

# On utilise le 'preprocess' de CLIP comme transformation
train_ds = ImageFolder(root=os.path.join(filtered_path, "train"), transfo
test_ds = ImageFolder(root=os.path.join(filtered_path, "test"), transform

print(f"Classes chargées : {train_ds.classes}")
print(f"Total images : {len(train_ds)} (Train) / {len(test_ds)} (Test)")
```

```
Classes chargées : ['apple_pie', 'french_fries', 'guacamole', 'hamburge
r', 'ice_cream', 'omelette', 'paella', 'pizza', 'sushi', 'tacos']
Total images : 2000 (Train) / 500 (Test)
```

Etape 1: Zero-Shot

1. Définir les prompts : Créer une liste de phrases comme "une photo de pizza".
2. Encoder le texte : Transformer ces phrases en vecteurs (embeddings) avec model.encode_text.
3. Comparer : Pour chaque image du set de test, calculer sa similarité avec tes 10 phrases. La phrase la plus proche donne la prédiction.
4. Calculer l'Accuracy : Comparer ces prédictions aux vrais labels.

1. Préparation des prompts

In [7]:
```
# On récupère l'ordre exact des classes chargées par ImageFolder
class_names = test_ds.classes
print(f"Ordre des classes : {class_names}")

# Création des prompts avec un template
templates = [f"a photo of {c}, a type of food" for c in class_names]
print(f"Exemple de prompt : {templates[0]}")
```

```
Ordre des classes : ['apple_pie', 'french_fries', 'guacamole', 'hamburge
r', 'ice_cream', 'omelette', 'paella', 'pizza', 'sushi', 'tacos']
Exemple de prompt : a photo of apple_pie, a type of food
```

2. Encodage du texte

In [8]:
```
# Tokenisation et Encodage
text_tokens = clip.tokenize(templates).to(device)

with torch.no_grad():
    text_features = model.encode_text(text_tokens)
    # Normalisation pour le calcul de la similarité cosinus
    text_features /= text_features.norm(dim=-1, keepdim=True)

print(f"Dimensions des features texte : {text_features.shape}") # [10, 51
```

```
Dimensions des features texte : torch.Size([10, 512])
```

3. Comparaison & Calcul de l'Accuracy

In [9]:
```
from torch.utils.data import DataLoader

test_loader = DataLoader(test_ds, batch_size=32, shuffle=False)
correct = 0
total = 0

model.eval()
with torch.no_grad():
    for images, labels in tqdm(test_loader, desc="Évaluation Zero-shot"):
        images = images.to(device)
        labels = labels.to(device)

        # 1. Encoder l'image
        image_features = model.encode_image(images)
        image_features /= image_features.norm(dim=-1, keepdim=True)

        # 2. Calculer la similarité (Produit scalaire car les vecteurs so
        # On multiplie par 100 (le logit_scale de CLIP) avant le softmax
        logits = (100.0 * image_features @ text_features.T).softmax(dim=-

        # 3. Prendre la classe avec le score le plus haut
        preds = logits.argmax(dim=-1)

        correct += (preds == labels).sum().item()
        total += labels.size(0)

zero_shot_accuracy = (correct / total) * 100
print(f"\n--- RÉSULTAT ---")
print(f"Accuracy Zero-shot sur Food-10 : {zero_shot_accuracy:.2f}%")
```

```
Évaluation Zero-shot: 100%|██████████| 16/16 [00:05<00:00,  2.69it/s]
--- RÉSULTAT ---
Accuracy Zero-shot sur Food-10 : 98.80%
```

Etape 2: Extraction et Stockage des Features

1. Passage unique : Tu passes toutes tes images de train_ds et test_ds dans model.encode_image.
2. Conversion NumPy : Tu stockes les résultats dans des tableaux NumPy (train_features et test_features). Pourquoi ? Parce qu'entraîner un classifieur sur des vecteurs déjà extraits prend 1 seconde, alors que le faire sur des images prendrait des minutes.

1. Fonction d'extraction des caractéristiques (Features)

In [10]:
```python
import numpy as np

def extract_and_save_features(dataset, filename_prefix):
    features_list = []
    labels_list = []

    # Utilisation d'un batch_size de 32 pour ne pas saturer la RAM du Mac
    loader = torch.utils.data.DataLoader(dataset, batch_size=32, shuffle=

    model.eval()
    with torch.no_grad():
        for images, labels in tqdm(loader, desc=f"Extraction {filename_pr
            # Envoi sur MPS (ou CPU)
            images = images.to(device)

            # 1. Passage dans l'encodeur d'images de CLIP
            features = model.encode_image(images)

            # 2. Normalisation L2 (essentielle pour la cohérence avec CLI
            features /= features.norm(dim=-1, keepdim=True)

            # 3. Stockage en NumPy (on quitte le GPU/MPS ici)
            features_list.append(features.cpu().numpy())
            labels_list.append(labels.numpy())

    # On concatène tous les batches
    final_features = np.concatenate(features_list, axis=0)
    final_labels = np.concatenate(labels_list, axis=0)

    # Sauvegarde sur le disque
    np.save(f"{filename_prefix}_features.npy", final_features)
    np.save(f"{filename_prefix}_labels.npy", final_labels)

    return final_features, final_labels
```

2. Exécution et Stockage

In [11]:
```python
print("🚀 Début de l'extraction massive...")

# Extraction pour le Train (200 images par classe)
train_features, train_labels = extract_and_save_features(train_ds, "train

# Extraction pour le Test (50 images par classe)
test_features, test_labels = extract_and_save_features(test_ds, "test")

print("\n✅ Extraction terminée et fichiers sauvegardés !")
print(f"Dimensions Train : {train_features.shape}") # Devrait être [2000,
print(f"Dimensions Test : {test_features.shape}")   # Devrait être [500,
```

```
🚀 Début de l'extraction massive...
Extraction train: 100%|██████████| 63/63 [00:19<00:00,  3.18it/s]
Extraction test: 100%|██████████| 16/16 [00:04<00:00,  3.36it/s]
✅ Extraction terminée et fichiers sauvegardés !
Dimensions Train : (2000, 512)
Dimensions Test : (500, 512)
```

Etape 3: Scénarios Linear Probing (1-shot et 5-shots)

1. Sélection des données (le "N-shot") :
    - Pour le 1-shot : Tu parcours ton set d'entraînement et tu prends exactement 1 image au hasard pour chacune des 10 classes.
    - Pour le 5-shots : Tu prends 5 images par classe.
2. Entraînement : Tu utilises une LogisticRegression (de Scikit-Learn) ou une couche Linear (PyTorch). Tu l'entraînes à faire le lien entre les features extraites à l'étape 2 et les labels.
3. Évaluation : Tu testes ce classifieur sur tes test_features.

1. Sélection des données (La logique N-Shot)

In [12]:
```
import numpy as np

def get_n_shot_data(features, labels, n_shots):
    x_sampled = []
    y_sampled = []

    unique_labels = np.unique(labels)

    for label in unique_labels:
        # On récupère tous les indices pour cette classe spécifique
        indices = np.where(labels == label)[0]

        # On en choisit n_shots au hasard
        selected_indices = np.random.choice(indices, n_shots, replace=Fal

        x_sampled.append(features[selected_indices])
        y_sampled.append(labels[selected_indices])

    # On transforme les listes en gros tableaux NumPy
    return np.concatenate(x_sampled), np.concatenate(y_sampled)
```

2. Entraînement et Évaluation (Le Linear Probe)

In [13]:
```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score

results = {}

for n in [1, 5]:
    # 1. Sélection des données pour ce scénario
    x_train_n, y_train_n = get_n_shot_data(train_features, train_labels,

    # 2. Création du classifieur
    # 'lbfgs' est très rapide pour les petits datasets
    classifier = LogisticRegression(max_iter=1000, C=0.3)

    # 3. Entraînement (très rapide, < 1 seconde)
    classifier.fit(x_train_n, y_train_n)

    # 4. Prédiction sur l'intégralité du set de TEST
    y_pred = classifier.predict(test_features)

    # 5. Calcul de l'Accuracy
    acc = accuracy_score(test_labels, y_pred) * 100
    results[f"{n}-shot"] = acc

    print(f"✅ Scénario {n}-shot terminé. Accuracy : {acc:.2f}%")
```

```
✅ Scénario 1-shot terminé. Accuracy : 83.20%
✅ Scénario 5-shot terminé. Accuracy : 95.80%
```

Etape 4: Evaluation

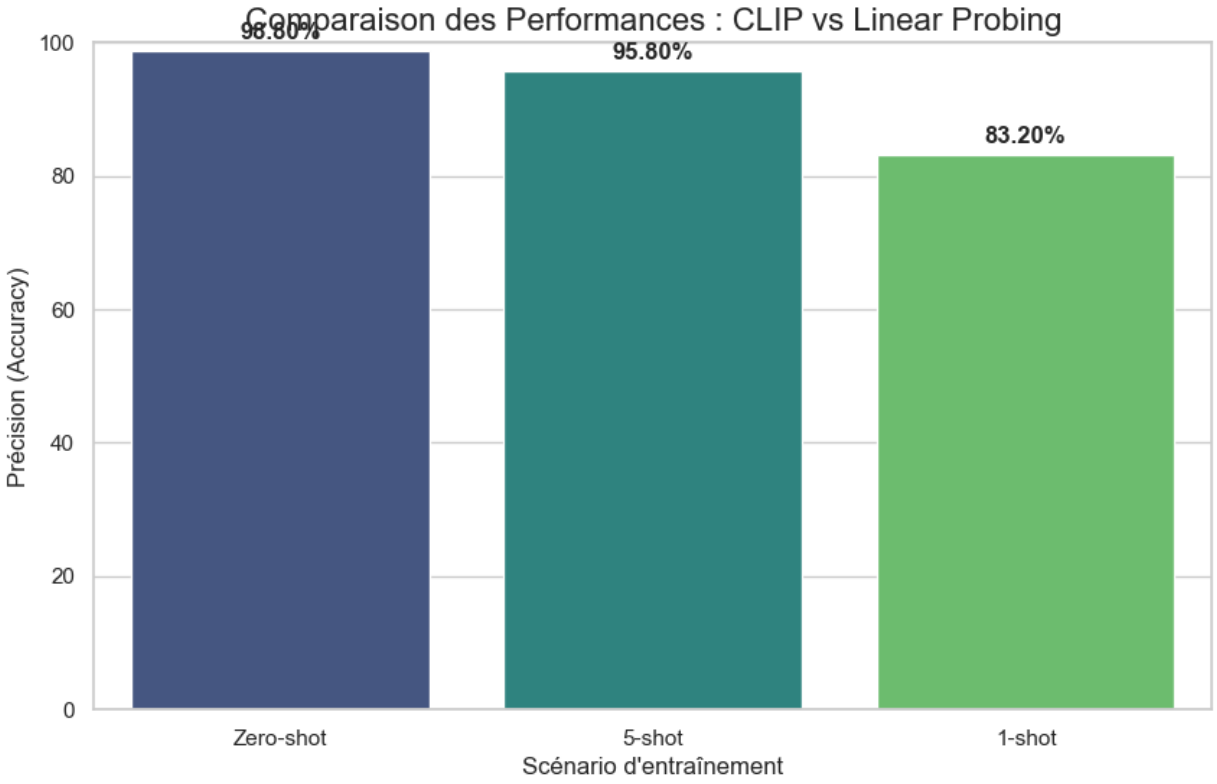
In [14]:
```
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd
from sklearn.metrics import confusion_matrix
import numpy as np

# Configuration du style des graphiques
sns.set_theme(style="whitegrid")

# --- Récupération des données ---
# On s'assure d'avoir les scores (remplace par tes vraies variables si el
# Exemple: scores_finaux = {'Zero-shot': zero_shot_accuracy, '1-shot': re
# Si tu as suivi le fil, 'results' contient déjà les 1-shot et 5-shot.
# On ajoute le zero-shot s'il n'y est pas :
if 'Zero-shot' not in results:
    results['Zero-shot'] = zero_shot_accuracy

# On ré-entraîne le meilleur modèle (5-shot) pour les analyses suivantes
x_train_5s, y_train_5s = get_n_shot_data(train_features, train_labels, n_
best_classifier = LogisticRegression(max_iter=1000, C=0.3)
best_classifier.fit(x_train_5s, y_train_5s)
y_pred_best = best_classifier.predict(test_features)

print("Données prêtes pour la visualisation.")
```

```
Données prêtes pour la visualisation.
```

Bar Chart de Comparaison

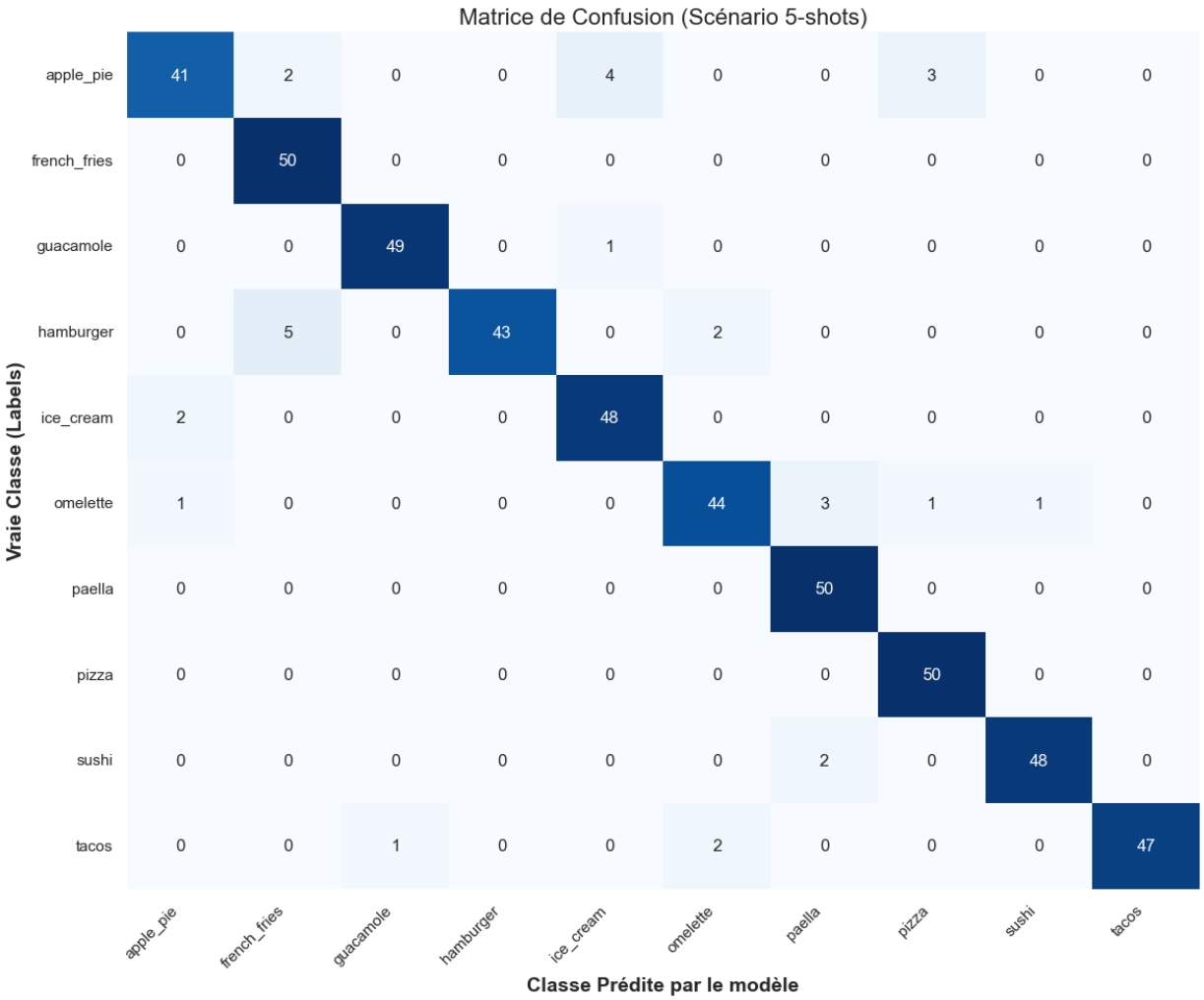
In [15]:
```
# Création d'un DataFrame pour faciliter l'utilisation de Seaborn
df_results = pd.DataFrame(list(results.items()), columns=['Méthode', 'Acc
# On trie pour avoir un ordre logique (Zero -> 1 -> 5)
df_results = df_results.sort_values(by='Méthode', ascending=False)

plt.figure(figsize=(10, 6))

# Création du barplot
ax = sns.barplot(x="Méthode", y="Accuracy (%)", data=df_results, palette=

# Ajout des valeurs exactes au-dessus des barres
for p in ax.patches:
    ax.annotate(f'{p.get_height():.2f}%',
                (p.get_x() + p.get_width() / 2., p.get_height()),
                ha = 'center', va = 'center',
                xytext = (0, 9),
                textcoords = 'offset points',
                fontweight='bold')

plt.title('Comparaison des Performances : CLIP vs Linear Probing', fontsi
plt.ylim(0, 100) # On fixe l'axe Y de 0 à 100%
plt.ylabel('Précision (Accuracy)', fontsize=12)
plt.xlabel('Scénario d\'entraînement', fontsize=12)
plt.show()
```

```
/var/folders/7x/5z5j8nzn73j56x9hr4346d_80000gn/T/ipykernel_52622/21363103
32.py:9: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be remov
ed in v0.14.0. Assign the `x` variable to `hue` and set `legend=False` fo
r the same effect.

  ax = sns.barplot(x="Méthode", y="Accuracy (%)", data=df_results, palett
e="viridis")
```



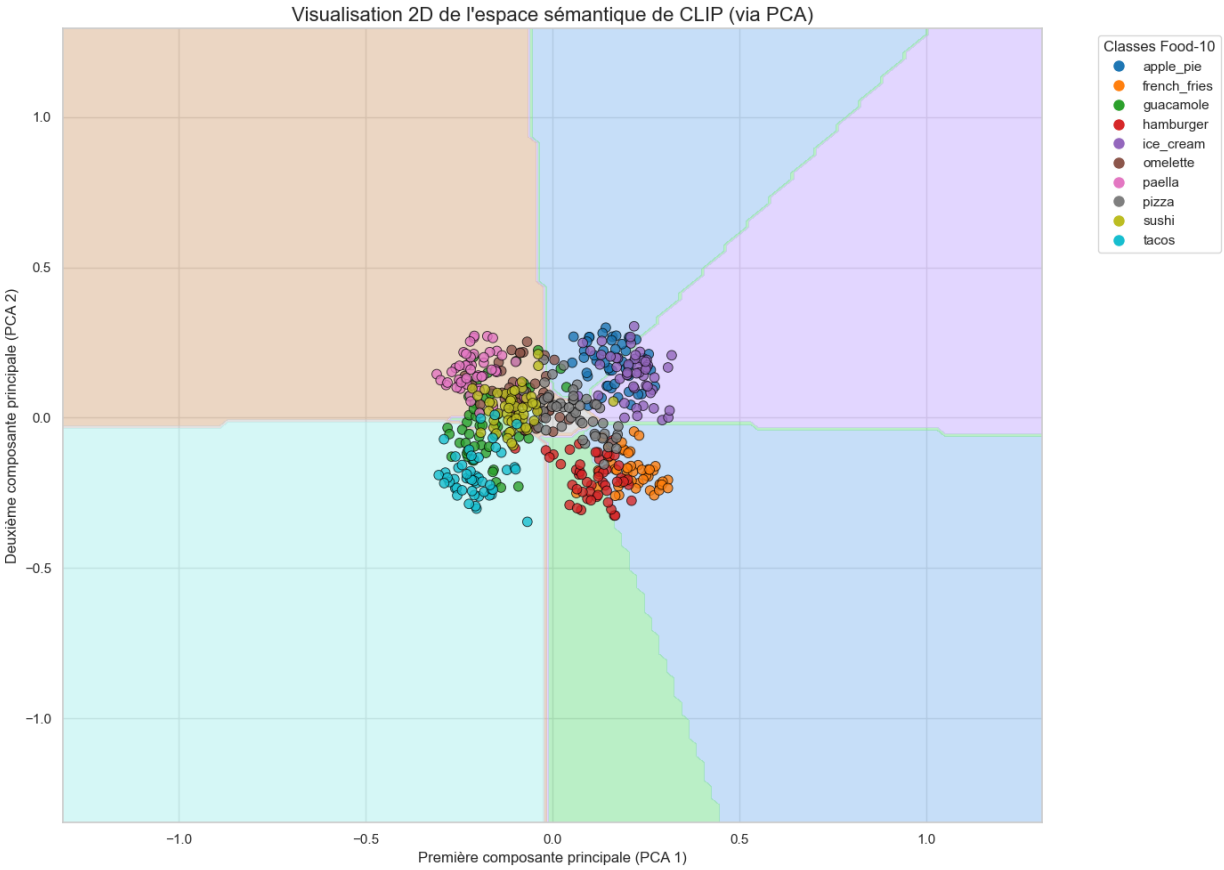
Matrice de Confusion (du modèle 5-shots)

In [16]:
```
# Calcul de la matrice de confusion sur le modèle 5-shots
cm = confusion_matrix(test_labels, y_pred_best)

plt.figure(figsize=(12, 10))

# Création de la heatmap
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
            xticklabels=class_names,
            yticklabels=class_names,
            cbar=False) # 'd' pour entier (integer), cbar=False enlève la

plt.title('Matrice de Confusion (Scénario 5-shots)', fontsize=16)
plt.ylabel('Vraie Classe (Labels)', fontsize=14, fontweight='bold')
plt.xlabel('Classe Prédite par le modèle', fontsize=14, fontweight='bold'

# Rotation des labels pour qu'ils soient lisibles
plt.xticks(rotation=45, ha='right')
plt.yticks(rotation=0)

plt.tight_layout() # Ajuste les marges pour que rien ne soit coupé
plt.show()
```



Scatter Plot et Frontières de Décision

In [17]:
```
from sklearn.decomposition import PCA
from matplotlib.colors import ListedColormap

# 1. Réduction de dimension (512D -> 2D)
pca = PCA(n_components=2)
X_test_2d = pca.fit_transform(test_features)

# 2. On entraîne un petit logistic regression SUR LES DONNÉES 2D juste po
clf_2d = LogisticRegression(C=1.0).fit(X_test_2d, test_labels)

# 3. Création d'une grille pour dessiner les frontières
h = .02  # finess de la grille
x_min, x_max = X_test_2d[:, 0].min() - 1, X_test_2d[:, 0].max() + 1
y_min, y_max = X_test_2d[:, 1].min() - 1, X_test_2d[:, 1].max() + 1
xx, yy = np.meshgrid(np.arange(x_min, x_max, h), np.arange(y_min, y_max,

# 4. Prédiction sur toute la grille pour colorer le fond
Z = clf_2d.predict(np.c_[xx.ravel(), yy.ravel()])
Z = Z.reshape(xx.shape)

# --- Plotting ---
plt.figure(figsize=(14, 10))

# Couleurs douces pour le fond (frontières)
cmap_light = ListedColormap(sns.color_palette("pastel", n_colors=10))
# Couleurs vives pour les points
cmap_bold = sns.color_palette("tab10", n_colors=10)

# Dessin des frontières (le fond coloré)
plt.contourf(xx, yy, Z, cmap=cmap_light, alpha=0.6)

# Dessin des points de test (scatter plot)
sns.scatterplot(x=X_test_2d[:, 0], y=X_test_2d[:, 1], hue=test_labels,
                palette=cmap_bold, s=60, edgecolor="black", alpha=0.8)

# Légende avec les vrais noms de classes
# On crée une légende personnalisée car le 'hue' utilise les IDs numériqu
handles = [plt.Line2D([0], [0], marker='o', color='w', markerfacecolor=c,
plt.legend(handles, class_names, title="Classes Food-10", bbox_to_anchor=

plt.title('Visualisation 2D de l\'espace sémantique de CLIP (via PCA)', f
plt.xlabel('Première composante principale (PCA 1)')
plt.ylabel('Deuxième composante principale (PCA 2)')
plt.tight_layout()
plt.show()
```


Visualisation 2D de l'espace sémantique de CLIP (via PCA)
Classes Food-10
apple_pie
french_fries
guacamole
hamburger
ice_cream
omelette
paella
pizza
sushi
tacos
Deuxième composante principale (PCA 2)
Première composante principale (PCA 1)
1.0
0.5
0.0
−0.5
−1.0
−1.0
−0.5
0.0
0.5
1.0