

Sujet – Design4Green 2025

« Résumer mieux avec moins » : API Flask de résumé éco-conçue

Les modèles génératifs coûtent en CPU/RAM/énergie. Dans une optique de sobriété, on vous demande de bâtir une API de résumé de textes qui soit légère, mesurable et optimisée.

Créer une application Flask exposant POST /summarize qui renvoie un résumé de 10–15 mots d'un texte donné, en réduisant l'empreinte (énergie, temps, mémoire) sans dégrader la qualité.

Contraintes techniques

- Backend : Flask (Python).
- Modèle par défaut : EleutherAI/pythia-70m-deduped (CPU ok).
- Deux modes avec le même endpoint :
 - optimized=false → baseline (fp32)
 - optimized=true → optimisé (ex. INT8 dynamique, low-rank, petites coupes, torch.compile... au choix)
- Mesure d'énergie via CodeCarbon (ou export JSON équivalent).
- Application web intégrée permettant de tester l'API avec interface graphique affichant toutes les métriques.

Livrables (H48)

1. Repo Git avec :
 - app.py (API Flask), requirements.txt, README.md (comment lancer les scripts)
 - Tous les scripts nécessaires pour lancer l'application web
 - judge.py (fourni par l'organisation) s'exécute sans changement
 - Script principal de lancement pour démarrer l'application web
2. Rapport court (minimum 2 pages) :
 - techniques d'optimisation appliquées, hyperparamètres
 - mesures : énergie (Wh), latence (ms), mémoire (si disponible)
 - arbitrages, limites, pistes d'amélioration
3. Application web fonctionnelle (accessible via script de lancement) :
 - Interface permettant de générer un résumé de 10-15 mots par rapport à une entrée
 - Design accessible, claire et cohérente avec les principes de sobriété numérique
 - Affichage : résumé généré, énergie consommée (Wh), latence (ms)

Évaluation

Le jury lance :

python judge.py

Le script appelle votre /summarize en baseline puis en mode optimisé et n'affiche que deux chiffres :

1. Score final (/100)
2. Économie d'énergie (%) vs baseline

Éligibilité : Qualité ≥ 0.95 (au moins 95 % des résumés font 10–15 mots).

Ce que vous pouvez optimiser (au choix) : Quantization CPU (INT8 dynamique), Low-rank sur quelques couches denses/attention, Torch.compile (si bénéfice mesuré), Pruning léger, Paramétrage (batch/seq_len raisonnables pour CPU).

Détails d'exécution et d'évaluation

- **Port & lancement :** l'API seule écoute sur `http://127.0.0.1:5000` (pour `judge.py`). Dans l'app web, l'API est intégrée.
- **Format d'appel (jury) :** POST /summarize avec JSON `{"text":"...", "optimized": true|false}`. Le jury n'enverra que ces deux champs.
- **Longueur & langue :** résumé en français, 10–15 mots (pas « Résumé : », pas d'intro). Entrée ≤ 4000 caractères.
- **Mesure énergie & latence :** mesurer par requête, du juste avant l'inférence (génération) au juste après. Application web doit afficher ces métriques en temps réel.
- **Baseline = FP32 :** quand `optimized=false`, utiliser strictement le modèle FP32 (aucune optimisation activée).
- **Reproductibilité :** fixer `PYTHONHASHSEED=0` et un `seed=42` (ex. torch, numpy) si vous faites du sampling.
- **Fonctionnement de l'application web :** L'utilisateur saisit un texte (≤ 4000 caractères), choisit le mode (baseline/optimisé), et obtient immédiatement le résumé avec toutes les métriques de performance affichées dans l'interface.

Note importante : L'API doit pouvoir fonctionner de manière autonome sur `http://127.0.0.1:5000` (pour l'évaluation `judge.py`) mais aussi être intégrée dans l'application web.