

Planning de travail (5 jours)

Accélération de l'inférence sur CIFAR-10 (latence GPU)

1 Objectif du sprint (5 jours)

But : obtenir un modèle CIFAR-10 atteignant $\geq 85\%$ d'accuracy sur test et minimisant la latence GPU en batch = 1, avec une démarche incrémentale et des mesures reproductibles.

2 Principes de mesure (à figer dès J1)

- **Métrique principale** : latence GPU (batch=1).
- **Protocole** : *warm-up* (50–200 itérations) + mesure (500–2000 itérations).
- **Synchronisation** : `torch.cuda.synchronize()` ou événements CUDA pour des temps corrects.
- **Rapports** : moyenne + p95 (95e percentile). Optionnel : p50, écart-type.
- **Conditions fixes** : même GPU, même prétraitement, même précision numérique (FP32/FP16), `model.eval()` et `torch.no_grad()`.

3 Itérations prévues (vue d'ensemble)

Chaque itération produit une **mesure** (accuracy + latence) et une **décision** (on garde / on abandonne / on affine).

Itération	Contenu	Sortie attendue
I0	Mise en place (data + code + benchmark)	Benchmark de latence validé
I1	Baseline <i>vitesse</i> : MobileNetV3/ShuffleNet entraîné from-scratch	1er modèle rapide + mesures
I2	Baseline <i>précision</i> : ResNet-18 CIFAR (teacher) from-scratch	Modèle robuste ($\geq 85\%$) + mesures
I3	Optimisations GPU (FP16, <code>compile</code> , <code>channels_last</code>)	Gains mesurés sur latence
I4	Distillation (si nécessaire) teacher→student	Student $\geq 85\%$ + latence meilleure
I5	Consolidation : tableau final + reproductibilité + rapport	Livrables finalisés

4 Planning détaillé sur 5 jours

Jour 1 (J1) — Démarrage + Itération I0

- Objectif** : disposer d'un environnement propre et d'un benchmark de latence GPU fiable.
- Installer / vérifier versions : PyTorch, torchvision, CUDA, drivers, `torch.compile` (si dispo).
 - Charger CIFAR-10 et définir transforms (normalisation, augmentations *train*).

- Implémenter le benchmark **latence** (batch=1) :
 - entrée déjà sur GPU (pas de transfert dans la boucle),
 - *warm-up* + mesure,
 - moyenne + p95,
 - sauvegarde dans un CSV.
- Vérifier la stabilité : répéter 3 runs (variance raisonnable).
- Critères d'acceptation J1 :**
- Un script de benchmark qui produit : moyenne, p95, et un fichier de résultats.
- Un protocole documenté (dans un README ou cellule notebook).

Jour 2 (J2) — Baseline vitesse + Itération I1

- Objectif :** entraîner un modèle léger from-scratch et mesurer sa latence.
- Adapter le notebook de départ :
 - **weights=None** (pas d'ImageNet),
 - tête de sortie en **10 classes**,
 - gestion propre du **device**.
 - Entrainer **MobileNetV3-Small ou ShuffleNetV2** :
 - objectif : monter rapidement vers $\geq 80\%$ puis viser $\geq 85\%$,
 - sauvegarder checkpoints et logs (accuracy train/test).
 - Mesurer latence (FP32) avec le benchmark J1.
 - Sorties J2 :**
 - Résultats (accuracy, latence moyenne, p95) pour **au moins un modèle léger**.
 - Première ligne du tableau comparatif.

Jour 3 (J3) — Baseline précision (teacher) + Itération I2

- Objectif :** obtenir une référence robuste $\geq 85\%$ et disposer d'un teacher.
- Implémenter/entraîner **ResNet-18 adaptée CIFAR-10** from-scratch.
 - Ajuster l'entraînement si nécessaire : LR schedule, augmentations, régularisation.
 - Évaluer sur test et mesurer latence (FP32) avec le benchmark.
 - Documenter les hyperparamètres (pour la reproductibilité).
 - Critères d'acceptation J3 :**
 - ResNet-18 atteint $\geq 85\%$ sur test.
 - Mesures latence ajoutées au tableau comparatif.

Jour 4 (J4) — Optimisations GPU + Itération I3 (et I4 si besoin)

- Objectif :** réduire la latence sans dégrader la précision en dessous de 85%.
- I3.1. FP16 en inférence (autocast) :**
- mesurer latence (moyenne + p95),
 - vérifier que l'accuracy ne chute pas significativement.
- I3.2. `torch.compile` (si stable) :**
- comparer latence avant/après,
 - noter le temps de compilation (hors métrique de latence).
- I3.3. `channels_last` :**

- tester sur modèle léger et ResNet,
- garder uniquement si amélioration mesurable.

Itération I4 (optionnelle, si le modèle léger n'atteint pas 85%) : Distillation

- Teacher : ResNet-18 (J3), Student : MobileNet/ShuffleNet (J2).
- Ajouter une perte de distillation (KL sur logits + CE).
- But : **Student** $\geq 85\%$ avec latence la plus faible.

Sorties J4 :

- Tableau mis à jour : FP32 vs FP16 vs compile vs channels_last.
- Décision : **candidat final** (meilleur compromis accuracy/latence).

Jour 5 (J5) — Consolidation + Itération I5

Objectif : finaliser livrables, reproductibilité et narration de la démarche.

- Nettoyer le code : scripts séparés (`train.py`, `eval.py`, `bench.py`) ou notebooks structurés.
- Figement des configs : seeds, versions, hyperparamètres, chemin des checkpoints.
- Refaire une **mesure finale** (3 runs) pour le candidat final et le baseline.
- Construire le **tableau final et la courte analyse** :
 - ce qui a été tenté,
 - ce qui a marché / pas marché,
 - justification du choix final.
- Préparer le rapport : méthode incrémentale (baseline \rightarrow optimisation \rightarrow mesures).

Critères d'acceptation J5 :

- Livrables prêts : code + rapport + résultats (CSV) + modèle final.
- Le modèle final respecte $\geq 85\%$ et présente la **meilleure latence** parmi les variantes conservées.

5 Gestion des risques (très court)

- **R1** : modèle léger $< 85\% \Rightarrow$ distillation (I4) + tuning léger (augmentations/LR schedule).
- **R2** : benchmark instable \Rightarrow augmenter itérations, isoler transferts, synchroniser correctement.
- **R3** : `torch.compile` instable \Rightarrow fallback FP16 + baseline FP32 + export TorchScript (optionnel).

6 Tableau de résultats (à compléter)

ID	Variante	Acc. (%)	Lat. moy. (ms)	Lat. p95 (ms)	Taille (MB)
B1	MobileNetV3/ShuffleNet (FP32)				
B2	ResNet-18 CIFAR (FP32)				
O1	B1 + FP16				
O2	B1 + FP16 + <code>compile</code>				
O3	B1 + channels_last				
D1	Distillation student (FP16)				