NH-LoRA: Future-Aware Structural Expansion of Low-Rank Adapters for Rehearsal-Free Class-Incremental Learning

Status dokumen. Ini adalah desain metode yang ditulis untuk proposal/paper internal. Arsitektur, persamaan, algoritma, diagram, dan protokol evaluasi sudah disusun agar siap dijadikan dasar implementasi dan penulisan metodologi, tetapi performanya tetap harus divalidasi secara empiris melalui eksperimen.

Ringkasan eksekutif

NH-LoRA dirancang sebagai metode PEFT continual learning yang *future-aware*, *structurally plastic*, dan *rehearsal-free*. Backbone ViT dibekukan, lalu pengetahuan lintas task diletakkan pada **Shared Core LoRA**, pengetahuan spesifik task diletakkan pada **Expandable Task Slot Bank**, sedangkan **Horizon Planner** memprediksi novelty, conflict, rank budget, dan consolidation signal untuk menentukan apakah suatu block perlu *reuse*, *expand*, *open new slot*, *freeze*, *merge*, atau *prune*.

Dokumen ini mencakup *problem setup*, delapan modul utama, persamaan inti, *objective function*, *pseudo-code* forward block, *pseudo-code* training loop, diagram arsitektur yang siap dilampirkan ke paper, serta rancangan protokol evaluasi di benchmark CIL berbasis PILOT.

1 Latar belakang dan posisi metode

Dalam class-incremental learning (CIL), model harus mempelajari kelas baru secara bertahap tanpa kehilangan pengetahuan kelas lama. Pada setting *rehearsal-free*, data task lama tidak boleh disimpan kembali, sehingga *trade-off* antara *plasticity* dan *stability* menjadi inti persoalan.

Pendekatan PTM-based continual learning saat ini banyak bergantung pada frozen backbone dan modul PEFT ringan seperti prompt tuning atau low-rank adaptation. NH-LoRA ditempatkan di ruang desain ini: backbone besar tetap beku agar representasi dasar stabil, sedangkan adaptasi task dilakukan melalui modul low-rank yang dikontrol secara dinamis.

Novelty utama NH-LoRA bukan sekadar menambah adapter baru untuk tiap task. Metode ini memperlakukan struktur adapter sebagai objek keputusan. Dengan kata lain, model tidak hanya belajar parameter, tetapi juga memutuskan bagaimana kapasitas harus dialokasikan, kapan rank perlu dinaikkan, kapan slot baru perlu dibuka, dan kapan update yang stabil perlu dikonsolidasikan ke memori bersama.

Prinsip desain NH-LoRA: frozen backbone, *shared-vs-specific memory separation*, *dynamic rank selection*, *slot-based structural expansion*, *instance-level sparse routing*, serta *post-task homeostasis* melalui *merge/freeze/prune*.

2 Problem setup

Diasumsikan terdapat urutan task $D_1, D_2, \ldots, D_T$. Setiap task memperkenalkan kelas baru dan model harus memprediksi seluruh kelas yang pernah dilihat sampai saat itu tanpa diberikan *task identity* saat *inference*.

Backbone utama adalah Vision Transformer (utama: ViT-B/16-IN21K). Backbone dibekukan penuh. Parameter yang dilatih hanya NH-LoRA modules dan incremental classifier head.

Untuk task t, model terlebih dahulu melakukan *warm-up sensing* beberapa batch awal untuk membentuk *task state* z_t. Task state ini dipakai Horizon Planner untuk mengambil keputusan struktural sebelum training penuh task t dimulai.

2.1 Bootstrap Session untuk Task Pertama

Task pertama ditangani dalam *bootstrap mode* khusus karena pada tahap ini NH-LoRA belum memiliki *task history*, *snapshot* model dari task sebelumnya, maupun himpunan *old classes*. Oleh karena itu, komponen yang bergantung pada histori, seperti komputasi *history-aware similarity*, *teacher-based knowledge distillation*, dan *feature-retention regularization*, belum valid untuk digunakan pada $t = 1$.

Selama fase *bootstrap*, NH-LoRA tetap menjalankan *warm-up sensing* untuk membentuk *task state* dari statistik task saat ini, tetapi komponen similarity digantikan dengan *null signal*. Alih-alih menggunakan kebijakan *horizon planning* penuh yang bersifat *history-aware*, model menginisialisasi *shared low-rank memory* berukuran kecil bersama satu *bootstrap slot* pada setiap *transformer block* terpilih. Proses routing juga disederhanakan dengan mengaktifkan satu-satunya slot yang tersedia secara deterministik.

Objektif optimisasi untuk task pertama menonaktifkan loss yang berkaitan dengan retensi pengetahuan:

$$\mathcal{L}_{\text{kd}}^{(1)} = 0, \qquad \mathcal{L}_{\text{feat}}^{(1)} = 0,$$

sehingga model dapat berfokus pada *current-task fitting* dan pembentukan substruktur yang kompak. Setelah training task pertama selesai, NH-LoRA menjalankan proses *light consolidation* dan menyimpan ringkasan task yang dihasilkan ke dalam *history bank*. Mulai dari task $t = 2$ dan seterusnya, NH-LoRA beralih ke mode *continual learning* standar dengan *full horizon planning*, *teacher-based retention*, dan adaptasi struktural yang *history-aware*.

3 Diagram arsitektur NH-LoRA

Gambar berikut dirancang agar cukup bersih untuk proposal atau lampiran paper. Diagram menampilkan alur *training-time structural control*, jalur *forward* utama di backbone ViT yang dibekukan, serta *post-task consolidation* yang mengembalikan pengetahuan stabil ke *shared memory*.

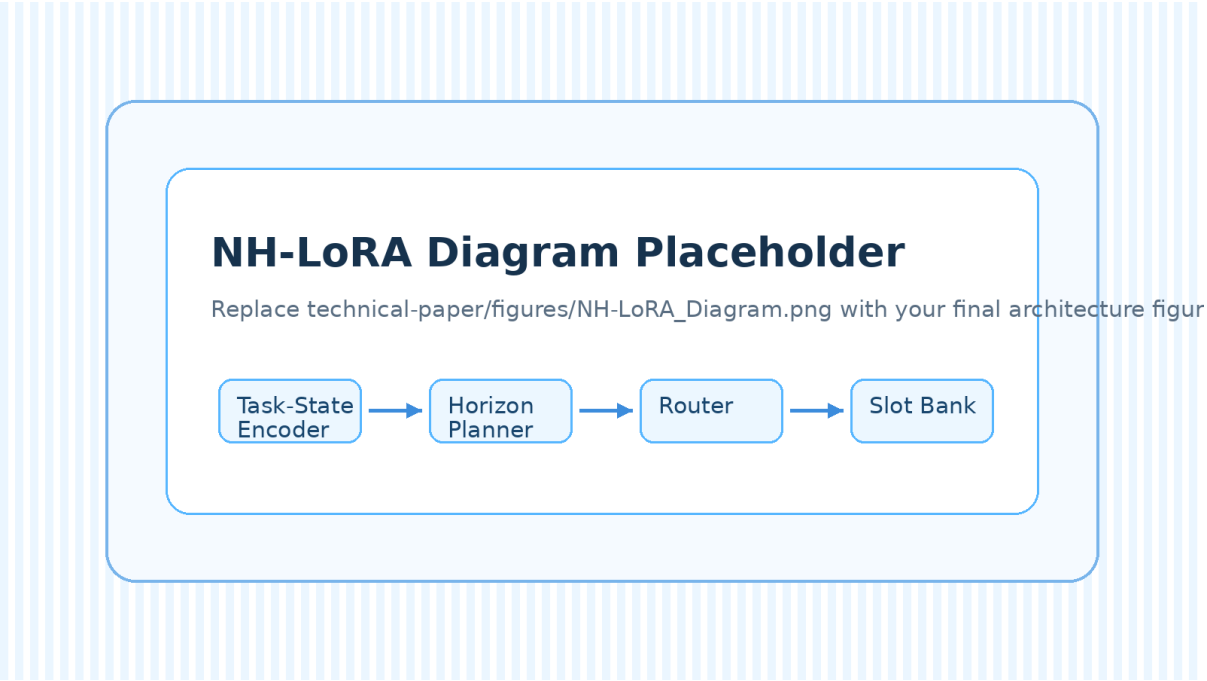

Figure 1: Overview of NH-LoRA. A frozen Vision Transformer is augmented with a shared low-rank memory and expandable slot-based adapters. A horizon planner predicts novelty, conflict, rank budget, and consolidation signals from current-task statistics and task history, while an instance router activates sparse task-specific pathways during inference. Stable slot updates are consolidated after each task to preserve reusable cross-task knowledge and limit uncontrolled parameter growth.

4 Ringkasan modul utama

Modul	**Peran utama**	**Jenis plasticity**	**Output / perilaku**
Frozen Vision Backbone	Menyediakan representasi dasar yang kuat dan stabil	Stability anchor	Parameter backbone tidak diupdate
Shared Core LoRA	Menyimpan pengetahuan lintas task yang reusable	Slow plasticity	Delta low-rank shared per block
Expandable Task Slot Bank	Menyimpan adaptasi spesifik task / subspace baru	Fast plasticity	Beberapa slot dengan rank mask dinamis
Task-State Encoder	Merangkum statistik task baru	Meta-sensing	z_t dari feature, gradient, similarity, entropy
Horizon Planner	Mengambil keputusan struktur adapter	Metaplastic control	Novelty, conflict, rank budget, consolidate
Instance Router	Memilih slot aktif per instance	Sparse pathway reuse	Koefisien routing $\alpha(x)$
Incremental Cosine Head	Melakukan klasifikasi seluruh kelas yang pernah dilihat	Task growth in label space	Expanded classifier weights

Consolidation & Homeostasis Unit	Merge / freeze / prune pasca-task	Structural plasticity + homeostasis	Kontrol pertumbuhan parameter

5 Penjelasan modul per modul

5.1 Frozen Vision Backbone

Backbone yang disarankan sebagai konfigurasi utama paper adalah ViT-B/16-IN21K. ViT-B/16-IN1K dapat dipakai sebagai kontrol, sedangkan CLIP ViT-B/16 dapat disimpan untuk studi tambahan.

Backbone dibekukan penuh agar representasi dasar tetap stabil dan biaya training rendah. Dengan demikian, semua adaptasi incremental benar-benar dapat diatribusikan ke NH-LoRA modules dan classifier head.

Pada implementasi awal, penyisipan NH-LoRA dapat difokuskan pada projection matrix Q, V, dan MLP di block ViT terpilih. Ini memberi titik masuk yang cukup representatif tanpa membuat biaya eksperimen terlalu berat.

5.2 Shared Core LoRA (SCM)

Pada setiap block l, NH-LoRA menyimpan satu *shared adapter* berbentuk

$$\Delta W_l^{\text{shared}} = B_l^{\text{shared}} A_l^{\text{shared}}.$$

Modul ini bertugas menyimpan pengetahuan yang cenderung reusable lintas task. Intuisi biologisnya adalah *slow plasticity*: update ada, tetapi konservatif dan diarahkan untuk menyimpan pola generik yang relatif stabil.

Rank *shared adapter* dibuat kecil, misalnya

$$r_{\text{shared}} \in \{2, 4, 8\},$$

dan *learning rate*-nya dapat dibuat lebih rendah daripada *slot adapter* agar *shared memory* tidak terlalu mudah ter-disturb oleh task baru.

5.3 Expandable Task Slot Bank (ESB)

Selain *shared memory*, setiap block l memiliki bank *slot adapter*

$$S_l = \left\{ \Delta W_{l,1}^{\text{slot}}, \ldots, \Delta W_{l,S_l}^{\text{slot}} \right\}.$$

Setiap slot ditulis sebagai

$$\Delta W_{l,s}^{\text{slot}} = B_{l,s} \, \text{Diag}(m_{l,s}) \, A_{l,s},$$

dengan $m_{l,s}$ sebagai *rank mask* biner. Rank aktif slot adalah

$$r_{l,s} = \|m_{l,s}\|_0.$$

Desain ini membuat rank tidak perlu benar-benar mengubah ukuran tensor pada runtime. Model hanya mengaktifkan sebagian dimensi low-rank yang memang dibutuhkan. Slot bank inilah yang berfungsi sebagai *fast plasticity* NH-LoRA.

5.4 Task-State Encoder (TSE)

Sebelum task ke-t dilatih penuh, model terlebih dahulu melakukan *warm-up sensing* pada beberapa batch awal untuk membentuk ringkasan kondisi task saat ini. Tujuannya adalah menghasilkan *task embedding* kompak yang cukup informatif untuk menggerakkan Horizon Planner secara stabil.

Misalkan $\mathcal{B}_t^{\text{warm}} = \{(x_n, y_n)\}_{n=1}^{N_w}$ adalah himpunan sampel dari fase *warm-up*. Dengan $f(\cdot)$ menyatakan ekstraktor fitur dari backbone beku beserta adapter yang sedang aktif, maka mean feature task saat ini didefinisikan sebagai

$$\mu_t = \frac{1}{N_w} \sum_{n=1}^{N_w} f(x_n). \tag{1}$$

Dispersi fitur diringkas melalui kovarians diagonal atau varians per dimensi:

$$\sigma_t = \frac{1}{N_w} \sum_{n=1}^{N_w} \big(f(x_n) - \mu_t\big)^{\odot 2}, \tag{2}$$

dengan $\odot 2$ menyatakan kuadrat elemen-per-elemen.

Untuk menangkap kebutuhan adaptasi awal, digunakan *gradient sketch* ringan pada parameter trainable. Secara praktis, statistik ini dapat dinyatakan sebagai norm gradien per kelompok parameter:

$$g_t = \Big[\|\nabla_{\Delta W_1} \mathcal{L}^{\text{warm}}\|_2,\ \|\nabla_{\Delta W_2} \mathcal{L}^{\text{warm}}\|_2,\ \ldots,\ \|\nabla_{\Delta W_M} \mathcal{L}^{\text{warm}}\|_2\Big], \tag{3}$$

dengan $\mathcal{L}^{\text{warm}}$ adalah loss pada fase *warm-up*, dan $\{\Delta W_m\}_{m=1}^{M}$ menyatakan himpunan modul adapter trainable yang dipantau.

Agar task baru dapat dibandingkan dengan pengalaman sebelumnya, dihitung pula sinyal kemiripan terhadap *history bank*. Jika $\mathcal{H}_{t-1} = \{h_1, \ldots, h_{t-1}\}$ adalah ringkasan histori task sebelumnya, maka similarity ringkas dapat ditulis sebagai

$$s_t = \begin{cases} 0, & t = 1, \\ \max\limits_{i \in \{1,\ldots,t-1\}} \cos\big(\psi_s(\mu_t),\, h_i\big), & t > 1, \end{cases} \tag{4}$$

dengan $\psi_s(\cdot)$ adalah proyeksi ringan agar mean feature saat ini berada pada ruang yang sama dengan representasi histori.

Selain itu, model juga menghitung uncertainty awal melalui rata-rata entropi prediksi:

$$e_t = \frac{1}{N_w} \sum_{n=1}^{N_w} \mathcal{H}\big(p_\theta(y \mid x_n)\big), \tag{5}$$

dengan $\mathcal{H}(\cdot)$ menyatakan entropi Shannon.

Seluruh statistik tersebut kemudian digabung menjadi vektor task mentah:

$$r_t = \big[\,\text{pool}(\mu_t);\ \text{pool}(\sigma_t);\ g_t;\ s_t;\ e_t\big]. \tag{6}$$

Task-State Encoder memetakan vektor ini ke *task embedding* berdimensi tetap:

$$z_t = \text{Encode}(\mu_t, \sigma_t, g_t, s_t, e_t) = W_2\, \phi\big(W_1\ \text{LN}(r_t) + b_1\big) + b_2, \tag{7}$$

dengan $\text{LN}(\cdot)$ adalah *Layer Normalization*, $\phi(\cdot)$ adalah fungsi aktivasi nonlinier seperti GELU atau ReLU, dan W_1, W_2, b_1, b_2 adalah parameter TSE yang dipelajari.

Agar *gradient sketch* g_t dapat dihitung sebelum struktur akhir task saat ini dimaterialisasi, NH-LoRA menggunakan *auxiliary warm-up head* sementara. Untuk setiap kelas baru c pada fase *warm-up*, bobot sementara diinisialisasi melalui *weight imprinting* dari prototype fitur:

$$w_c^{\text{tmp}} = \frac{\mu_c^{\text{warm}}}{\|\mu_c^{\text{warm}}\|_2}, \tag{8}$$

dengan μ_c^{warm} adalah rata-rata fitur sampel kelas c pada batch *warm-up*.

Berdasarkan head sementara tersebut, loss *warm-up* didefinisikan sebagai

$$\mathcal{L}^{\text{warm}} = \text{CE}\Big(y, \hat{y}^{\text{tmp}}\Big), \tag{9}$$

dengan $\hat{y}^{\text{tmp}}$ adalah prediksi dari *auxiliary warm-up head*. Loss ini hanya digunakan untuk membentuk statistik awal seperti g_t dan tidak dipakai sebagai head final untuk training penuh task ke-t.

Setelah Horizon Planner menghasilkan keputusan struktural, classifier head final untuk kelas baru dimaterialisasi pada fase ekspansi struktur, sedangkan *auxiliary* head dibuang. Dengan mekanisme ini, perhitungan g_t tetap terdefinisi tanpa menimbulkan ketergantungan sirkular antara planning dan ekspansi classifier.

Untuk task pertama, karena belum tersedia histori sebelumnya, sinyal similarity diatur ke nol, yaitu $s_1 = 0$. Dengan demikian, TSE pada fase *bootstrap* tetap dapat membentuk state awal dari statistik task saat ini tanpa bergantung pada informasi historis.

Secara intuitif, μ_t dan σ_t menangkap struktur representasi task baru, g_t mencerminkan kebutuhan adaptasi awal, s_t mengukur kedekatan terhadap pengalaman sebelumnya, sedangkan e_t merepresentasikan tingkat ketidakpastian model. TSE kemudian meringkas seluruh sinyal tersebut ke dalam z_t, yang selanjutnya digunakan oleh Horizon Planner untuk menghasilkan keputusan struktural pada adapter.

5.5 Horizon Planner (HP)

Horizon Planner (HP) bertugas memetakan ringkasan task saat ini ke keputusan struktural adapter pada setiap block terpilih. Secara umum, planner menerima *task embedding* z_t dari Task-State Encoder (TSE) serta ringkasan histori task sebelumnya dari *history bank*, lalu menghasilkan empat sinyal utama, yaitu *novelty score*, *conflict score*, *rank budget*, dan *consolidation signal*.

History bank dan ringkasan task. Setelah task ke-i selesai dipelajari, model tidak menyimpan ulang data mentah task tersebut, melainkan hanya menyimpan ringkasan statistiknya ke dalam *history bank*. Untuk task ke-i, ringkasan histori didefinisikan sebagai

$$h_i = \psi_h\Big(\big[\,\text{pool}(\bar{\mu}_i),\, \text{pool}(\bar{\sigma}_i),\, \bar{g}_i,\, \bar{u}_i,\, \bar{r}_i,\, \bar{e}_i\big]\Big) \in \mathbb{R}^{d_h}, \tag{10}$$

dengan $\bar{\mu}_i$ menyatakan ringkasan mean feature / prototype, $\bar{\sigma}_i$ menyatakan dispersi fitur, $\bar{g}_i$ adalah *gradient sketch*, $\bar{u}_i$ adalah ringkasan penggunaan slot dan shared core, $\bar{r}_i$ adalah ringkasan rank aktif, dan $\bar{e}_i$ adalah ringkasan uncertainty pada task ke-i. Fungsi $\psi_h(\cdot)$ merepresentasikan proyeksi ringan, misalnya multilayer perceptron (MLP) kecil, yang memetakan statistik mentah ke vektor histori berdimensi tetap.

Dengan demikian, *history bank* sebelum task ke-t dimulai dapat ditulis sebagai

$$\mathcal{H}_{t-1} = \{h_1, h_2, \ldots, h_{t-1}\}. \tag{11}$$

Sementara itu, TSE menghasilkan *task embedding* untuk task saat ini:

$$z_t \in \mathbb{R}^{d_z}. \tag{12}$$

History-aware aggregation (*HistAgg*). Agar planner tidak hanya membandingkan task saat ini dengan satu histori tertentu, NH-LoRA menggunakan agregasi histori yang dikondisikan oleh z_t. Untuk setiap task baru, pertama-tama dibentuk *query* dari state saat ini serta *key* dan *value* dari seluruh elemen *history bank*:

$$q_t = W_q z_t, \qquad k_i = W_k h_i, \qquad v_i = W_v h_i, \tag{13}$$

dengan W_q, W_k, dan W_v merupakan proyeksi terlatih.

Bobot perhatian terhadap setiap histori kemudian dihitung sebagai

$$a_i^{(t)} = \frac{\exp\left(\frac{q_t^\top k_i}{\sqrt{d_a}}\right)}{\sum_{j=1}^{t-1} \exp\left(\frac{q_t^\top k_j}{\sqrt{d_a}}\right)}, \tag{14}$$

sehingga ringkasan histori teragregasi didefinisikan sebagai

$$\tilde{h}_t = \sum_{i=1}^{t-1} a_i^{(t)} v_i. \tag{15}$$

Untuk task pertama, karena belum ada histori sebelumnya, digunakan

$$\tilde{h}_1 = \mathbf{0}. \tag{16}$$

Representasi planner per-layer. Planner bekerja pada setiap block / layer terpilih. Untuk layer ke-l, planner membentuk representasi gabungan antara state task saat ini dan histori teragregasi:

$$u_l^t = [z_t;\, \tilde{h}_t;\, |z_t - \tilde{h}_t|;\, z_t \odot \tilde{h}_t;\, e_l], \tag{17}$$

dengan e_l adalah *layer embedding* yang membedakan identitas tiap layer, dan $\odot$ menyatakan perkalian elemen-per-elemen.

Selanjutnya, representasi laten planner didefinisikan sebagai

$$r_l^t = \phi_l(u_l^t), \tag{18}$$

dengan $\phi_l(\cdot)$ merupakan MLP kecil untuk layer ke-l.

Empat keluaran planner. Dari representasi laten r_l^t, Horizon Planner menghasilkan empat sinyal utama:

$$\nu_l^t = \sigma\left(w_{\nu,l}^\top r_l^t + b_{\nu,l}\right), \tag{19}$$

$$\xi_l^t = \sigma\left(w_{\xi,l}^\top r_l^t + b_{\xi,l}\right), \tag{20}$$

$$\rho_l^t = r_{\min} + (r_{\max} - r_{\min}) \cdot \sigma\left(w_{\rho,l}^\top r_l^t + b_{\rho,l}\right), \tag{21}$$

$$\kappa_l^t = \sigma\left(w_{\kappa,l}^\top r_l^t + b_{\kappa,l}\right), \tag{22}$$

dengan $\sigma(\cdot)$ adalah fungsi sigmoid.

Di sini, $\nu_l^t \in (0,1)$ menyatakan *novelty score*, $\xi_l^t \in (0,1)$ menyatakan *conflict score*, $\rho_l^t \in [r_{\min}, r_{\max}]$ menyatakan *rank budget*, dan $\kappa_l^t \in (0,1)$ menyatakan *consolidation signal*.

Dalam implementasi diskret, *rank budget* dapat diubah menjadi rank aktif integer:

$$\hat{r}_l^t = \mathrm{clip}\Big(\mathrm{round}(\rho_l^t),\, r_{\min},\, r_{\max}\Big). \tag{23}$$

Shared gate untuk shared core. Selain empat sinyal utama di atas, pada implementasi NH-LoRA digunakan pula *shared gate* untuk mengontrol kontribusi shared core pada layer ke-l:

$$\beta_l^t = \sigma\Big(w_{\beta,l}^\top r_l^t + b_{\beta,l}\Big), \tag{24}$$

dengan $\sigma(\cdot)$ adalah fungsi sigmoid, sehingga $\beta_l^t \in (0,1)$.

Dengan tambahan ini, bobot efektif pada layer ke-l dapat ditulis ulang sebagai

$$W_l^{\text{eff}}(x,t) = W_l^0 + \beta_l^t \Delta W_l^{\text{shared}} + \sum_{s \in \mathcal{S}_l^{\text{act}}(x)} \alpha_{l,s}(x)\, \Delta W_{l,s}^{\text{slot}}. \tag{25}$$

Secara intuitif, β_l^t mengatur seberapa besar pengetahuan lintas task dari shared core perlu digunakan pada task saat ini. Nilai β_l^t yang tinggi menunjukkan kecenderungan reuse shared knowledge yang lebih besar, sedangkan nilai yang lebih rendah menandakan bahwa adaptasi lebih banyak bergantung pada jalur slot spesifik-task.

Aturan keputusan struktural. Berdasarkan kombinasi ν_l^t dan ξ_l^t, planner memilih salah satu aksi struktural pada layer ke-l:

$$\mathcal{A}_l^t = \begin{cases} \texttt{reuse\_shared}, & \nu_l^t < \tau_\nu \ \wedge \ \xi_l^t < \tau_\xi, \\ \texttt{expand\_rank\_existing\_slot}, & \nu_l^t \geq \tau_\nu \ \wedge \ \xi_l^t < \tau_\xi, \\ \texttt{open\_new\_slot}, & \nu_l^t \geq \tau_\nu \ \wedge \ \xi_l^t \geq \tau_\xi, \\ \texttt{freeze\_old\_strong\_retention}, & \nu_l^t < \tau_\nu \ \wedge \ \xi_l^t \geq \tau_\xi. \end{cases} \tag{26}$$

Interpretasinya adalah sebagai berikut. Ketika novelty dan conflict sama-sama rendah, task baru dipandang masih cukup dekat dengan pengetahuan yang telah ada, sehingga model cukup menggunakan shared core dengan penyesuaian kecil. Ketika novelty tinggi tetapi conflict rendah, task baru memerlukan kapasitas tambahan, namun masih cukup kompatibel dengan struktur yang ada, sehingga planner menaikkan rank pada slot yang paling relevan. Ketika novelty dan conflict sama-sama tinggi, task baru dipandang cukup berbeda sekaligus berpotensi mengganggu pengetahuan lama, sehingga slot baru dibuka. Sebaliknya, ketika novelty rendah tetapi conflict tinggi, planner menahan perubahan agresif dan mengaktifkan retensi kuat agar interferensi terhadap pengetahuan lama dapat ditekan.

Pemilihan slot untuk ekspansi rank. Jika aksi yang dipilih adalah `expand_rank_existing_slot`, maka planner memilih slot lama yang paling kompatibel dengan task saat ini. Misalkan $c_{l,s}$ adalah centroid atau ringkasan representasi untuk slot ke-s pada layer ke-l, maka slot target didefinisikan sebagai

$$s_l^\star = \arg\max_{s \in \mathcal{S}_l} \text{Aff}(z_t, c_{l,s}), \tag{27}$$

dengan $\text{Aff}(\cdot,\cdot)$ dapat berupa cosine similarity atau skor afinitas lain yang setara.

Rank slot terpilih kemudian diperbarui menjadi

$$r_{l,s^\star}^{(t)} = \min\Big(r_{l,s^\star}^{(t-1)} + \Delta r_l^t,\ r_{\max}\Big), \tag{28}$$

dengan

$$\Delta r_l^t = \max\Big(1,\ \hat{r}_l^t - r_{l,s^\star}^{(t-1)}\Big). \tag{29}$$

Batas kapasitas slot dan kandidat ekspansi. Untuk menjaga pertumbuhan parameter tetap terkendali, setiap layer ke-l memiliki batas maksimum jumlah slot:

$$|\mathcal{S}_l| \leq S_{\max}. \tag{30}$$

Jika aksi planner adalah `open_new_slot` tetapi jumlah slot pada layer tersebut sudah mencapai batas, yaitu $|\mathcal{S}_l| = S_{\max}$, maka aksi tersebut dialihkan menjadi `expand_rank_existing_slot` dengan menggunakan slot target $s_l^\star$ yang telah didefinisikan sebelumnya. Dengan demikian, pertumbuhan struktur tetap terkendali tanpa mengubah mekanisme afinitas yang digunakan untuk memilih slot lama yang paling kompatibel.

Inisialisasi slot baru. Jika aksi yang dipilih adalah `open_new_slot`, maka planner membuat slot baru pada layer ke-l dengan rank awal

$$r_{l,\text{new}}^{(t)} = \hat{r}_l^t. \tag{31}$$

Slot baru ini diperlakukan sebagai jalur *fast plasticity* untuk menyerap subspace yang belum dapat direpresentasikan dengan baik oleh shared core maupun slot lama.

Sinyal konsolidasi. Sinyal κ_l^t tidak langsung memengaruhi forward pass pada task saat ini, tetapi dipakai pada tahap pasca-task untuk mengarahkan proses merge / freeze / prune. Secara sederhana, flag konsolidasi dapat ditulis sebagai

$$c_l^t = \mathbb{I}\Big[\kappa_l^t \geq \tau_\kappa\Big], \tag{32}$$

dengan $\mathbb{I}[\cdot]$ adalah fungsi indikator. Ketika $c_l^t = 1$, slot yang stabil dan berguna diperbolehkan untuk dipertimbangkan masuk ke proses konsolidasi menuju shared core.

Ringkasan fungsi planner. Dengan notasi di atas, Horizon Planner pada layer ke-l dapat diringkas sebagai

$$(\nu_l^t,\ \xi_l^t,\ \rho_l^t,\ \kappa_l^t) = P_l(z_t, \mathcal{H}_{t-1}), \tag{33}$$

dengan

$$P_l(z_t, \mathcal{H}_{t-1}) = \text{Heads}_l\Big(\phi_l([z_t; \tilde{h}_t; |z_t - \tilde{h}_t|; z_t \odot \tilde{h}_t; e_l])\Big). \tag{34}$$

Secara intuitif, z_t menangkap kondisi task saat ini, $\tilde{h}_t$ merangkum histori yang paling relevan terhadap task tersebut, sedangkan empat keluaran planner mengontrol arah adaptasi struktural secara layer-wise.

Materialisasi aksi struktural. Empat sinyal utama yang dihasilkan planner tidak langsung dipakai sebagai konfigurasi *forward* final. Sebaliknya, sinyal tersebut terlebih dahulu dipetakan oleh fungsi *materialize action* menjadi konfigurasi struktural layer-wise, misalnya *shared gate* β_l^t, himpunan slot aktif kandidat, serta *rank configuration* untuk setiap slot yang diaktifkan. Dengan demikian, Horizon Planner beroperasi pada level pengambilan keputusan, sedangkan parameter struktural yang benar-benar dipakai pada *forward pass* dibentuk pada tahap materialisasi aksi.

5.6 Instance Router (IR)

Pada saat *forward*, router tidak mengaktifkan seluruh slot sekaligus, melainkan hanya memilih sejumlah kecil slot paling relevan untuk setiap instance. Untuk sample x pada layer ke-l, router terlebih dahulu membentuk query instance:

$$q_l(x) = \phi_l(\mathrm{Pool}(H_l(x))) \,, \tag{35}$$

dengan $H_l(x)$ adalah representasi token pada layer ke-l dan $\phi_l(\cdot)$ adalah proyeksi ringan yang dipelajari.

Berdasarkan query tersebut, himpunan slot aktif didefinisikan sebagai

$$\mathcal{S}_l^{\mathrm{act}}(x) = \mathrm{TopK}_{s \in \mathcal{S}_l} \cos\big(q_l(x), k_{l,s}\big), \tag{36}$$

dengan

$$|\mathcal{S}_l^{\mathrm{act}}(x)| = K_{\mathrm{route}}, \qquad K_{\mathrm{route}} \ll |\mathcal{S}_l|. \tag{37}$$

Koefisien routing kemudian dihitung hanya pada slot-slot dalam $\mathcal{S}_l^{\mathrm{act}}(x)$ melalui *TopK-Softmax*:

$$\alpha_{l,s}(x) = \mathrm{TopKSoftmax}\left(\frac{\cos(q_l(x), k_{l,s})}{\tau}\right), \qquad s \in \mathcal{S}_l^{\mathrm{act}}(x). \tag{38}$$

Dengan demikian, bobot efektif pada layer ke-l dapat ditulis sebagai

$$W_l^{\mathrm{eff}}(x,t) = W_l^0 + \beta_l^t \Delta W_l^{\mathrm{shared}} + \sum_{s \in \mathcal{S}_l^{\mathrm{act}}(x)} \alpha_{l,s}(x)\, \Delta W_{l,s}^{\mathrm{slot}}. \tag{39}$$

Desain ini menjaga inference tetap jarang (*sparse*), mendorong *selective pathway reuse*, dan mengurangi interferensi antar-slot karena hanya sebagian kecil slot yang aktif untuk setiap instance.

5.7 Incremental Classifier Head (ICH)

Classifier yang disarankan adalah *normalized cosine classifier*:

$$\ell_c(x) = \tau_{\mathrm{cls}} \cdot \cos(f_\theta(x), w_c) \,.$$

Setiap task baru menambahkan bobot kelas baru sehingga

$$W_{\mathrm{cls}}^{(t)} = \left[W_{\mathrm{cls}}^{(t-1)}; W_{\mathrm{new}}\right].$$

Inisialisasi kelas baru bisa memakai *weight imprinting* dari rata-rata fitur kelas saat ini, atau *random init* dengan *warm start* beberapa epoch.

Cosine classifier umumnya lebih stabil dibanding linear head biasa untuk setup frozen backbone plus adapter kecil, karena skala norm fitur tidak terlalu mendominasi proses incremental.

5.8 Consolidation and Homeostasis Unit (CHU)

Sesudah task selesai, NH-LoRA menjalankan consolidation. Untuk setiap slot dihitung utility $u_{l,s}$, stability $\psi_{l,s}$, dan redundancy $r_{l,s}^{\mathrm{red}}$ terhadap slot lain atau *shared core*.

Pada implementasi awal, keputusan *merge*, *prune*, atau *keep/freeze* ditentukan melalui aturan heuristik sederhana berbasis usage, stability, redundancy, dan *consolidation flag*.

Usage slot ke-s pada layer ke-l didefinisikan sebagai rata-rata koefisien routing:

$$u_{l,s}^{(t)} = \frac{1}{N_t} \sum_{x \in D_t} \alpha_{l,s}(x). \tag{40}$$

Stability slot didefinisikan dari besar perubahan parameternya selama task ke-t:

$$\psi_{l,s}^{(t)} = \exp\left(- \frac{\left\| \Delta W_{l,s}^{\text{end}} - \Delta W_{l,s}^{\text{start}} \right\|_F}{\left\| \Delta W_{l,s}^{\text{start}} \right\|_F + \epsilon} \right). \tag{41}$$

Redundancy slot terhadap slot lain dapat dihitung sebagai

$$r_{l,s}^{\text{red}} = \max_{u \neq s} \cos\Big(\text{vec}(\Delta W_{l,s}), \text{vec}(\Delta W_{l,u}) \Big). \tag{42}$$

Berdasarkan tiga besaran tersebut, keputusan pasca-task didefinisikan sebagai:

$$\texttt{merge}(l,s) \iff u_{l,s}^{(t)} \geq \tau_u^+ \ \wedge \ \psi_{l,s}^{(t)} \geq \tau_\psi \ \wedge \ c_l^t = 1, \tag{43}$$

$$\texttt{prune}(l,s) \iff u_{l,s}^{(t)} \leq \tau_u^- \ \wedge \ r_{l,s}^{\text{red}} \geq \tau_r, \tag{44}$$

dan jika kedua kondisi di atas tidak terpenuhi, maka slot dipertahankan atau dibekukan:

$$\texttt{keep/freeze}(l,s) \iff \neg\, \texttt{merge}(l,s) \ \wedge \ \neg\, \texttt{prune}(l,s). \tag{45}$$

Dengan formulasi heuristik di atas, slot yang sering digunakan, stabil, dan ditandai layak konsolidasi akan dipertimbangkan untuk di-*merge* ke shared core. Sebaliknya, slot yang jarang dipakai dan sangat redundan akan dipruning, sedangkan sisanya dipertahankan atau dibekukan sebagai memori spesifik-task. Formulasi ini sengaja dibuat sederhana agar implementasi awal tetap stabil, sambil tetap mempertahankan fungsi *homeostasis* untuk mengendalikan pertumbuhan kapasitas model.

6 Pemetaan neuromorphic yang valid untuk NH-LoRA

Prinsip neuromorphic	**Interpretasi pada NH-LoRA**
Fast plasticity	Slot adapter baru cepat menyerap subspace task baru.
Slow plasticity	Shared Core LoRA diupdate lebih konservatif agar menyimpan pengetahuan lintas task.
Structural plasticity	Kapasitas model boleh grow / rank-expand / merge / prune.
Metaplasticity	Horizon Planner mengatur kapan model harus agresif beradaptasi dan kapan harus konservatif.
Homeostasis	Growth penalty, rank regularization, dan pruning menjaga pertumbuhan kapasitas tetap terkendali.

Istilah "Neuromorphic" di sini digunakan sebagai framing *brain-inspired design principle*, bukan klaim bahwa NH-LoRA adalah *spiking neural network* murni. Pemakaian istilah ini tetap valid selama paper secara eksplisit menjelaskan bahwa *inspiration-level mapping* berada pada *multi-timescale plasticity*, *metaplastic control*, dan *homeostatic capacity regulation*.

7 Objektif Training

Secara keseluruhan, objektif training NH-LoRA dirumuskan sebagai kombinasi dari loss klasifikasi, distilasi, retensi fitur, regularisasi subspace, regularisasi rank, penalti pertumbuhan struktur, serta regularisasi routing:

$$\mathcal{L} = \mathcal{L}_{\text{cls}} + \lambda_{\text{kd}}\mathcal{L}_{\text{kd}} + \lambda_{\text{feat}}\mathcal{L}_{\text{feat}} + \lambda_{\text{orth}}\mathcal{L}_{\text{orth}} + \lambda_{\text{rank}}\mathcal{L}_{\text{rank}} + \lambda_{\text{grow}}\mathcal{L}_{\text{grow}} + \lambda_{\text{route}}\mathcal{L}_{\text{route}}.$$

Sedangkan untuk task pertama loss yang aktif hanya:

$$\mathcal{L}^{(1)} = \mathcal{L}_{\text{cls}} + \lambda_{\text{rank}}\mathcal{L}_{\text{rank}} + \lambda_{\text{route}}\mathcal{L}_{\text{route}} + \lambda_{\text{orth}}\mathcal{L}^{(1)}_{\text{orth}}.$$

dengan $\mathcal{L}^{(1)}_{\text{kd}} = 0$, $\mathcal{L}^{(1)}_{\text{feat}} = 0$, dan $\mathcal{L}^{(1)}_{\text{grow}}$ dapat dimatikan atau dibuat sangat kecil.

Masing-masing komponen memiliki peran sebagai berikut:

- $\mathcal{L}_{\text{cls}}$ digunakan untuk mempelajari kelas-kelas pada task saat ini.
- $\mathcal{L}_{\text{kd}}$ digunakan untuk mempertahankan respons model terhadap kelas lama melalui distilasi logit dari snapshot model sebelumnya.
- $\mathcal{L}_{\text{feat}}$ digunakan untuk menjaga agar representasi fitur pada layer tertentu tidak bergeser terlalu jauh setelah mempelajari task baru.
- $\mathcal{L}_{\text{orth}}$ digunakan untuk mendorong slot baru agar tidak menempati subspace yang terlalu tumpang tindih dengan slot yang sudah ada.
- $\mathcal{L}_{\text{rank}}$ digunakan untuk menekan rank aktif agar tetap kecil dan efisien.
- $\mathcal{L}_{\text{grow}}$ digunakan untuk mengontrol pembukaan slot baru agar ekspansi struktur hanya terjadi saat benar-benar dibutuhkan.
- $\mathcal{L}_{\text{route}}$ digunakan untuk mencegah collapse pada router, sehingga distribusi pemilihan slot tetap seimbang.

Berikut adalah definisi tiap komponen loss.

1) Classification loss

Loss klasifikasi didefinisikan sebagai cross-entropy antara label target y dan prediksi model $\hat{y}$:

$$\mathcal{L}_{\text{cls}} = \text{CE}(y, \hat{y}).$$

2) Logit distillation loss

Untuk menjaga pengetahuan pada kelas-kelas lama, digunakan teacher berupa snapshot model setelah task $t-1$. Loss distilasi logit dirumuskan sebagai:

$$\mathcal{L}_{\text{kd}} = \text{KL}(p_{\theta^{t-1}}(y_{\text{old}} \mid x; T) \,\|\, p_\theta(y_{\text{old}} \mid x; T)) \,.$$

Loss ini diterapkan pada sampel task saat ini dengan skema retensi bergaya *Learning without Forgetting* (LwF).

3) Feature retention loss

Selain mempertahankan logit, model juga didorong untuk menjaga kestabilan representasi internal pada layer-layer tertentu:

$$\mathcal{L}_{\text{feat}} = \sum_{l \in \mathcal{L}_{\text{ret}}} \left\| h_l^t(x) - h_l^{t-1}(x) \right\|_2^2 .$$

Di sini, $\mathcal{L}_{\text{ret}}$ menyatakan himpunan layer yang dipantau untuk retensi fitur.

4) Orthogonality loss antar slot

Agar slot baru tidak belajar arah subspace yang terlalu mirip dengan slot lama, digunakan regularisasi ortogonalitas:

$$\mathcal{L}_{\text{orth}} = \sum_{l} \sum_{s \neq u} \left\| A_{l,s}^{\top} A_{l,u} \right\|_F^2 .$$

Loss ini membantu menjaga diversitas representasi antar slot dalam setiap layer.

5) Rank regularization

Untuk mendorong efisiensi kapasitas, rank aktif pada tiap slot diregularisasi melalui:

$$\mathcal{L}_{\text{rank}} = \sum_{l,s} \| m_{l,s} \|_1 .$$

Regularisasi ini mendorong jumlah dimensi low-rank yang aktif tetap kecil.

6) Growth penalty

Agar struktur model tidak tumbuh secara berlebihan, pembukaan slot baru dikenai penalti:

$$\mathcal{L}_{\text{grow}} = \sum_{l} \mathbb{1}[\text{new slot at } l] .$$

Dengan demikian, ekspansi slot hanya akan dipilih ketika benar-benar diperlukan oleh kondisi task.

7) Router balance / anti-collapse loss

Untuk mencegah router terus-menerus memilih slot yang sama, digunakan regularisasi keseimbangan routing:

$$\mathcal{L}_{\text{route}} = \sum_{l} \text{KL}(\bar{\alpha}_l \,\|\, \text{Uniform}) .$$

Di sini, $\bar{\alpha}_l$ merepresentasikan distribusi rata-rata routing pada layer l. Loss ini bersifat opsional, tetapi berguna untuk mengurangi risiko *routing collapse*.

8 Pseudo-code arsitektur

Pseudo-code berikut memperlihatkan *forward computation* pada satu NH-augmented ViT block.

Listing 1: NH-LoRA block forward

```
def NH_BLOCK_FORWARD(h_l, layer_id, task_state, slot_bank, shared_core, router,
    planner_out):
    beta_l   = planner_out[layer_id]["shared_gate"]
    rank_cfg = planner_out[layer_id]["rank_cfg"]
    active   = planner_out[layer_id]["active_slots"]
```

```
    q = router[layer_id].project(pool_tokens(h_l))
    alpha = topk_softmax(
        cosine(q, slot_bank[layer_id].keys[active]),
        k=2,
        tau=router[layer_id].tau
    )

    delta_shared = beta_l * (shared_core[layer_id].B @ shared_core[layer_id].A)

    delta_slots = 0.0
    for j, s in enumerate(active):
        mask_s = prefix_rank_mask(rank_cfg[s], slot_bank[layer_id].r_max)
        delta_s = slot_bank[layer_id].B[s] @ diag(mask_s) @ slot_bank[layer_id].A[s]
        delta_slots += alpha[j] * delta_s

    W_eff = frozen_weight[layer_id] + delta_shared + delta_slots
    out = linear_or_attention_projection(h_l, W_eff)
    return out
```

Interpretasi utamanya adalah bahwa planner bekerja di level struktur, sedangkan router bekerja di level instance. Keduanya saling melengkapi: planner menentukan kapasitas yang boleh aktif, router menentukan slot mana yang dipakai untuk sample tertentu.

9 Pseudo-code training loop

Untuk task pertama, NH-LoRA beroperasi dalam mode bootstrap, di mana history-dependent planning dan teacher-based retention dinonaktifkan, dan model menginisialisasi shared memory dan struktur slot minimalnya hanya dari task saat ini.

Pseudo-code berikut menggambarkan training NH-LoRA pada rehearsal-free class-incremental learning.

Listing 2: NH-LoRA training loop for rehearsal-free CIL

```
initialize frozen ViT backbone
insert shared LoRA cores into selected blocks
initialize empty slot banks for all blocks
initialize classifier head for first task
history_bank = []

for task_id, D_t in enumerate(task_sequence, start=1):

    # Phase A: warm-up sensing
    warm_batches = sample_warmup_batches(D_t, n_batches=W)
    feat_stats   = compute_feature_statistics(warm_batches, backbone, current_model)
    grad_stats   = compute_gradient_sketch(warm_batches, current_model)

    if task_id == 1:
        # No previous task history exists yet
        sim_stats = zero_similarity_stats()

        task_state = task_state_encoder(
            feat_mean=feat_stats["mean"],
            feat_cov=feat_stats["cov_diag"],
            entropy=feat_stats["entropy"],
            grad=grad_stats,
```

```
            similarity=sim_stats
        )

        # Bootstrap policy for the first task
        planner_out = {}
        for l in selected_blocks:
            planner_out[l] = bootstrap_structure_policy(
                layer=l,
                shared_core=shared_core[l],
                slot_bank=slot_bank[l],
                shared_rank=r_shared_init,
                boot_slot_rank=r_boot_init,
                open_bootstrap_slot=True,
                consolidate=False
            )

        teacher_model = None
        old_classes = []

    else:
        sim_stats = compare_with_history(feat_stats, history_bank)

        task_state = task_state_encoder(
            feat_mean=feat_stats["mean"],
            feat_cov=feat_stats["cov_diag"],
            entropy=feat_stats["entropy"],
            grad=grad_stats,
            similarity=sim_stats
        )

        # Phase B: horizon planning
        planner_out = {}
        for l in selected_blocks:
            novelty, conflict, rank_budget, consolidate = planner[l](task_state,
history_bank)

            # Convert planner signals into concrete structural config
            # such as shared gate, active slots, and rank configuration
            if novelty < tau_n and conflict < tau_c:
                action = "reuse_shared"
            elif novelty >= tau_n and conflict < tau_c:
                action = "expand_rank_existing_slot"
            elif novelty >= tau_n and conflict >= tau_c:
                action = "open_new_slot"
            else:
                action = "freeze_old_strong_retention"

            planner_out[l] = materialize_action(
                layer=l,
                action=action,
                rank_budget=rank_budget,
                consolidate=consolidate,
                slot_bank=slot_bank[l]
            )

        teacher_model = copy_frozen_snapshot(current_model)
        old_classes = seen_classes_before(task_id)
```

```
# Phase C: structure expansion
apply_structure_changes(slot_bank, shared_core, planner_out)
expand_classifier_head_for_new_classes(classifier, D_t.classes)

# Phase D: train current task
for epoch in range(num_epochs):
    for x, y in loader(D_t):
        logits, features, route_info = current_model(x, task_state, planner_out)

        loss_cls   = cross_entropy(logits, y)
        loss_rank  = rank_penalty(slot_bank)
        loss_route = routing_balance_loss(route_info)

        if task_id == 1:
            # No previous model / no old classes yet
            loss_orth = slot_orthogonality(slot_bank)

            loss = (
                loss_cls
                + lam_orth  * loss_orth
                + lam_rank  * loss_rank
                + lam_route * loss_route
            )

        else:
            with no_grad():
                teacher_logits, teacher_features = teacher_model(x)

            loss_kd    = kd_loss(
                logits[:, old_classes],
                teacher_logits[:, old_classes],
                T=T_kd
            )
            loss_feat  = feature_retention(
                features,
                teacher_features,
                layers=retention_layers
            )
            loss_orth  = slot_orthogonality(slot_bank)
            loss_grow  = growth_penalty(planner_out)

            loss = (
                loss_cls
                + lam_kd    * loss_kd
                + lam_feat  * loss_feat
                + lam_orth  * loss_orth
                + lam_rank  * loss_rank
                + lam_grow  * loss_grow
                + lam_route * loss_route
            )

        optimizer.zero_grad()
        loss.backward()
        optimizer.step()

# Phase E: consolidation
usage_stats = estimate_slot_usage(D_t, current_model, task_state, planner_out)
stability   = estimate_slot_stability(slot_bank)
```

```
    for l in selected_blocks:
        for s in slot_bank[l].slots:
            if should_merge(s, usage_stats, stability, planner_out[l]["
    consolidate_flag"]):
                merge_slot_into_shared(shared_core[l], slot_bank[l][s], merge_rate=
    lam_merge)
            elif should_prune(s, usage_stats, stability):
                prune_slot(slot_bank[l], s)
            else:
                freeze_or_keep(slot_bank[l], s)

    # Phase F: save task summary
    history_bank.append(build_task_summary(D_t, feat_stats, grad_stats, usage_stats)
    )
```

10 Rancangan evaluasi eksperimen

Untuk versi paper yang kuat, evaluasi NH-LoRA disusun agar mencakup *sanity check*, *fine-grained stress*, *style/domain shift*, dan *long-sequence heterogeneity*. Framework yang paling natural untuk menjalankannya adalah PILOT karena toolbox ini memang menyediakan benchmark dan baseline PTM-based CIL seperti L2P dan CODA-Prompt.

Benchmark	**Peran di paper**	**Statistik ringkas**	**Split awal yang disarankan**
CIFAR-100	Sanity check, debugging, ablation cepat	100 kelas; 600 gambar per kelas	10 tasks × 10 classes
CUB-200-2011	Fine-grained CIL	200 kategori; 11,788 gambar	20 tasks × 10 classes
ImageNet-R	Style / domain shift within class recognition	200 kelas; 30,000 rendition images	10 tasks × 20 classes
OmniBenchmark (Jika sempat)	Long-sequence stress test	21 realm-wise datasets; 7,372 concepts; 1,074,346 images	Diletakkan sebagai stress test terpisah

Baseline utama yang fair untuk paper inti adalah L2P atau CODA-Prompt. Keduanya berasal dari keluarga prompt-based rehearsal-free continual learning di atas pre-trained ViT. Untuk paper yang lebih sulit dibantah, sangat disarankan menambah minimal satu baseline LoRA-based, misalnya CL-LoRA, agar perbandingan menjadi prompt vs LoRA vs NH-LoRA.

- **Metrik utama:** final average accuracy, average incremental accuracy, forgetting, backward transfer, last-task accuracy.
- **Metrik efisiensi:** parameter growth per task, total active rank, jumlah opened/pruned slots, training time per task, inference overhead.
- **Ablasi (jika sempat):** tanpa planner, tanpa shared core, tanpa dynamic rank, tanpa new-slot expansion, tanpa consolidation, tanpa instance router, novelty-only planner, dan conflict-only planner.

11 Catatan implementasi

Rank mask sebaiknya diimplementasikan dengan kapasitas maksimum tetap $r_{\max}$ dan aktivasi parsial melalui mask, bukan dengan benar-benar merealokasi tensor setiap kali rank berubah.

History bank tidak perlu menyimpan data mentah. Cukup simpan *summary feature statistics*, *similarity anchors*, dan *usage summary* dari slot-slot sebelumnya.

Planner dapat diawali dengan MLP kecil yang bekerja per-layer. Setelah baseline stabil, barulah dipertimbangkan planner yang lebih kuat seperti *hypernetwork* ringan atau transformer kecil di atas *history summaries*.

12 Referensi

[**1**] Sun, H.-L. et al. (2025). *PILOT: A Pre-Trained Model-Based Continual Learning Toolbox*. Science China Information Sciences. https://doi.org/10.1007/s11432-024-4276-4

[**2**] Zhou, D.-W. et al. (2024). *Continual Learning with Pre-Trained Models: A Survey*. IJCAI 2024. https://www.ijcai.org/proceedings/2024/924

[**3**] Wang, Z. et al. (2022). *Learning to Prompt for Continual Learning*. CVPR 2022 / OpenReview. https://openreview.net/forum?id=RzXb6a3H3rs

[**4**] Smith, J. S. et al. (2023). *CODA-Prompt: COntinual Decomposed Attention-Based Prompting for Rehearsal-Free Continual Learning*. CVPR 2023. https://openaccess.thecvf.com/content/CVPR2023/html/Smith_CODA-Prompt_COntinual_Decomposed_Attention-Based_Prompting_for_Rehearsal-Free_Continual_Learning_CVPR_2023_paper.html

[**5**] He, J., Duan, Z., & Zhu, F. (2025). *CL-LoRA: Continual Low-Rank Adaptation for Rehearsal-Free Class-Incremental Learning*. CVPR 2025. https://openaccess.thecvf.com/content/CVPR2025/html/He_CL-LoRA_Continual_Low-Rank_Adaptation_for_Rehearsal-Free_Class-Incremental_Learning_CVPR_2025_paper.html

[**6**] Yu, J., Zhuge, Y., Zhang, L., Hu, P., Wang, D., Lu, H., & He, Y. (2024). *Boosting Continual Learning of Vision-Language Models via Mixture-of-Experts Adapters*. CVPR 2024. https://arxiv.org/abs/2403.11549

[**7**] Yoon, J., Yang, E., Lee, J., & Hwang, S. J. (2018). *Lifelong Learning with Dynamically Expandable Networks*. ICLR 2018. https://openreview.net/forum?id=Sk7KsfW0-

[**8**] Han, B., Zhao, F., Zeng, Y., Pan, W., & Shen, G. (2023). *Enhancing Efficient Continual Learning with Dynamic Structure Development of Spiking Neural Networks*. IJCAI 2023. https://arxiv.org/abs/2308.04749

[**9**] Lu, H., Zhao, C., Xue, J., Yao, L., Moore, K., & Gong, D. (2024). *Adaptive Rank, Reduced Forgetting: Continual Learning with Dynamic Rank-Selective LoRA*. arXiv preprint arXiv:2412.01004. https://arxiv.org/abs/2412.01004

[**10**] Zhou, D.-W., Wang, F.-Y., Ye, H.-J., Ma, L., Pu, S., & Zhan, D.-C. (2022). *Forward Compatible Few-Shot Class-Incremental Learning*. CVPR 2022. https://openaccess.thecvf.com/content/CVPR2022/papers/Zhou_Forward_Compatible_Few-Shot_Class-Incremental_Learning_CVPR_2022_paper.pdf

[**11**] CIFAR-100 official dataset page. https://www.cs.toronto.edu/~kriz/cifar.html

[**12**] CUB-200-2011 official dataset page. https://www.vision.caltech.edu/datasets/cub_200_2011/

[**13**] ImageNet-R official repository. https://github.com/hendrycks/imagenet-r

[**14**] OmniBenchmark official project page. https://zhangyuanhan-ai.github.io/OmniBenchmark/