Запропоновано формальний, але інтуїтивно зрозумілий алгоритм, який автоматично формує дерево рівнів / класів, спираючись на (і) відстані між векторними представленнями класів, (іі) величину плутанини між ними та (ііі) прості порогові правила. Алгоритм узгоджує емпіричні спостереження авторів щодо трирівневої схеми «H / V → T / B / O → G / M / Z»  й узагальнює їх до автоматично масштабованої процедури.

# Запропонований метод

Вхідні дані

| Позначення | Розмірність | Пояснення || ---------------------------------- | ------------------------------------------------------------ | --------------------------------------------------------------------- || D=\{(x\_i,y\_i)\}\_{i=1}^{N} | N\times(зображення + мітка) | Навчальний набір кадрів БПЛА з їхніми початковими мітками || L=\{l\_1,\dots ,l\_m\} | m | Базові класи (тобто найдрібніші, що вже не діляться) || m | скаляр | Кількість базових класів || B | функція \mathbb{R}^{H\times W\times3}\!\to\!\mathbb{R}^{d} | Back-bone, який перетворює ROI у вектор ознак розмірності d || λ\in[0,1] | скаляр | Вага між «просторовою відстанню» й «частотою помилкових сплутувань» || δ\_{\min} | скаляр | Мінімальна відстань між кластерами, за якої їх ще розділяємо || ε\_{\max} | скаляр | Максимально допустима частка помилкових сплутувань усередині кластера || Θ=(\text{FPS}\_{\min},\,T\_{\max}) | вектор 2\times1 | Мінімальна швидкість (кадрів/с) та максимальна латентність (мс) |

## Крок 1: Витяг ознак для усіх ROI

1.1 Для кожного зображення x\_i виконуємо первинне детектування (будь‑який SOTA‑детектор) та отримуємо множину ROI R\_i=\{r\_{i1},\dots ,r\_{ik}\}.

1.2 Для кожного ROI обчислюємо вектор ознак

## Крок 2: Побудова «профілю» класу

2.1 Для кожного атомарного класу l\in L збираємо множину ознак F\_l=\{f\_{ij}\mid y\_{ij}=l\}.

2.2 Обчислюємо центроїд

2.3 Одноразово навчаємо одноетапний класифікатор без ієрархії C^{\text{flat}} (наприклад, простий MLP) на всіх f\_{ij} та формуємо матрицю плутанини

де \text{FP}(p\to q) — кількість неправильних присвоєнь p\mapsto q.

Тут під одноетапним класифікатором без ієрархії C^{\text{flat}} розуміємо модель, що одразу прогнозує один з усіх m базових класів без ієрархії.

## Крок 3: Визначення міри близькості між класами

Комбінуємо геометричну й емпіричну інформацію:

Чим більшим є M\_{pq} — тим легше класи розділити.

## Крок 4: Ієрархічне агломеративне групування

4.1 Будуємо повнозважений граф з вагами M\_{pq} та застосовуємо agglomerative clustering (average linkage), отримуючи дендрограму T.

4.2 Переходимо зверху донизу: для кожної внутрішньої вершини S\subseteq L перевіряємо:

* Якщо умова виконується → не розділяємо (класи лишаються у цьому ж листі).
* Якщо умова не виконується → ділимо вершину на її дві дочірні кластери й повторюємо перевірку.

У результаті маємо K рівнів \{L\_1,\dots ,L\_K\} з дедалі дрібнішими класами; алгоритм природно повторює логіку H/V → T/B/O → G/M/Z з рукопису , але без ручного втручання.

## Крок 5. Призначення моделей на кожен рівень ієрархії

Мета: для кожного рівня k вибрати пару детектор D\_k і класифікатор C\_k так, щоб:виконувалися ресурсні обмеження Θ=(\text{FPS}\_{\min},\,T\_{\max});модель залишалася придатною до малого/великого числа класів та розміру об’єктів.

### 5.1 Вхідні параметри (усі легко вимірювані)

| Змінна | Розмірність / Тип | Як отримати | Що означає | | || ---------------- | ------------------ | ---------------------------------------------------------------------------------- | --------------------------------------------------------------------------- | ------------------------------------------------------------------ | ----------------------------------------------------- || ( | L\_k | ) | скаляр (ціле) | Злічити кількість унікальних підкласів, що залишилися на рівні k | Скільки категорій треба розрізняти на поточному рівні || \bar{s}\_k | скаляр (пікселі) | Взяти медіану значень \sqrt{w\_{ij}h\_{ij}} для всіх ROI рівня k | Середній лінійний розмір об’єктів, які обробляє рівень | | || FPS\_{\min} | скаляр (кадри / с) | Задано технічним завданням | Мінімально допустима швидкість системи в робочому режимі | | || T\_{\max} | скаляр (мс) | Задано технічним завданням | Гранична сумарна латентність повного каскаду | | || C\_{\text{GPU}} | скаляр (TFLOPS) | Паспортна потужність доступного GPU (або фактична, виміряна утилітою nvidia-smi) | Обчислювальний ресурс, на який спирається вибір «важкої» чи «легкої» моделі | | |

### 5.2 Дерево рішень (детермінований алгоритм)

pseudocodeif |L\_k| ≤ 2 and \bar{s}\_k ≥ 48:D\_k = YOLOv11-lC\_k = MLP (1×256)elif 3 ≤ |L\_k| ≤ 6 and \bar{s}\_k ≥ 32:D\_k = YOLOv11-mC\_k = FT-Transformer (d=512, h=8, 2 blocks)elif |L\_k| > 6 or \bar{s}\_k < 32:D\_k = RT-DETR-TinyC\_k = FT-Transformer (d=768, h=12, 4 blocks)

# ресурсна деградація

if C\_GPU < 5:D\_k = YOLO-NanoC\_k = MobileNet-MLP

### 5.3 Правила корекції

1. Перевірка латентності: якщо сумарна латентність рівня T\_k > T\_{\max}/K, знижуємо варіант на один рівень у дереві.
2. Малі об’єкти (\bar{s}\_k < 16 px): додаємо попередню SR-підвищувальну мережу (наприклад, ESRGAN ×2) перед D\_k.
3. FPS-тест: якщо фактичний FPS <\text{FPS}\_{\min}, замінюємо C\_k на компактний MLP і повторюємо тест.

### 5.4 Відтворюваність

* Порогові значення (48 px, 32 px, 16 px, TFLOPS = 5) обрано на базі емпіричних меж з розд. 2.5 рукопису; у реплікації їх можна варіювати, але сам алгоритм (порівняння « > / ≤ ») залишається сталим.
* Вхідні змінні |L\_k|, \bar{s}\_k, C\_{\text{GPU}} вимірюються однозначно; отже для тих самих даних і «заліза» вибір моделей повториться.
* Таблиця/псевдокод вище може бути реалізована в два рядки Python (), тому pipeline легко інтегрується у CI.

Вихід: для кожного рівня k повертаємо пару (D\_k,\;C\_k), гарантувавши, що вся ієрархія задовольняє Θ і не потребує ручного тюнінгу.

## Крок 6: Послідовне (ззовні → всередину) навчання

6.1 Навчаємо (D\_1,F\_1,C\_1) на всьому датасеті.

6.2 Для кожного наступного рівня k: використовуємо тільки ті ROI, які попередній рівень відніс до надкласу, та навчаємо (D\_k,F\_k,C\_k).

6.3 Для побудови конкатенованого вектора ознак шукаємо підмножину шарів \mathcal{S} максимізуючи

аналогічно вибору у табл. 1 і рис. 5 рукопису .

## Крок 7: Калібрування порогів та перевірка ресурсних обмежень

7.1 Для кожного C\_k калібруємо поріг довіри p\_k^\star за критерієм Youden J.

7.2 Обчислюємо сумарну латентність і FPS; якщо не виконує Θ — збільшуємо δ\_{\min } (отримаємо менше рівнів) та/або замінюємо моделі згідно (5)–(6).

## Крок 8: Експорт структури

Формуємо JSON / YAML‑файл

yaml:levels:

* id: 1classes: ["H","V"]detector: YOLOv11-lclassifier: FT-Transformerthreshold: 0.82
* id: 2parent\_class: "V"classes: ["T","B","O"]detector: YOLOv11-mclassifier: FT-Transformerthreshold: 0.80
* id: 3parent\_class: "O"classes: ["G","M","Z"]detector: YOLOv11-sclassifier: MLPthreshold: 0.78
* збережені ваги моделей.

Вихідні дані:

* Дерево ієрархії H=(L\_1,\dots ,L\_K) з моделями \{(D\_k,F\_k,C\_k)\}\_{k=1}^{K}.
* Конфігураційний файл для розгортання на БПЛА чи наземній станції.
* Аналітичний звіт (таблиці метрик, граф FPS‑vs‑F1, дендрограма).

## Опис усіх позначень

| Позначення | Розмірність | Значення | | || ---------- | -------------------------- | ------------------------------------------------ | ------ | ---------------------------------------- || N | скаляр | Кількість зображень у датасеті | | || x\_i | H\times W\times3 | i-те зображення RGB | | || y\_i | скаляр з діапазону [1,m] | Базова мітка з множини L | | || m | скаляр | Число базових (найдрібніших) класів | | || f\_{ij} | \mathbb{R}^{d} | Вектор ознак j-го ROI на x\_i | | || d | скаляр | Розмірність простору ознак після B | | || \mu\_l | \mathbb{R}^{d} | Центроїд ознак класу l | | || \Omega | m\times m | Матриця нормованих помилок між парами класів | | || M | m\times m трикутна | Матриця комбінованої близькості (3) | | || K | скаляр | Кінцева кількість ієрархічних рівнів | | || L\_k | множина | Підмножина класів, що розрізняється на рівні k | | || ( | L\_k | ) | скаляр | Поточна кількість підкласів на рівні k || k | індекс 1\ldots K | Номер рівня в ієрархії | | |

## Коротке обґрунтування вигод

| Критерій | Ручна схема (2.4–2.5) | Автоматичний алгоритм || --------------------------- | --------------------- | --------------------- || Вибір рівнів / класів | емпірично | формула (4) || Підбір моделей | фіксований | дерево рішень (5–6) || Пороги довіри | не описано | калібрування 7.1 || Підтримка FPS | декларативно | перевірка 7.2 || Масштабування на нові класи | ручне | повторити кроки 1–4 |

Алгоритм робить побудову структури трасованою, відтворюваною й адаптивною до нових наборів класів чи апаратних обмежень, зберігаючи концепцію, закладену авторами .

## Крок 5. Призначення моделей кожному рівню $k$ (удосконалене дерево рішень)

Нехай

* |L\_k| — кількість підкласів, які треба розрізнити на рівні k;
* \bar{s}\_k — медіанний розмір ROI (у пікселях) між усіма підкласами рівня;
* C\_{\mathrm{GPU}} — доступна обчислювальна потужність (TFLOPS).

| Вхідні умови | Детектор D\_k | Витяг F\_k | Класифікатор C\_k | | || ------------------------------------------ | -------------- | --------------------------------- | ------------------------------------ | ---------------------------- | --------------------------------------------- || ( | L\_k | !\le!2) та \bar{s}\_k\!>\!48 px | YOLOv11-l (≈130 FPS на RTX 3090) | Стандартний CSPDarknet-53 | Linear MLP (1 прих. шар, 256 нейронів) || (3!\le! | L\_k | !\le!6) & \bar{s}\_k\!\ge\!32 px | YOLOv11-m | CSPDarknet-53 + SPP-FPN | FT-Transformer (d = 512, h = 8, 2 блоки) || ( | L\_k | !>!6) або \bar{s}\_k\!<\!32 px | RT-DETR-Tiny (ембединг 256) | CSPDarknet-53 + FPN + CARAFE | FT-Transformer (d = 768, h = 12, 4 блоки) || При дефіциті TFLOPS (C\_{\mathrm{GPU}}<5) | YOLO-Nano | Lightweight FPN | MobileNet-MLP | | |