Repositorio: https://github.com/josuemj/Lake-Cyanobacteria-Sentinel2-Analysis.git

Sebastian huertas 22295

Josue marroquin 22397

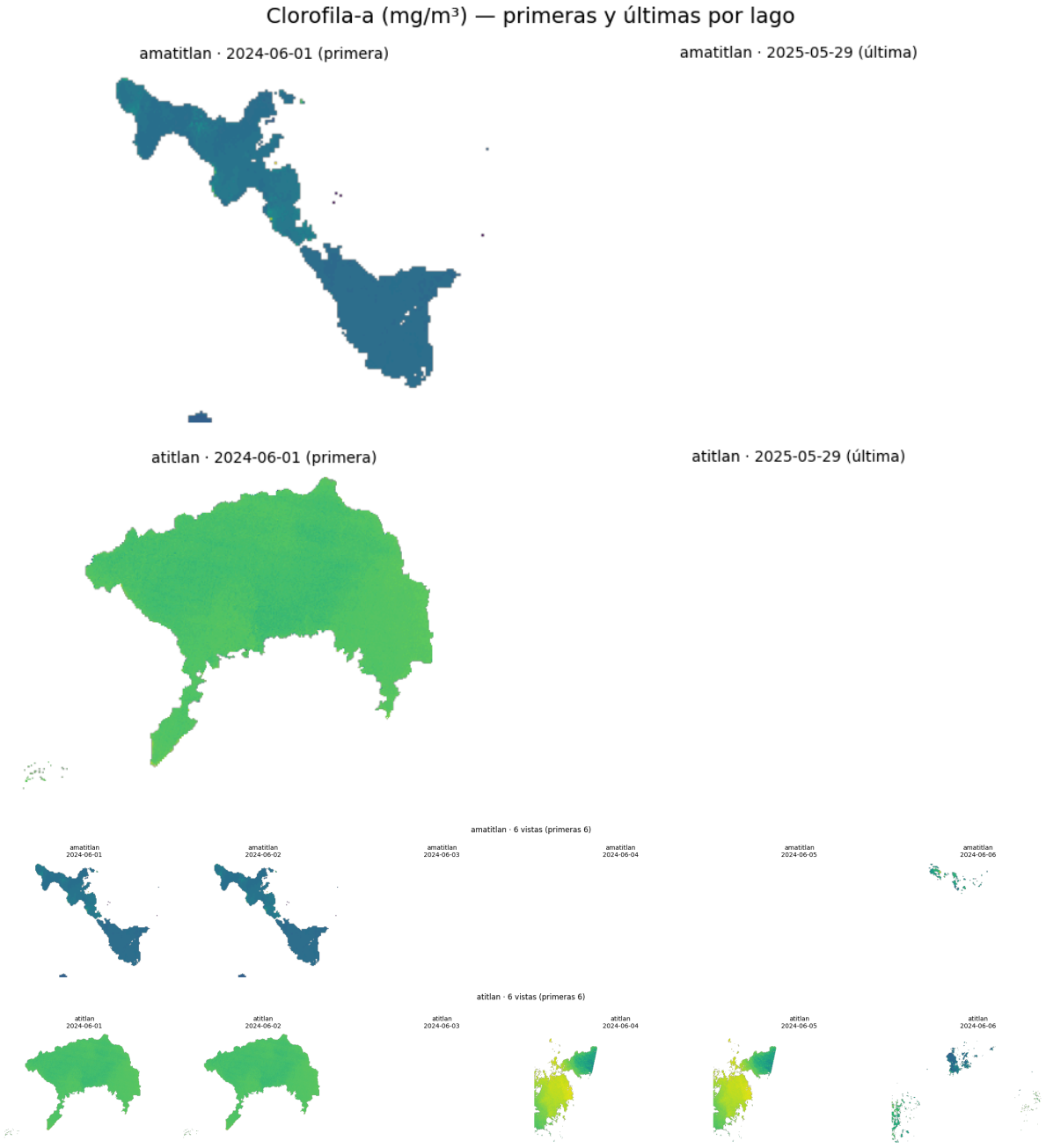
Analísis

In [33]:
```
import os
import re
from glob import glob
from datetime import datetime
import numpy as np
import rasterio
import matplotlib.pyplot as plt
import matplotlib.dates as mdates
from scipy.signal import find_peaks
import folium
from folium.raster_layers import ImageOverlay
from rasterio.plot import reshape_as_image
from typing import Tuple, Optional, Dict, Any, List
import pandas as pd
from rasterio.warp import transform as rio_transform
from sklearn.ensemble import RandomForestRegressor, RandomForestClassifier
from sklearn.metrics import classification_report, confusion_matrix, roc_auc_score

from scipy.ndimage import uniform_filter
from collections import deque
```

In [2]:
```
# Descubrir todos los chl.tif y agrupar por lago ===
base_dir = "out"
pattern = os.path.join(base_dir, "*_*", "chl.tif")  # out/DATE_lake/chl.tif
paths = sorted(glob(pattern))

# Extraer fecha y lago del nombre de carpeta: "YYYY-MM-DD_lake-slug"
rx = re.compile(r"(?P<date>\d{4}-\d{2}-\d{2})_(?P<lake>.+)$")

by_lake = {}
for p in paths:
    parent = os.path.basename(os.path.dirname(p))  # "YYYY-MM-DD_lake"
    m = rx.match(parent)
    if not m:
        continue
    date = datetime.strptime(m.group("date"), "%Y-%m-%d").date()
    lake = m.group("lake")
    by_lake.setdefault(lake, []).append((date, p))

# Ordenar por fecha dentro de cada lago
for lake in by_lake:
    by_lake[lake].sort(key=lambda x: x[0])

# === 2) Reporte de conteos ===
total = 0
```

```
print("Conteo de imágenes por lago:\n")
for lake, items in by_lake.items():
    print(f"- {lake}: {len(items)}")
    total += len(items)
print(f"\nTOTAL imágenes: {total}")

# Si no hay nada, salir
if total == 0:
    raise SystemExit("No se encontraron archivos chl.tif con el patrón esperado.")

# Función para cargar y mostrar una imagen chl.tif ===
def show_chl(path, ax=None, title=None):
    with rasterio.open(path) as src:
        arr = src.read(1)  # banda única
    if ax is None:
        ax = plt.gca()
    im = ax.imshow(arr, cmap="viridis")
    ax.set_axis_off()
    if title:
        ax.set_title(title, fontsize=10)
    return im

#  Visualización rápida: 1ra y última imagen por lago ===
n_lakes = len(by_lake)
fig, axes = plt.subplots(n_lakes, 2, figsize=(10, 4*n_lakes))
if n_lakes == 1:
    axes = np.array([axes])  # normaliza a 2D

for row, (lake, items) in enumerate(sorted(by_lake.items())):
    first_date, first_path = items[0]
    last_date,  last_path  = items[-1]

    im1 = show_chl(first_path, ax=axes[row, 0], title=f"{lake} · {first_date} (prim
    im2 = show_chl(last_path,  ax=axes[row, 1], title=f"{lake} · {last_date} (últim

fig.suptitle("Clorofila-a (mg/m³) — primeras y últimas por lago", fontsize=14)
plt.tight_layout()
plt.show()

#  (Opcional) Panel rápido de hasta 6 fechas por lago ===
# Útil para "ver la película" por encima
max_panels = 6
for lake, items in sorted(by_lake.items()):
    subset = items[:max_panels]  # primeras N (cambia a items[-max_panels:] para úl
    n = len(subset)
    fig, axes = plt.subplots(1, n, figsize=(4*n, 4))
    if n == 1:
        axes = [axes]
    for ax, (d, p) in zip(axes, subset):
        show_chl(p, ax=ax, title=f"{lake}\n{d}")
    fig.suptitle(f"{lake} · {n} vistas (primeras {n})", fontsize=12)
    plt.tight_layout()
    plt.show()
```

```
Conteo de imágenes por lago:

- amatitlan: 363
- atitlan: 363

TOTAL imágenes: 726
```



Analsis temporarl, índice promedio (mensual) de cianobacteria y evolución temporal

In [3]:
```
import os
from collections import defaultdict
import numpy as np
```

```
import rasterio
import matplotlib.pyplot as plt
import pandas as pd

# === Agrupar por mes (YYYY-MM) dentro de cada lago ===
by_lake_month = defaultdict(lambda: defaultdict(list))  # {lake: {YYYY-MM: [(date,
for lake, items in by_lake.items():
    for d, p in items:
        ym = f"{d.year:04d}-{d.month:02d}"
        by_lake_month[lake][ym].append((d, p))

# Función: promedio mensual por píxel (en memoria) ===
def monthly_mean_stack(paths):
    """Carga una lista de chl.tif y devuelve mean_arr (HxW float32) y meta_base (pa
    arrays = []
    meta_base = None
    for path in paths:
        with rasterio.open(path) as src:
            arr = src.read(1).astype(np.float32)
            if meta_base is None:
                meta_base = src.meta.copy()
            arrays.append(arr)
    if not arrays:
        return None, None
    stack = np.stack(arrays, axis=0)            # (N, H, W)
    mean_arr = np.nanmean(stack, axis=0)        # (H, W), ignora NaN
    return mean_arr, meta_base

# monthly_means: {lake: {ym: mean_arr}}
monthly_means = defaultdict(dict)
# También guardamos una media espacial por mes para gráfica temporal
monthly_stats = defaultdict(list)  # {lake: [{"ym":..., "chl_mean":...}, ...]}

for lake, month_dict in sorted(by_lake_month.items()):
    for ym, items in sorted(month_dict.items()):  # items: [(date, path), ...]
        month_paths = [p for _, p in items]
        mean_arr, meta = monthly_mean_stack(month_paths)
        if mean_arr is None:
            continue
        monthly_means[lake][ym] = mean_arr
        # media espacial ignorando NaN
        m = float(np.nanmean(mean_arr)) if np.isfinite(mean_arr).any() else np.nan
        monthly_stats[lake].append({"ym": ym, "chl_mean": m})
        print(f"[{lake}] {ym}: {len(month_paths)} imgs -> promedio mensual en memor

# Graficar progresión mensual por lago (sin guardar a disco) ===
for lake, month_map in sorted(monthly_means.items()):
    if not month_map:
        continue
    # Orden consistente por mes
    months_sorted = sorted(month_map.keys())
    arrays = [monthly_means[lake][ym] for ym in months_sorted]

    # Escala común por lago (mejor comparación): usar percentiles (2–98) para robus
    all_vals = np.concatenate([a[np.isfinite(a)].ravel() for a in arrays if np.isfi
    if all_vals.size > 0:
```

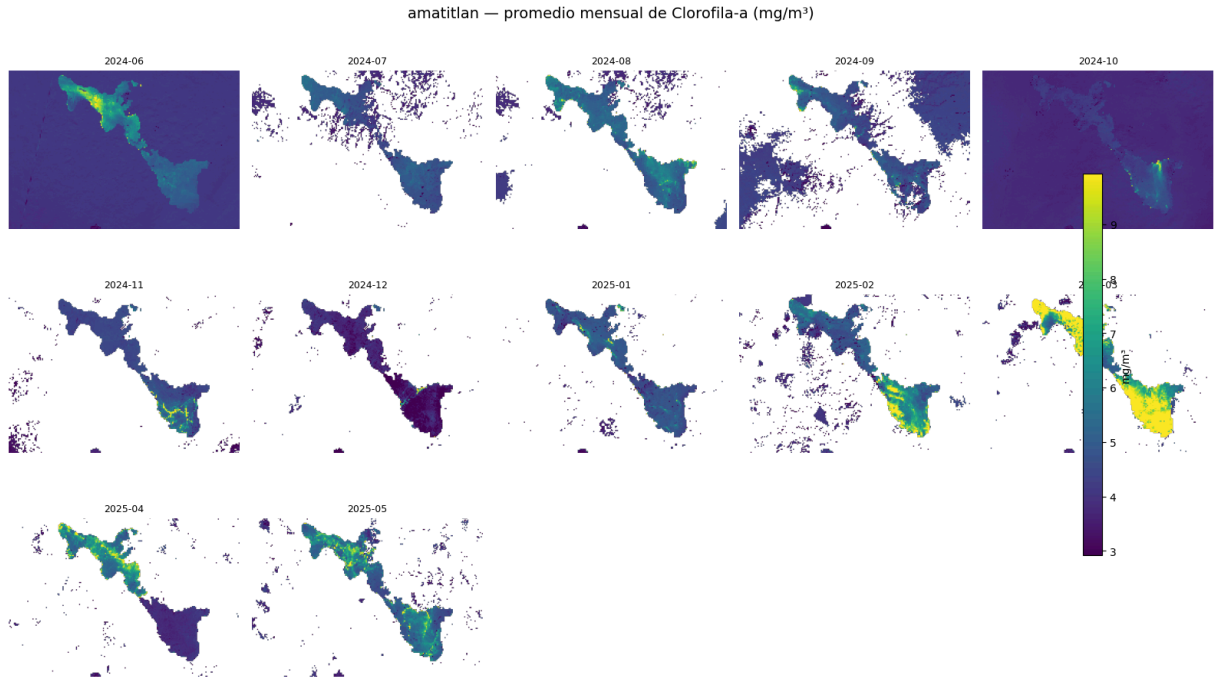
```
        vmin = float(np.percentile(all_vals, 2))
        vmax = float(np.percentile(all_vals, 98))
        if vmin >= vmax:  # fallback si degenera
            vmin, vmax = float(np.nanmin(all_vals)), float(np.nanmax(all_vals))
    else:
        vmin, vmax = None, None

    n = len(months_sorted)
    cols = min(5, n)
    rows = (n + cols - 1) // cols
    fig, axes = plt.subplots(rows, cols, figsize=(cols * 3.2, rows * 3.2))
    axes = np.array(axes).reshape(-1)

    last_im = None
    for ax, ym, arr in zip(axes, months_sorted, arrays):
        im = ax.imshow(arr, cmap="viridis", vmin=vmin, vmax=vmax)
        ax.set_title(ym, fontsize=9)
        ax.axis("off")
        last_im = im

    # Ejes sobrantes off
    for ax in axes[len(arrays):]:
        ax.axis("off")

    fig.suptitle(f"{lake} — promedio mensual de Clorofila-a (mg/m³)", fontsize=14)
    if last_im is not None:
        cbar = fig.colorbar(last_im, ax=axes.tolist(), fraction=0.02, pad=0.02)
        cbar.set_label("mg/m³")
    plt.tight_layout()
    plt.show()

# Serie temporal (media espacial por mes), sin archivos intermedios ===
for lake, recs in sorted(monthly_stats.items()):
    if not recs:
        continue
    df = pd.DataFrame(recs).sort_values("ym")
    plt.figure(figsize=(8, 4))
    plt.plot(df["ym"], df["chl_mean"], marker="o")
    plt.xticks(rotation=45, ha="right")
    plt.ylabel("Clorofila-a media (mg/m³)")
    plt.title(f"{lake} — serie mensual (media espacial)")
    plt.grid(True, alpha=0.3)
    plt.tight_layout()
    plt.show()
```

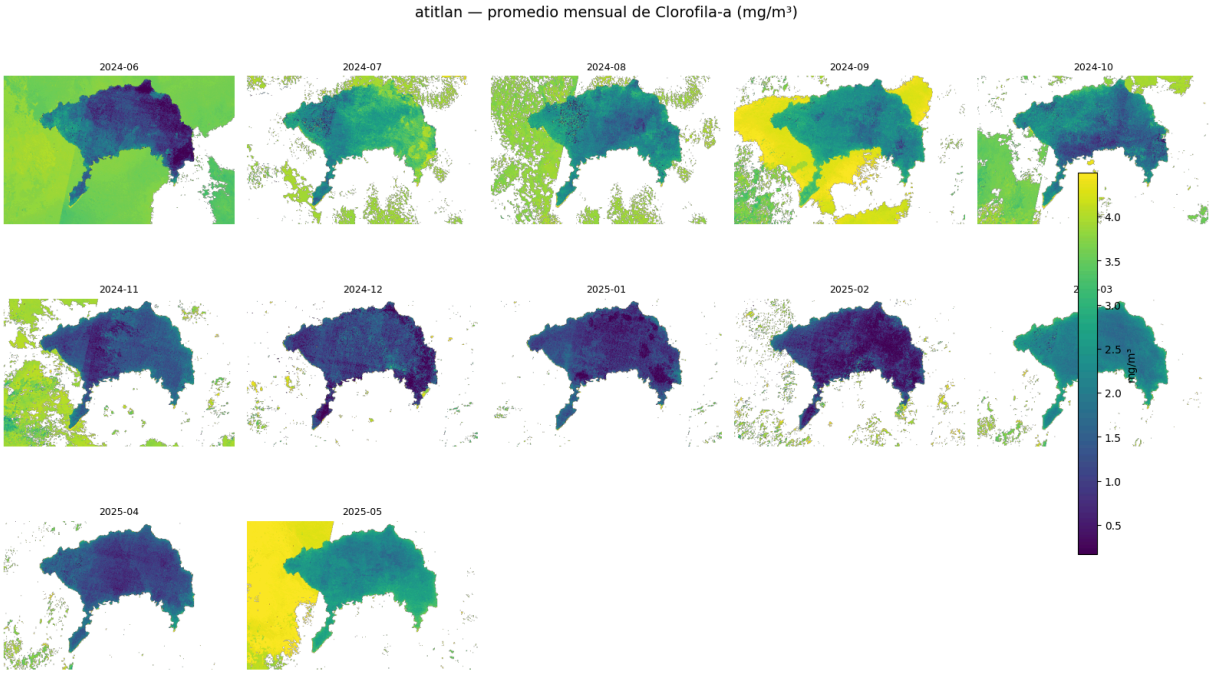
```
[amatitlan] 2024-06: 30 imgs -> promedio mensual en memoria (HxW=(153, 223))
[amatitlan] 2024-07: 31 imgs -> promedio mensual en memoria (HxW=(153, 223))
[amatitlan] 2024-08: 31 imgs -> promedio mensual en memoria (HxW=(153, 223))
```

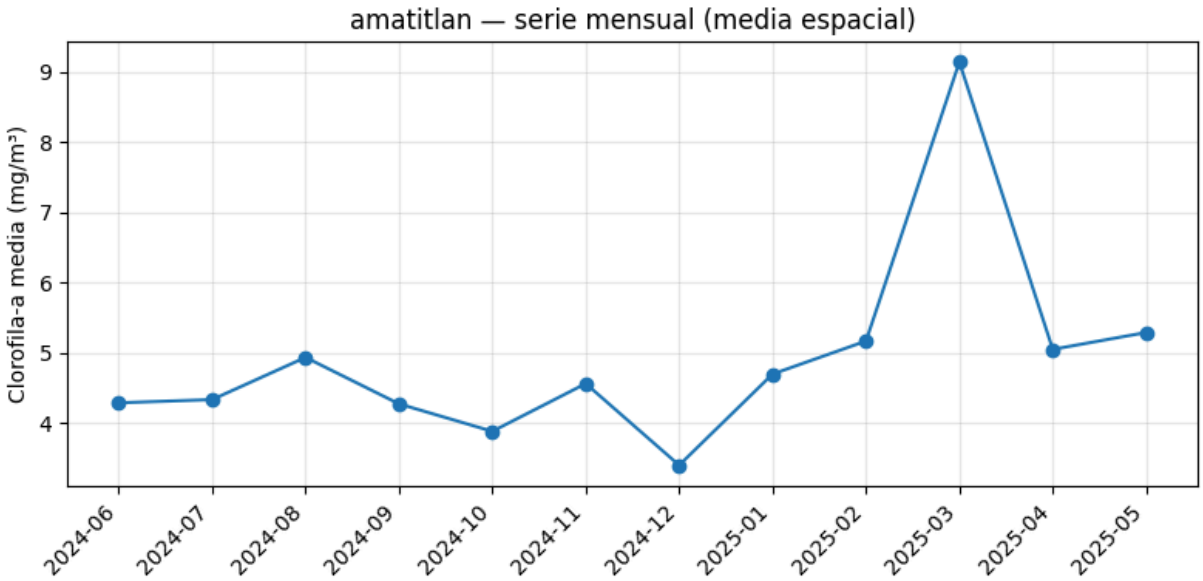
```
C:\Users\thiag\AppData\Local\Temp\ipykernel_28692\3891551696.py:29: RuntimeWarning:
Mean of empty slice
  mean_arr = np.nanmean(stack, axis=0)        # (H, W), ignora NaN
```

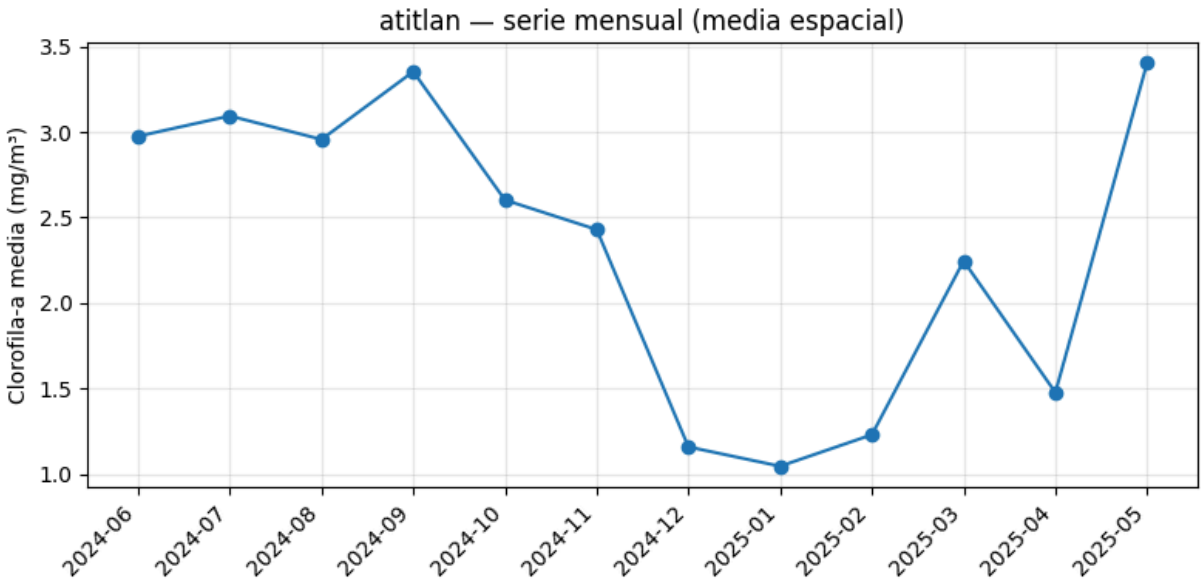
```
[amatitlan] 2024-09: 30 imgs -> promedio mensual en memoria (HxW=(153, 223))
[amatitlan] 2024-10: 31 imgs -> promedio mensual en memoria (HxW=(153, 223))
[amatitlan] 2024-11: 30 imgs -> promedio mensual en memoria (HxW=(153, 223))
[amatitlan] 2024-12: 31 imgs -> promedio mensual en memoria (HxW=(153, 223))
[amatitlan] 2025-01: 31 imgs -> promedio mensual en memoria (HxW=(153, 223))
[amatitlan] 2025-02: 28 imgs -> promedio mensual en memoria (HxW=(153, 223))
[amatitlan] 2025-03: 31 imgs -> promedio mensual en memoria (HxW=(153, 223))
[amatitlan] 2025-04: 30 imgs -> promedio mensual en memoria (HxW=(153, 223))
[amatitlan] 2025-05: 29 imgs -> promedio mensual en memoria (HxW=(153, 223))
[atitlan] 2024-06: 30 imgs -> promedio mensual en memoria (HxW=(292, 455))
[atitlan] 2024-07: 31 imgs -> promedio mensual en memoria (HxW=(292, 455))
[atitlan] 2024-08: 31 imgs -> promedio mensual en memoria (HxW=(292, 455))
[atitlan] 2024-09: 30 imgs -> promedio mensual en memoria (HxW=(292, 455))
[atitlan] 2024-10: 31 imgs -> promedio mensual en memoria (HxW=(292, 455))
[atitlan] 2024-11: 30 imgs -> promedio mensual en memoria (HxW=(292, 455))
[atitlan] 2024-12: 31 imgs -> promedio mensual en memoria (HxW=(292, 455))
[atitlan] 2025-01: 31 imgs -> promedio mensual en memoria (HxW=(292, 455))
[atitlan] 2025-02: 28 imgs -> promedio mensual en memoria (HxW=(292, 455))
[atitlan] 2025-03: 31 imgs -> promedio mensual en memoria (HxW=(292, 455))
[atitlan] 2025-04: 30 imgs -> promedio mensual en memoria (HxW=(292, 455))
[atitlan] 2025-05: 29 imgs -> promedio mensual en memoria (HxW=(292, 455))
```

```
C:\Users\thiag\AppData\Local\Temp\ipykernel_28692\3891551696.py:88: UserWarning: Thi
s figure includes Axes that are not compatible with tight_layout, so results might b
e incorrect.
  plt.tight_layout()
```

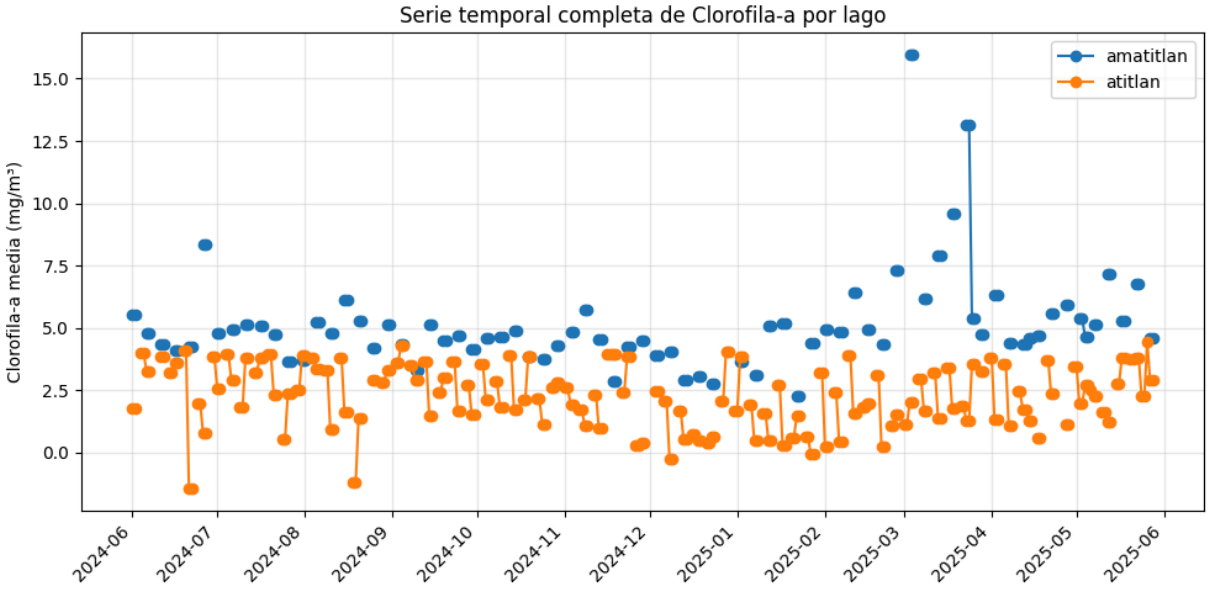



atitlan — promedio mensual de Clorofila-a (mg/m³)
2024-06
2024-07
2024-08
2024-09
2024-10
2024-11
2024-12
2025-01
2025-02
2025-04
2025-05
mg/m³
4.0
3.5
3.0
2.5
2.0
1.5
1.0
0.5


amatitlan — serie mensual (media espacial)
Clorofila-a media (mg/m³)
9
8
7
6
5
4
2024-06
2024-07
2024-08
2024-09
2024-10
2024-11
2024-12
2025-01
2025-02
2025-03
2025-04
2025-05



In [4]:
```
# Serie temporal completa: todas las fechas por lago ===
plt.figure(figsize=(10, 5))
for lake, items in sorted(by_lake.items()):
    # Para cada fecha, calcular la media espacial de la imagen
    dates = []
    means = []
    for d, p in items:
        with rasterio.open(p) as src:
            arr = src.read(1).astype(np.float32)
            m = float(np.nanmean(arr)) if np.isfinite(arr).any() else np.nan
            dates.append(d)
            means.append(m)
    plt.plot(dates, means, marker='o', label=lake)

plt.gca().xaxis.set_major_locator(mdates.MonthLocator())
plt.gca().xaxis.set_major_formatter(mdates.DateFormatter('%Y-%m'))
plt.xticks(rotation=45, ha='right')
plt.ylabel('Clorofila-a media (mg/m³)')
plt.title('Serie temporal completa de Clorofila-a por lago')
plt.grid(True, alpha=0.3)
plt.legend()
plt.tight_layout()
plt.show()
```

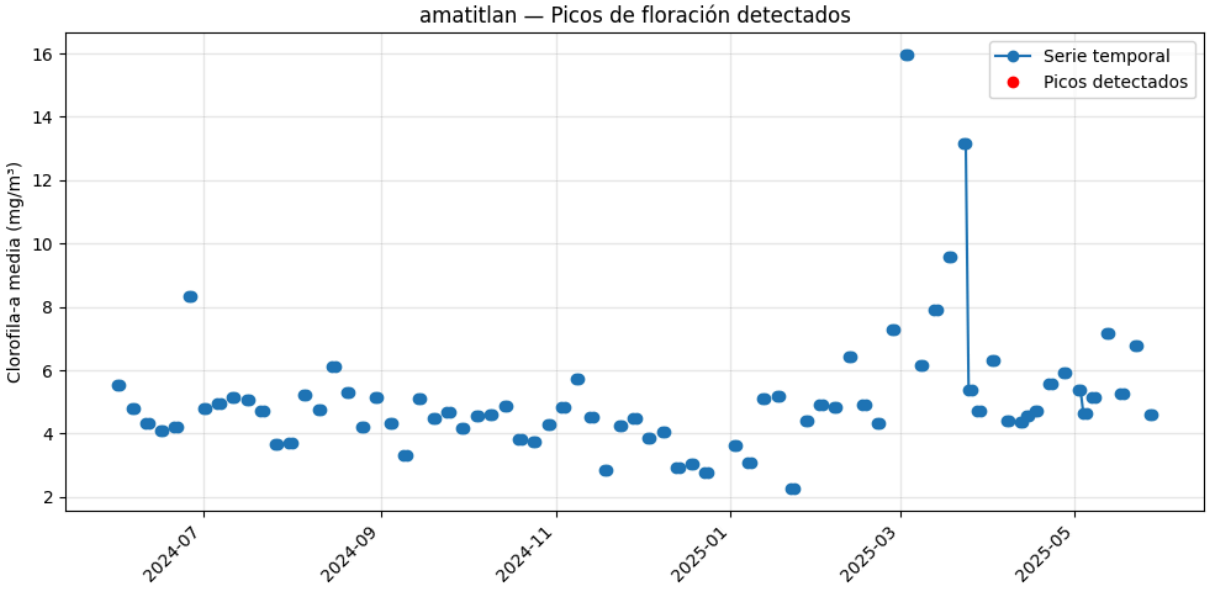


Picos de flora

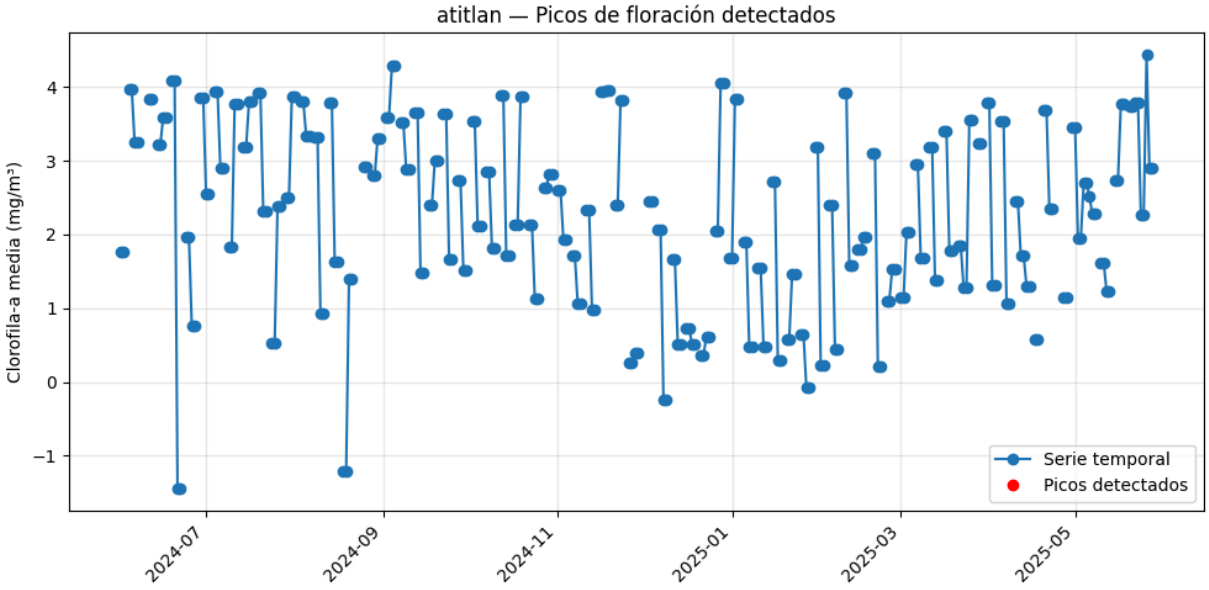
In [5]:
```
# Parámetros para detección de picos (ajustables según el rango de datos)
peak_prominence = 5  # mg/m³, ajusta según tu dataset
peak_distance = 2    # mínimo separación en número de observaciones


for lake, items in sorted(by_lake.items()):
    dates = []
    means = []
    for d, p in items:
        with rasterio.open(p) as src:
            arr = src.read(1).astype(np.float32)
            m = float(np.nanmean(arr)) if np.isfinite(arr).any() else np.nan
            dates.append(d)
            means.append(m)
    means_arr = np.array(means)
    # Detección de picos
    peaks, props = find_peaks(means_arr, prominence=peak_prominence, distance=peak_
    print(f'\nLago: {lake}')
    if len(peaks) == 0:
        print('No se detectaron picos de floración destacados.')
    else:
        print(f'Se detectaron {len(peaks)} picos de floración:')
        for idx in peaks:
            print(f'  - Fecha crítica: {dates[idx]} | Clorofila-a media: {means_arr
    # Opcional: graficar con picos resaltados
    plt.figure(figsize=(10, 5))
    plt.plot(dates, means_arr, marker='o', label='Serie temporal')
    plt.plot([dates[i] for i in peaks], means_arr[peaks], 'ro', label='Picos detect
    plt.xticks(rotation=45, ha='right')
    plt.ylabel('Clorofila-a media (mg/m³)')
    plt.title(f'{lake} — Picos de floración detectados')
    plt.grid(True, alpha=0.3)
```

```
    plt.legend()
    plt.tight_layout()
    plt.show()
```

```
Lago: amatitlan
No se detectaron picos de floración destacados.
```




```
Lago: atitlan
No se detectaron picos de floración destacados.
```



Aunque el algoritmo no haya marcado “picos” formales según el umbral configurado, los gráficos muestran patrones que permiten sacar conclusiones cualitativas:

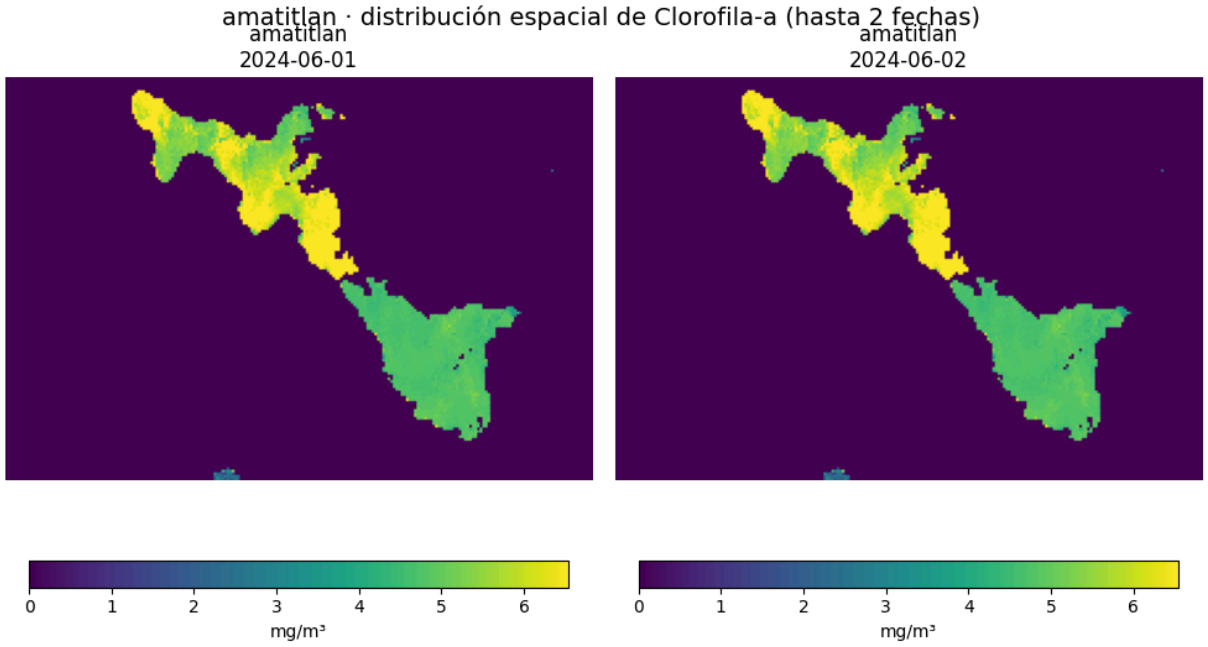
Lago Amatitlán: la mayor parte del tiempo la clorofila-a media se mantiene estable entre ~3 y 6 mg/m³, pero hay al menos un evento muy sobresaliente a inicios de marzo de 2025 que supera los 16 mg/m³, seguido de una caída rápida. Esto sugiere un episodio aislado de floración intensa que quizá no fue detectado por el criterio de picos debido a la configuración de sensibilidad.

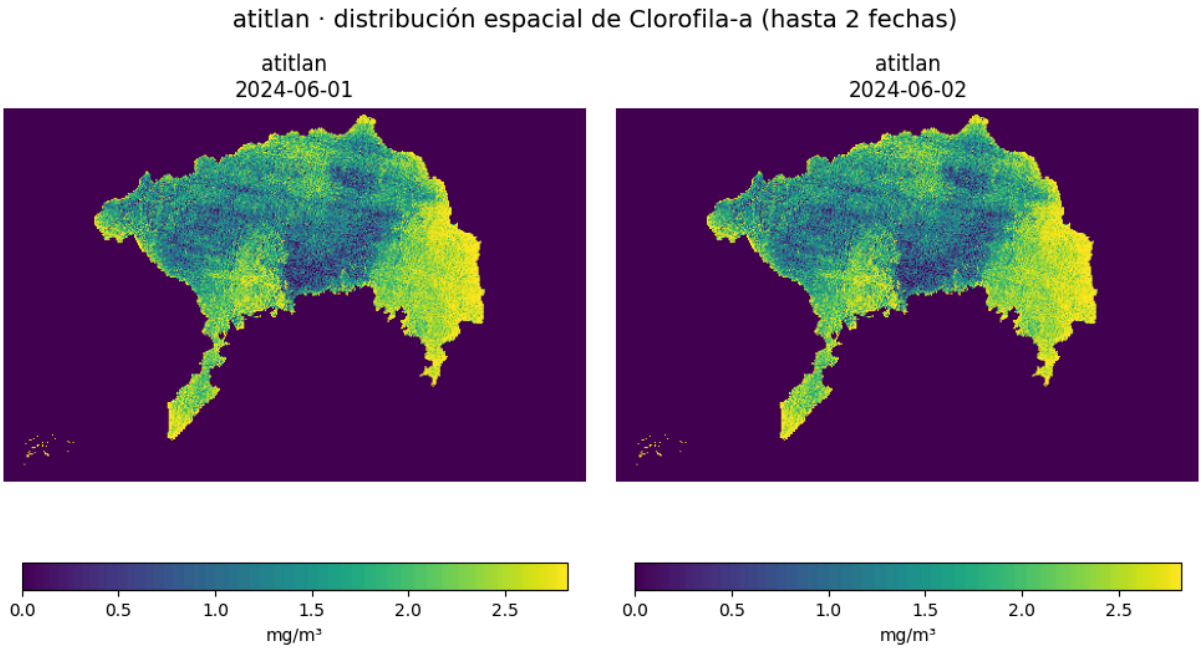
Lago Atitlán: las concentraciones son menores (~0 a 4 mg/m³) y más variables día a día, con descensos abruptos y recuperaciones rápidas. No se observan valores tan altos como en Amatitlán, pero sí una fluctuación constante que podría enmascarar picos moderados. También se nota una ligera tendencia a valores algo más bajos en el período noviembre 2024 – febrero 2025, seguida de una recuperación.

Analisis espacial

Mapea la distribución de cianobacteria dentro de cada lago. mapas comparativos entre diferentes fechas para cada lago

In [6]:
```
max_compare = 2  # hasta 4 fechas por lago
for lake, items in sorted(by_lake.items()):
    subset = items[:max_compare]
    n = len(subset)
    rows = 2
    cols = 2
    fig, axes = plt.subplots(rows, cols, figsize=(10, 10))
    axes = axes.flatten()
    for i, (d, p) in enumerate(subset):
        ax = axes[i]
        with rasterio.open(p) as src:
            arr = src.read(1)
            arr = np.nan_to_num(arr, nan=0)
            vmin = np.nanpercentile(arr, 2)
            vmax = np.nanpercentile(arr, 98)
            im = ax.imshow(arr, cmap='viridis', vmin=vmin, vmax=vmax)
            ax.set_title(f'{lake}\n{d}')
            ax.axis('off')
            cbar = fig.colorbar(im, ax=ax, orientation='horizontal', fraction=0.05,
            cbar.set_label('mg/m³')
    # Ejes sobrantes off
    for j in range(n, rows*cols):
        axes[j].axis('off')
    fig.suptitle(f'{lake} · distribución espacial de Clorofila-a (hasta 2 fechas)',
    plt.tight_layout()
    plt.show()
```


amatitlan · distribución espacial de Clorofila-a (hasta 2 fechas)
amatitlan
2024-06-01
amatitlan
2024-06-02
0
1
2
3
4
5
6
mg/m³
0
1
2
3
4
5
6
mg/m³



9, 10 Análisis de los lagos y comparación entre ellos:

9.1. Análisis por lago

Lago Amatitlán

Los valores de clorofila-a (CHL) se mantienen mayormente entre 3 y 6 mg/m³ durante gran parte del período.

Se observa un aumento notable hacia marzo de 2025, con un pico cercano a 16 mg/m³, lo que indica un episodio fuerte de proliferación, aunque no se marcó como “pico detectado” bajo el criterio automático.

La distribución espacial muestra que las concentraciones más altas se dan en sectores cercanos a desembocaduras y zonas someras, probablemente por aporte de nutrientes.

Lago Atitlán

Valores mucho más bajos en general, en un rango entre 0.5 y 4 mg/m³, con alta variabilidad espacial pero sin incrementos abruptos sostenidos.

La serie temporal muestra fluctuaciones rápidas y frecuentes, pero sin un episodio dominante que sobresalga.

Las imágenes espaciales reflejan distribución relativamente homogénea, con concentraciones más altas en áreas cercanas a poblaciones y desembocaduras de ríos.

9.2. Comparación de intensidad y frecuencia

Intensidad: Amatitlán presenta un episodio de clorofila-a mucho más intenso (pico ~16 mg/m³) que cualquier evento en Atitlán.

Frecuencia: Atitlán tiene más fluctuaciones pequeñas y frecuentes, mientras que Amatitlán muestra una tendencia más estable interrumpida por un pico muy marcado.

Esto sugiere que Amatitlán podría experimentar floraciones esporádicas pero muy intensas, mientras que Atitlán presenta una dinámica más continua pero de menor magnitud.

9.3. Posibles causas de las diferencias

Geografía e hidrología: Amatitlán es un lago más pequeño y menos profundo que Atitlán, lo que facilita el calentamiento y la acumulación de nutrientes en la columna de agua.

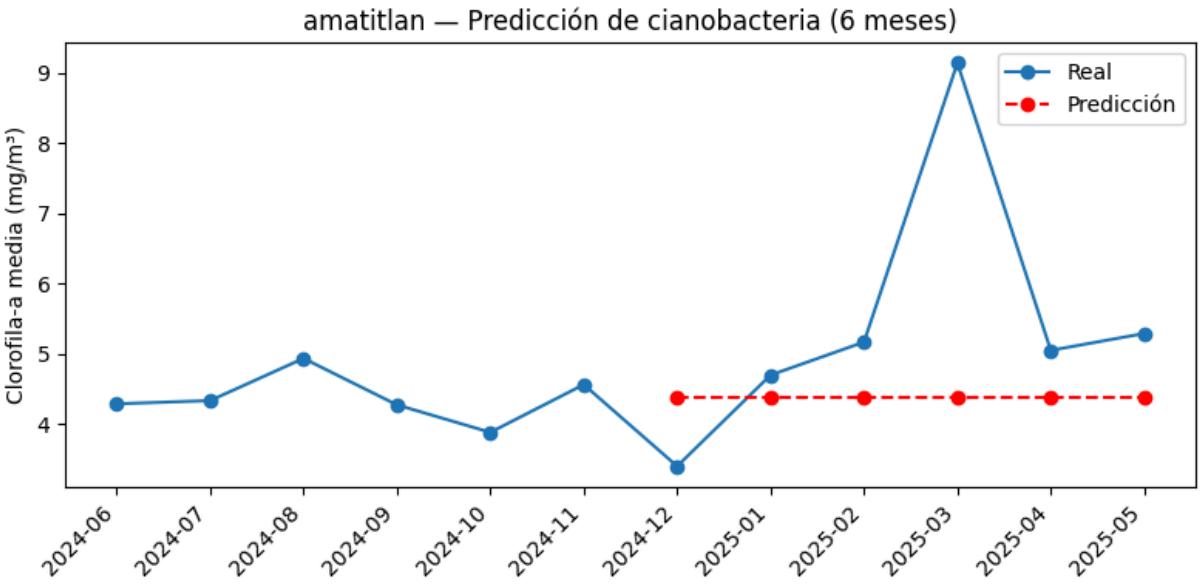
Uso del suelo y presión urbana: Amatitlán está rodeado por áreas urbanas densas (Ciudad de Guatemala y Villa Nueva), con descargas de aguas residuales y escorrentía urbana, lo que eleva las cargas de fósforo y nitrógeno. Atitlán.

Temperatura y clima: Amatitlán probablemente alcanza temperaturas más altas en superficie, favoreciendo el crecimiento de cianobacterias. Atitlán, al estar a mayor altitud y con mayor volumen de agua, mantiene temperaturas más moderadas y mezclas más estables.

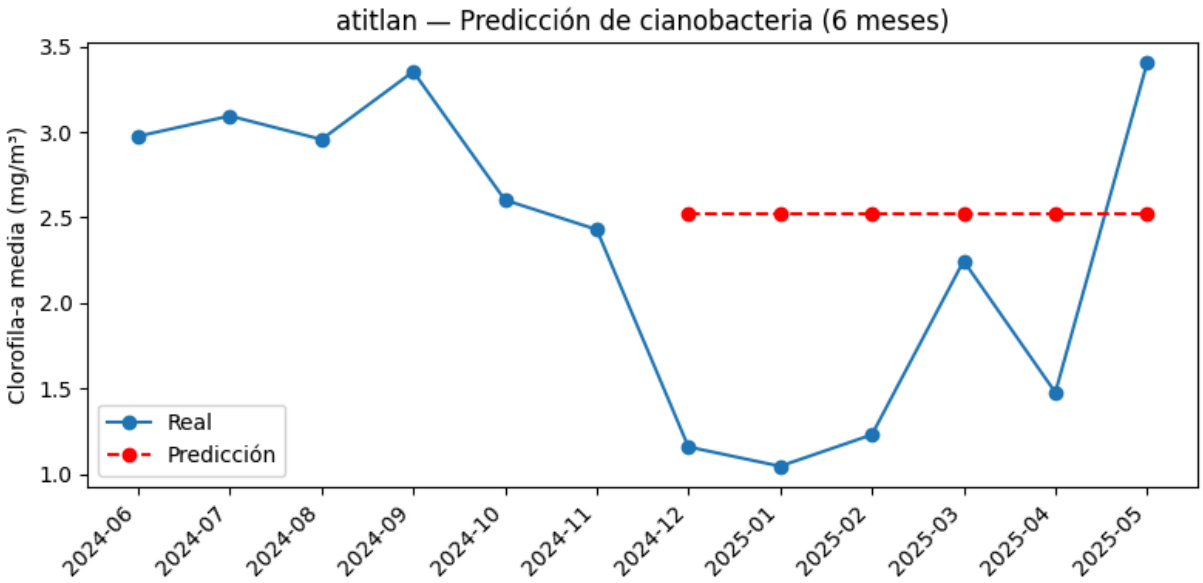
10. Serie temporal

In [7]:
```
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error
import numpy as np

horizon = 6  # meses a predecir
```

```
for lake, recs in sorted(monthly_stats.items()):
    if len(recs) < horizon + 2:
        print(f"No hay suficientes datos para predicción en {lake}.")
        continue
    df = pd.DataFrame(recs).sort_values("ym")
    # Convertir fechas a números (ordinales)
    X = np.arange(len(df)).reshape(-1, 1)
    y = df["chl_mean"].values
    # Entrenar modelo en todos menos los últimos 'horizon' meses
    X_train, y_train = X[:-horizon], y[:-horizon]
    X_test = X[-horizon:]
    # Modelo
    model = RandomForestRegressor(n_estimators=100, random_state=42)
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
    # Mostrar resultados
    plt.figure(figsize=(8, 4))
    plt.plot(df["ym"], y, marker="o", label="Real")
    plt.plot(df["ym"].iloc[-horizon:], y_pred, "ro--", label="Predicción")
    plt.xticks(rotation=45, ha="right")
    plt.ylabel("Clorofila-a media (mg/m³)")
    plt.title(f"{lake} — Predicción de cianobacteria ({horizon} meses)")
    plt.legend()
    plt.tight_layout()
    plt.show()
    print(f"Predicción para {lake} ({horizon} meses): {y_pred}")
    print(f"Error cuadrático medio: {mean_squared_error(y[-horizon:], y_pred):.2f}"
```




```
Predicción para amatitlan (6 meses): [4.37481648 4.37481648 4.37481648 4.37481648 4.
37481648 4.37481648]
Error cuadrático medio: 4.27
```




```
Predicción para atitlan (6 meses): [2.52354513 2.52354513 2.52354513 2.52354513 2.52
354513 2.52354513]
Error cuadrático medio: 1.28
```

11. Modelo Cianobacteria

In [13]:

```python
def _nearest_item_for_date(items: List[Tuple], target_date) -> Tuple:
    """items: [(date, path), ...] ordenados. Devuelve (date, path) con fecha más ce
    if isinstance(target_date, str):
        target_date = pd.to_datetime(target_date).date()
    diffs = [abs((d - target_date).days) for d, _ in items]
    i = int(np.argmin(diffs))
    return items[i]

def _read_chl(path: str) -> Tuple[np.ndarray, Dict[str, Any]]:
    with rasterio.open(path) as src:
        arr = src.read(1).astype(np.float32)
        meta = {
            "crs": src.crs,
            "transform": src.transform,
            "nodata": src.nodata,
            "bounds": src.bounds,
            "shape": (src.height, src.width),
        }
    return arr, meta

def _valid_pixels(arr: np.ndarray, nodata: Optional[float]) -> np.ndarray:
    v = np.isfinite(arr)
    if nodata is not None:
        v &= arr != nodata
    return arr[v]

def _otsu_threshold(values: np.ndarray, bins: int = 256, clip_pct: Tuple[float,floa
    """Umbral Otsu robusto (recorta percentiles extremos)."""
    if values.size == 0:
        return np.nan
```

```
    lo, hi = np.percentile(values, clip_pct)
    vals = np.clip(values, lo, hi)
    hist, bin_edges = np.histogram(vals, bins=bins, range=(lo, hi))
    hist = hist.astype(float)
    prob = hist / hist.sum() if hist.sum() > 0 else hist
    cum_prob = np.cumsum(prob)
    cum_mean = np.cumsum(prob * (bin_edges[:-1] + bin_edges[1:]) / 2.0)
    total_mean = cum_mean[-1] if cum_mean.size else 0.0
    var_between = (total_mean * cum_prob - cum_mean)**2 / (cum_prob * (1 - cum_prob
    k = int(np.nanargmax(var_between))
    # umbral es el borde entre bin k y k+1 (promedio)
    thr = (bin_edges[k] + bin_edges[k+1]) / 2.0
    return float(thr)

def _classify_value(value: float, threshold: float, mode: str = "greater") -> bool:
    """Etiqueta binaria: 1 si hay cianobacteria. Por defecto: value >= threshold."""
    if not np.isfinite(value) or not np.isfinite(threshold):
        return False
    return bool(value >= threshold) if mode == "greater" else bool(value <= thresho

# --- API principal ---
def predict_cyano_point_lonlat(lake: str, when, lon: float, lat: float,
                               umbral_mg_m3: Optional[float] = None,
                               method: str = "otsu",
                               bins: int = 256) -> Dict[str, Any]:
    """
    Clasifica un punto (lon, lat WGS84) en la imagen del lago más cercana a 'when'.
    Si umbral_mg_m3 es None, se calcula con Otsu sobre la imagen.
    """
    if lake not in by_lake:
        raise ValueError(f"Lago '{lake}' no encontrado en 'by_lake'.")

    date_sel, path = _nearest_item_for_date(by_lake[lake], when)
    arr, meta = _read_chl(path)

    # reproyección lon/lat -> CRS del raster
    if meta["crs"] is not None and str(meta["crs"]) != "EPSG:4326":
        xs, ys = rio_transform("EPSG:4326", meta["crs"], [lon], [lat])
        x, y = xs[0], ys[0]
    else:
        x, y = lon, lat

    with rasterio.open(path) as src:
        try:
            row, col = src.index(x, y)
        except Exception:
            return {
                "lake": lake, "date": str(date_sel), "lon": lon, "lat": lat,
                "in_bounds": False, "value_mg_m3": np.nan, "threshold_mg_m3": np.na
            }

    # fuera de imagen
    H, W = meta["shape"]
    if not (0 <= row < H and 0 <= col < W):
        return {
            "lake": lake, "date": str(date_sel), "lon": lon, "lat": lat,
```

```
            "in_bounds": False, "value_mg_m3": np.nan, "threshold_mg_m3": np.nan, "
        }

    value = float(arr[row, col])

    # umbral
    if umbral_mg_m3 is None:
        if method.lower() == "otsu":
            vals = _valid_pixels(arr, meta["nodata"])
            thr = _otsu_threshold(vals, bins=bins)
        elif method.lower() == "p90":
            thr = float(np.percentile(_valid_pixels(arr, meta["nodata"]), 90))
        elif method.lower() == "mean+std":
            v = _valid_pixels(arr, meta["nodata"])
            thr = float(np.nanmean(v) + np.nanstd(v))
        else:
            raise ValueError("Método desconocido. Use: 'otsu' | 'p90' | 'mean+std'
    else:
        thr = float(umbral_mg_m3)

    label = _classify_value(value, thr, mode="greater")

    return {
        "lake": lake,
        "date": str(date_sel),
        "lon": lon,
        "lat": lat,
        "row": int(row),
        "col": int(col),
        "in_bounds": True,
        "value_mg_m3": value,
        "threshold_mg_m3": float(thr),
        "method": method if umbral_mg_m3 is None else "fixed",
        "has_cyano": label,
        "path": path,
    }

def predict_cyano_point_rc(lake: str, when, row: int, col: int,
                           umbral_mg_m3: Optional[float] = None,
                           method: str = "otsu",
                           bins: int = 256) -> Dict[str, Any]:
    """Clasifica un punto por índice de píxel (row, col)."""
    if lake not in by_lake:
        raise ValueError(f"Lago '{lake}' no encontrado en 'by_lake'.")

    date_sel, path = _nearest_item_for_date(by_lake[lake], when)
    arr, meta = _read_chl(path)
    H, W = meta["shape"]
    if not (0 <= row < H and 0 <= col < W):
        return {
            "lake": lake, "date": str(date_sel), "row": int(row), "col": int(col),
            "in_bounds": False, "value_mg_m3": np.nan, "threshold_mg_m3": np.nan, "
        }

    value = float(arr[row, col])
```

```
    if umbral_mg_m3 is None:
        if method.lower() == "otsu":
            thr = _otsu_threshold(_valid_pixels(arr, meta["nodata"]), bins=bins)
        elif method.lower() == "p90":
            thr = float(np.percentile(_valid_pixels(arr, meta["nodata"]), 90))
        elif method.lower() == "mean+std":
            v = _valid_pixels(arr, meta["nodata"])
            thr = float(np.nanmean(v) + np.nanstd(v))
        else:
            raise ValueError("Método desconocido. Use: 'otsu' | 'p90' | 'mean+std'
    else:
        thr = float(umbral_mg_m3)

    label = _classify_value(value, thr, mode="greater")

    return {
        "lake": lake,
        "date": str(date_sel),
        "row": int(row),
        "col": int(col),
        "in_bounds": True,
        "value_mg_m3": value,
        "threshold_mg_m3": float(thr),
        "method": method if umbral_mg_m3 is None else "fixed",
        "has_cyano": label,
        "path": path,
    }
```

In [14]:
```
res = predict_cyano_point_lonlat(lake="amatitlan", when="2024-04-15", lon=-90.62, l
print(res)
```

```
{'lake': 'amatitlan', 'date': '2024-06-01', 'lon': -90.62, 'lat': 14.46, 'row': 63,
'col': 32, 'in_bounds': True, 'value_mg_m3': nan, 'threshold_mg_m3': 6.1013991704210
75, 'method': 'otsu', 'has_cyano': False, 'path': 'out\\2024-06-01_amatitlan\\chl.ti
f'}
```

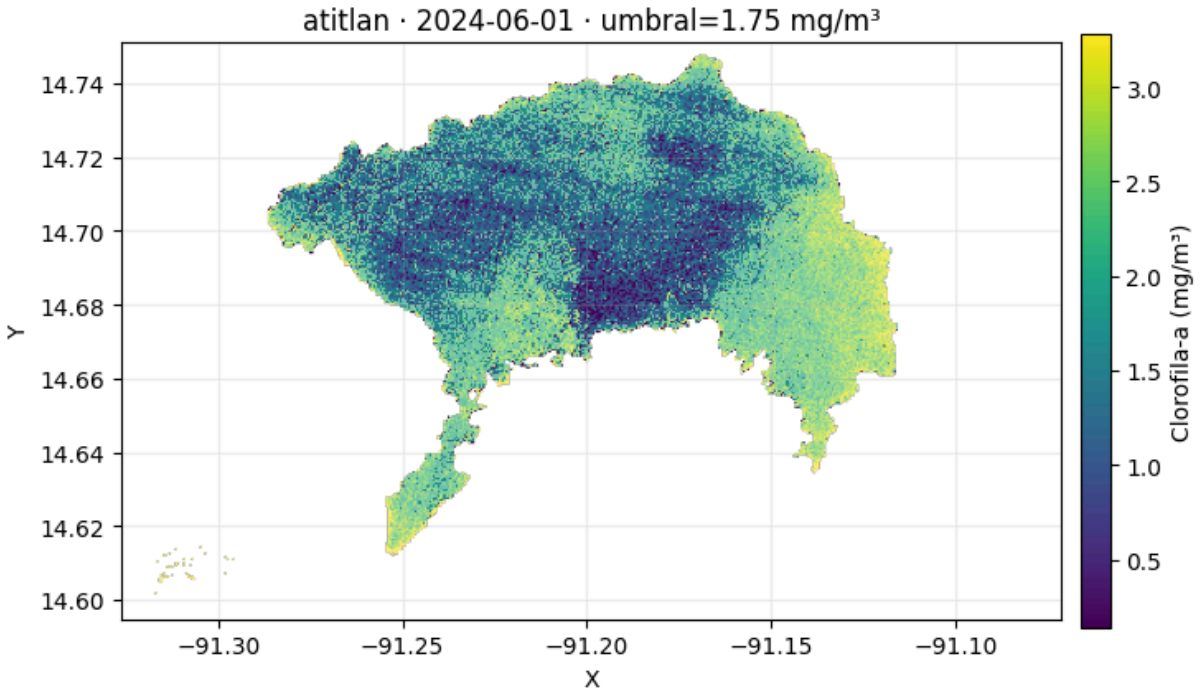
In [23]:
```
res = predict_cyano_point_lonlat(lake="atitlan", when="2024-04-15", lon=-90.62, lat
print(res)
```

```
{'lake': 'atitlan', 'date': '2024-06-01', 'lon': -90.62, 'lat': 14.46, 'in_bounds':
False, 'value_mg_m3': nan, 'threshold_mg_m3': nan, 'has_cyano': False}
```

In [24]:
```
def cyano_mask_static(lake: str, when, umbral_mg_m3=None, method: str = "otsu", bin
    date_sel, path = _nearest_item_for_date(by_lake[lake], when)
    arr, meta = _read_chl(path)
    vals = _valid_pixels(arr, meta["nodata"])

    # umbral
    if umbral_mg_m3 is None:
        if method.lower() == "otsu":
            thr = _otsu_threshold(vals, bins=bins)
        elif method.lower() == "p90":
            thr = float(np.percentile(vals, 90))
        elif method.lower() == "mean+std":
            thr = float(np.nanmean(vals) + np.nanstd(vals))
        else:
            raise ValueError("Método desconocido: 'otsu' | 'p90' | 'mean+std' o fij
```

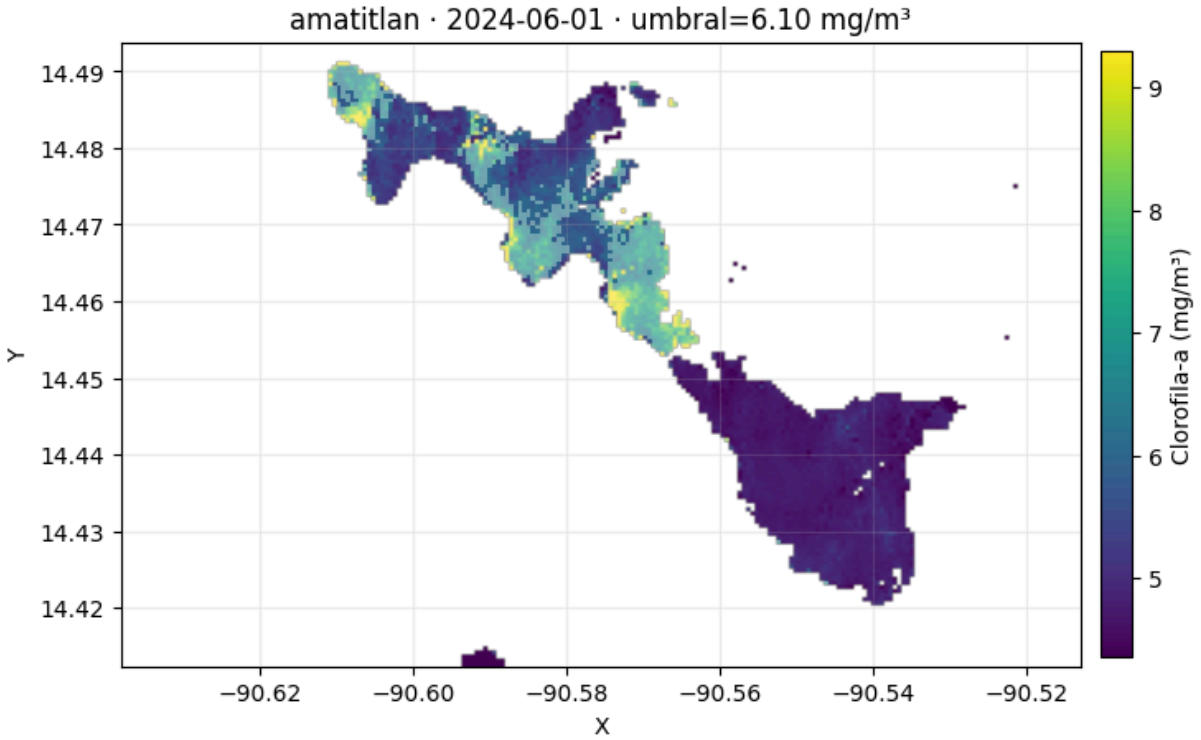
```
    else:
        thr = float(umbral_mg_m3)

    # base raster (clorofila) con percentiles robustos
    arr_plot = np.nan_to_num(arr, nan=np.nan)
    finite = np.isfinite(arr_plot)
    if finite.any():
        vmin = float(np.nanpercentile(arr_plot[finite], 2))
        vmax = float(np.nanpercentile(arr_plot[finite], 98))
        if vmin >= vmax:
            vmin, vmax = float(np.nanmin(arr_plot[finite])), float(np.nanmax(arr_pl
    else:
        vmin, vmax = None, None

    # máscara binaria
    mask = (arr >= thr).astype(np.uint8)

    # extents geográficos del raster
    minx, miny, maxx, maxy = meta["bounds"]

    # pinta
    fig, ax = plt.subplots(figsize=(7.5, 6.5))
    im = ax.imshow(arr, extent=(minx, maxx, miny, maxy), origin="upper",
                   cmap="viridis", vmin=vmin, vmax=vmax)
    # overlay máscara (verde translúcido)
    ax.imshow(np.where(mask==1, 1, np.nan), extent=(minx, maxx, miny, maxy), origin
              cmap="Greens", alpha=0.35, interpolation="nearest")

    cbar = fig.colorbar(im, ax=ax, fraction=0.03, pad=0.02)
    cbar.set_label("Clorofila-a (mg/m³)")
    ax.set_title(f"{lake} · {date_sel} · umbral={thr:.2f} mg/m³")
    ax.set_xlabel("X"); ax.set_ylabel("Y")
    ax.grid(True, alpha=0.2)
    plt.tight_layout()
    plt.show()
    return {"lake": lake, "date": str(date_sel), "threshold_mg_m3": thr}
```

In [25]:
```
cyano_mask_static("atitlan", "2024-04-15", method="otsu")
```



Out[25]:
```
{'lake': 'atitlan',
 'date': '2024-06-01',
 'threshold_mg_m3': 1.7476880316721515}
```

In [26]:
```
cyano_mask_static("amatitlan", "2024-04-15", method="otsu")
```



Out[26]:
```
{'lake': 'amatitlan',
 'date': '2024-06-01',
 'threshold_mg_m3': 6.101399170421075}
```

- Amatitlán (2024-06-01)

Umbral (Otsu): 6.10 mg/m³.

El punto elegido cae dentro del raster pero su píxel es NaN ⇒ probablemente zona enmascarada (nubes/artefacto/no-data) o fuera del cuerpo de agua en esa grilla.

Con ese umbral, solo parches con clorofila alta serían clasificados como “con cianobacteria”; el resto queda por debajo.

- Atitlán (2024-06-01)

Umbral (Otsu): 1.75 mg/m³.

El punto está fuera del raster (in_bounds=False). Se ve en el gráfico que la extensión de X ~ [-91.30, -91.10] y Y ~ [14.60, 14.75]; tu punto (-90.62, 14.46) está fuera de ese rango.

12. Modelo cianobacteria series temporales

In [69]:
```
def _safe_otsu_from_array(arr: np.ndarray, bins: int = 256) -> float:
    vals = _valid_pixels(arr, nodata=None)
    if vals.size == 0: return np.nan
    if np.nanstd(vals) < 1e-8: return float(np.nanmean(vals))
    return float(_otsu_threshold(vals, bins=bins))

def month_feats(dt: pd.Timestamp):
    ang = 2*np.pi*(dt.month/12.0)
    return np.sin(ang), np.cos(ang)

def local_mean(arr: np.ndarray, k: int = 3) -> np.ndarray:
    a = np.nan_to_num(arr, nan=0.0)
    m = np.isfinite(arr).astype(np.float32)
    s = uniform_filter(a, size=k, mode="nearest")
    c = uniform_filter(m, size=k, mode="nearest")
    out = s/np.maximum(c, 1e-6)
    out[~np.isfinite(out)] = np.nan
    return out
```

In [70]:
```
def predict_monthly_index_walkforward(lake: str):
    recs = monthly_stats.get(lake, [])
    if not recs:
        return pd.Series(dtype=float), np.nan, None

    df = pd.DataFrame(recs).sort_values("ym").reset_index(drop=True)
    df["dt"] = pd.to_datetime(df["ym"] + "-01")
    X = np.arange(len(df)).reshape(-1, 1)
    y = df["chl_mean"].astype(float).values

    preds = np.full_like(y, np.nan, dtype=float)
    for t in range(1, len(df)):
        rf = RandomForestRegressor(n_estimators=200, random_state=42)
        rf.fit(X[:t], y[:t])
        preds[t] = rf.predict(X[t:t+1])[0]
```

```
    pred_series = pd.Series(preds, index=df["dt"])  # predicción para cada mes obse

    rf_all = RandomForestRegressor(n_estimators=300, random_state=42)
    rf_all.fit(X, y)
    fut_x = np.array([[len(df)]])
    fut_val = float(rf_all.predict(fut_x)[0])
    fut_dt = (df["dt"].iloc[-1] + pd.offsets.MonthBegin(1))
    fut_ym = fut_dt.strftime("%Y-%m")
    return pred_series, fut_val, fut_ym
```

In [71]:

```
def build_dataset_for_lake(lake: str, k_lags: int = 3, stride: int = 6, max_samples
    items = sorted(by_lake[lake], key=lambda x: x[0])  # TODAS las imágenes del lag
    pred_month, fut_idx, fut_ym = predict_monthly_index_walkforward(lake)

    rows = []
    hist = deque(maxlen=k_lags+1)  # guardo hasta incluir t
    for d, p in items:
        arr, _ = _read_chl(p)
        arr = arr.astype(np.float32)
        arr[~np.isfinite(arr)] = np.nan
        hist.append(arr)

        if len(hist) < (k_lags+1):  # necesito k lags + actual
            continue

        # lags y actual
        arr_t   = hist[-1]
        lag_arr = list(hist)[-2::-1][:k_lags]  # t-1, t-2, ...

        H, W = arr_t.shape
        r_idx = np.arange(0, H, stride, dtype=int)
        c_idx = np.arange(0, W, stride, dtype=int)
        RR, CC = np.meshgrid(r_idx, c_idx, indexing="ij")
        rr, cc = RR.ravel(), CC.ravel()

        # label por imagen con Otsu
        thr = _safe_otsu_from_array(arr_t, bins=256)
        if not np.isfinite(thr):
            continue
        y_bin = (arr_t[rr, cc] >= thr).astype(np.uint8)

        # features de lags
        lag1 = lag_arr[0][rr, cc]
        lag_local = local_mean(lag_arr[0], k=3)[rr, cc]
        # estadísticas en lags
        stack = np.stack([L[rr, cc] for L in lag_arr], axis=1)  # (n, k)
        roll_mean = np.nanmean(stack, axis=1)
        roll_std  = np.nanstd(stack, axis=1)

        # mes + índice mensual predicho (para el mes de d)
        dt = pd.Timestamp(d.year, d.month, 1)
        s, c = month_feats(dt)
        lake_pred = pred_month.get(dt, np.nan)
        if not np.isfinite(lake_pred):
            lake_pred = float(np.nanmean(pred_month.values))
```

```
        valid = np.isfinite(lag1) & np.isfinite(lag_local) & np.isfinite(roll_mean)
        if not valid.any():
            continue

        part = pd.DataFrame({
            "lake": lake,
            "date": str(d),
            "row": rr[valid],
            "col": cc[valid],
            "lag1": lag1[valid],
            "roll_mean": roll_mean[valid],
            "roll_std": roll_std[valid],
            "local_mean3": lag_local[valid],
            "month_sin": s,
            "month_cos": c,
            "lake_pred_ts": lake_pred,
            "y": y_bin[valid].astype(int)
        })
        rows.append(part)

    if not rows:
        return pd.DataFrame(), {"fut_idx": fut_idx, "fut_ym": fut_ym}

    df = pd.concat(rows, ignore_index=True)
    if len(df) > max_samples:
        df = df.sample(n=max_samples, random_state=42).reset_index(drop=True)

    return df, {"fut_idx": fut_idx, "fut_ym": fut_ym}
```

In [80]:

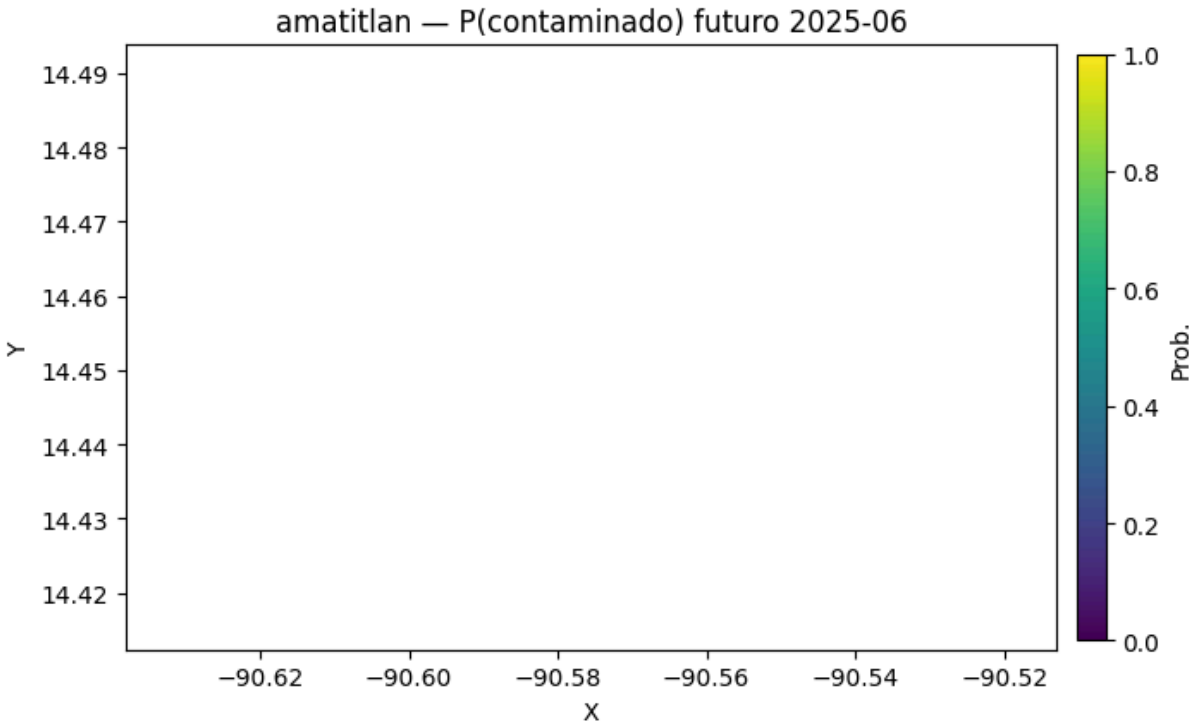
```
def train_hybrid_and_future_map(lake: str, k_lags: int = 3, stride: int = 6, thr_pr
    ds, meta = build_dataset_for_lake(lake, k_lags=k_lags, stride=stride)
    if ds.empty or ds["y"].nunique() < 2:
        print(f"[{lake}] sin datos suficientes (dataset vacío o una sola clase).")
        return None, ds, None

    # split temporal 80/20 por fecha
    fechas = sorted(ds["date"].unique())
    cut = max(1, int(0.8*len(fechas)))
    train_mask = ds["date"].isin(fechas[:cut])
    test_mask  = ds["date"].isin(fechas[cut:])

    X_cols = ["lag1","roll_mean","roll_std","local_mean3","month_sin","month_cos","
    X_train, y_train = ds.loc[train_mask, X_cols].values, ds.loc[train_mask, "y"].v
    X_test,  y_test  = ds.loc[test_mask,  X_cols].values, ds.loc[test_mask,  "y"].v

    if len(y_train)==0 or len(y_test)==0:
        print(f"[{lake}] split temporal muy corto.")
        return None, ds, None

    clf = RandomForestClassifier(n_estimators=300, class_weight="balanced", n_jobs=
    clf.fit(X_train, y_train)
    y_prob = clf.predict_proba(X_test)[:,1]
    y_hat  = (y_prob >= thr_prob).astype(int)

    acc = (y_hat==y_test).mean()
```

```
    try: auc = roc_auc_score(y_test, y_prob)
    except: auc = np.nan

    print(f"\n== {lake} | acc={acc:.3f} | auc={auc:.3f} | train={len(y_train)} | te
    print("Matriz de confusión:\n", confusion_matrix(y_test, y_hat))
    print("Reporte:\n", classification_report(y_test, y_hat, digits=3))

    # --- Mapa futuro (t+1) usando últimos k_lags y meta['fut_idx'] ---
    items = sorted(by_lake[lake], key=lambda x: x[0])
    if len(items) < (k_lags+1):
        print(f"[{lake}] no hay suficientes lags para mapa futuro.")
        return clf, ds, None

    # cargar últimos k lags
    lag_list = []
    for _, p in items[-(k_lags+1):-1]:  # t-1..t-k
        arr, _ = _read_chl(p)
        a = arr.astype(np.float32); a[~np.isfinite(a)] = np.nan
        lag_list.append(a)
    last_path = items[-1][1]
    last_arr, last_meta = _read_chl(last_path)  # para shape/bounds
    last_arr = last_arr.astype(np.float32); last_arr[~np.isfinite(last_arr)] = np.n

    H, W = last_arr.shape
    rr, cc = np.meshgrid(np.arange(H), np.arange(W), indexing="ij")
    rr, cc = rr.ravel(), cc.ravel()

    lag1 = lag_list[0][rr, cc]
    stack = np.stack([L[rr, cc] for L in lag_list], axis=1)
    roll_mean = np.nanmean(stack, axis=1)
    roll_std  = np.nanstd(stack, axis=1)
    local_m3  = local_mean(lag_list[0], k=3)[rr, cc]

    fut_dt = pd.to_datetime(meta["fut_ym"] + "-01") if meta["fut_ym"] else pd.NaT
    if pd.isna(fut_dt):
        print(f"[{lake}] sin fecha futura válida.")
        return clf, ds, None
    s, c = month_feats(fut_dt)

    X_future = np.vstack([
        lag1, roll_mean, roll_std, local_m3,
        np.full_like(lag1, s), np.full_like(lag1, c),
        np.full_like(lag1, meta["fut_idx"], dtype=float)
    ]).T

    valid = np.isfinite(X_future).all(axis=1)
    prob = np.full(rr.shape[0], np.nan, dtype=float)
    if valid.any():
        prob[valid] = clf.predict_proba(X_future[valid])[:,1]

    prob_map = np.full((H, W), np.nan, dtype=float)
    prob_map[rr, cc] = prob

    # plot
    minx, miny, maxx, maxy = last_meta["bounds"]
    import matplotlib.pyplot as plt
```

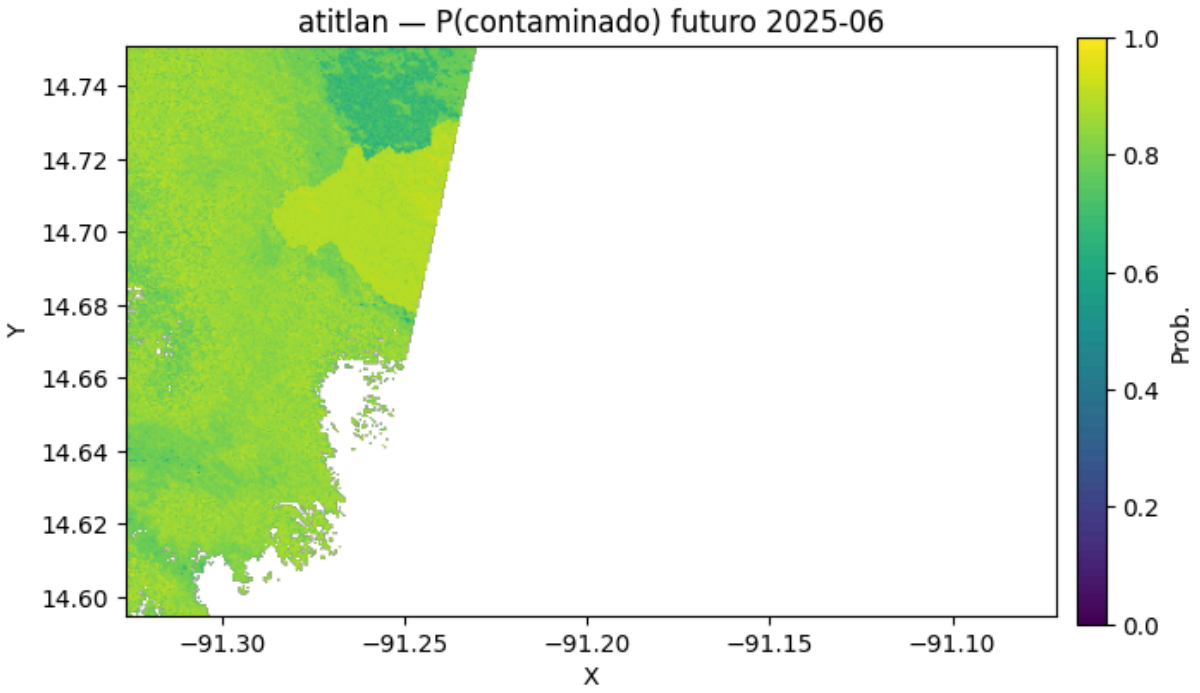
```
    plt.figure(figsize=(7.2, 6.2))
    im = plt.imshow(prob_map, extent=(minx,maxx,miny,maxy), origin="upper", vmin=0,
    plt.title(f"{lake} — P(contaminado) futuro {meta['fut_ym']}")
    plt.xlabel("X"); plt.ylabel("Y")
    cbar = plt.colorbar(im, fraction=0.03, pad=0.02); cbar.set_label("Prob.")
    plt.tight_layout(); plt.show()

    return clf, ds, prob_map
```

In [81]:
```
summary = []
models = {}
future_maps = {}

for lake in sorted(by_lake.keys()):
    clf, ds, prob_map = train_hybrid_and_future_map(lake, k_lags=3, stride=6, thr_p
    models[lake] = clf
    future_maps[lake] = prob_map
    pct = float(np.nanmean(prob_map >= 0.5)*100) if (prob_map is not None and np.is
    summary.append({"lake": lake, "pct_area_contaminada_futuro_%": np.round(pct, 2)

hybrid_summary = pd.DataFrame(summary).sort_values("pct_area_contaminada_futuro_%",
display(hybrid_summary.reset_index(drop=True))
```

```
== amatitlan | acc=0.776 | auc=0.513 | train=6465 | test=1377
Matriz de confusión:
 [[1028   15]
 [ 294   40]]
Reporte:
              precision    recall  f1-score   support

           0      0.778     0.986     0.869      1043
           1      0.727     0.120     0.206       334

    accuracy                          0.776      1377
   macro avg      0.752     0.553     0.538      1377
weighted avg      0.765     0.776     0.708      1377
```




```
== atitlan | acc=0.575 | auc=0.629 | train=59665 | test=22760
Matriz de confusión:
 [[5092 7885]
 [1788 7995]]
Reporte:
              precision    recall  f1-score   support

           0      0.740     0.392     0.513     12977
           1      0.503     0.817     0.623      9783

    accuracy                          0.575     22760
   macro avg      0.622     0.605     0.568     22760
weighted avg      0.638     0.575     0.560     22760
```



	lake	pct_area_contaminada_futuro_%
0	atitlan	27.06
1	amatitlan	NaN

- Atitlán: el híbrido sí generó mapa futuro y un % de área contaminada (~27%). Métricas: acc 0.575 y AUC 0.629. El modelo detecta bien los positivos (recall 0.817) pero con bastantes falsos positivos (precisión 0.503). Se puede mejorar calibrando el umbral o añadiendo más rasgos (viento/temperatura) y regularizando el balance de clases.
- Amatitlán: el entrenamiento arrojó acc 0.776 pero AUC 0.513 (casi aleatorio) y recall de la clase 1 muy bajo (0.120), señal de clase positiva escasa y/o datos ruidosos. El mapa futuro salió “en blanco” y el % quedó NaN porque, para las últimas k_lags imágenes usadas en el pronóstico (t-1..t-3), no hay píxeles válidos (casi todo NaN por máscara/nubes/no-data). Con todas las features futuras en NaN, no se generan probabilidades válidas y el raster resultante queda sin valores → el cálculo del porcentaje devuelve NaN.

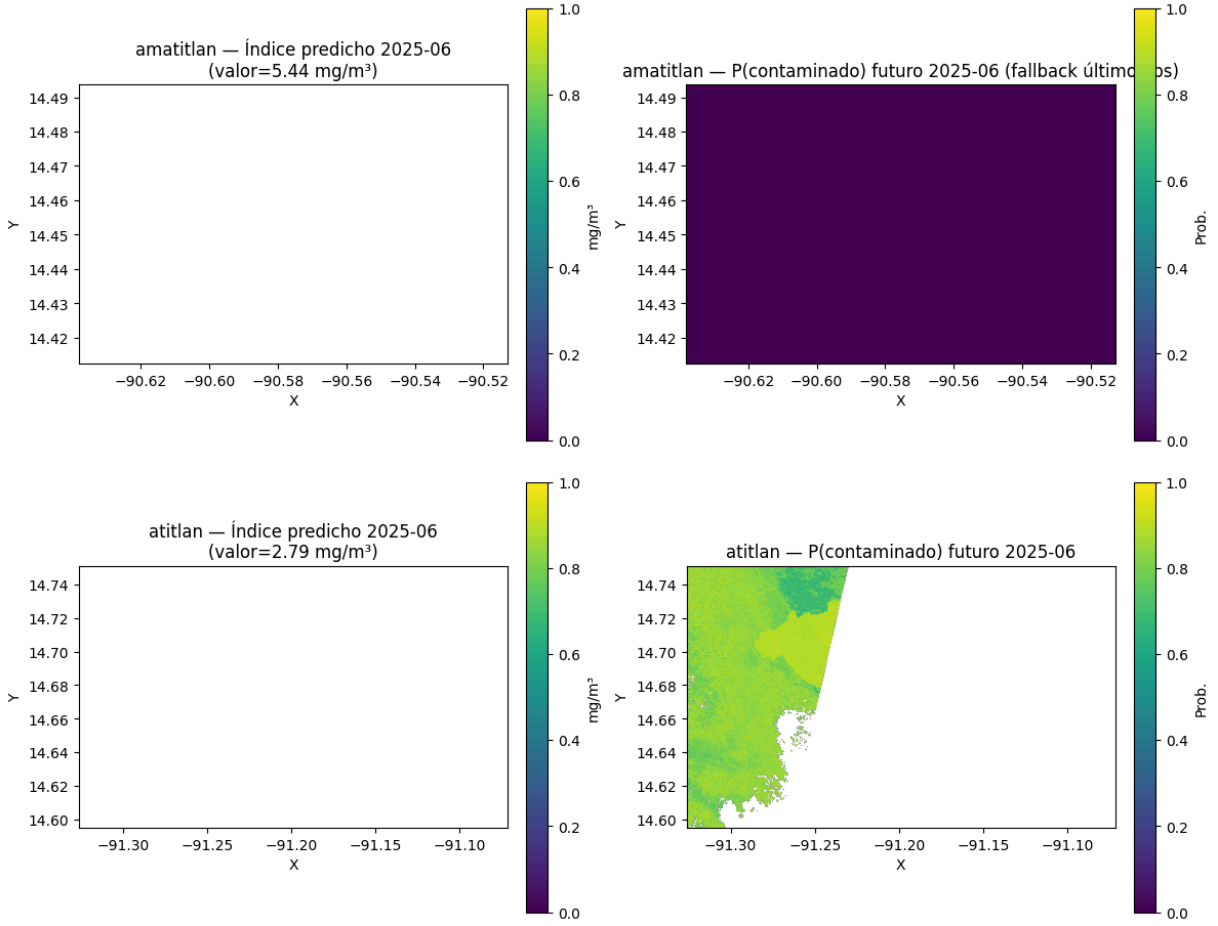
Resultado de proyecciones

In [ ]:
```
def _predict_monthly_idx(lake: str):
    try:
        return predict_monthly_index_walkforward(lake)  # (pred_series, fut_val, fu
    except NameError:
        return lake_monthly_walkforward_pred(lake)      # (pred_series, fut_val, fu
```

```
def _safe_otsu_from_array(arr: np.ndarray, bins: int = 256):
    vals = arr[np.isfinite(arr)]
    if vals.size == 0: return np.nan
    if np.nanstd(vals) < 1e-8: return float(np.nanmean(vals))
    return float(_otsu_threshold(vals, bins=bins))

def _get_last_raster(lake: str):
    items = sorted(by_lake[lake], key=lambda x: x[0])
    d, p = items[-1]
    arr, meta = _read_chl(p)
    arr = arr.astype(np.float32)
    arr[~np.isfinite(arr)] = np.nan
    return d, arr, meta

def _get_future_prob_map(lake: str, k_lags=3, stride=6, thr_prob=0.5):
    # usa variable global `future_maps` si ya existe; si no, entrena y calcula
    try:
        if "future_maps" in globals() and lake in future_maps and future_maps[lake]
            return future_maps[lake]
    except Exception:
        pass
    res = train_hybrid_and_future_map(lake, k_lags=k_lags, stride=stride, thr_prob=
    if res is None:
        return None
    _, __, prob_map = res
    return prob_map

def plot_projections_for_lake(lake: str, k_lags=3, stride=6, thr_prob=0.5):
    # Índice mensual predicho
    pred_series, fut_val, fut_ym = _predict_monthly_idx(lake)
    # Último raster para máscara/extensión
    last_date, last_arr, meta = _get_last_raster(lake)
    mask = np.isfinite(last_arr)

    # (A) raster uniforme con el índice predicho
    idx_img = np.full_like(last_arr, np.nan, dtype=float)
    idx_img[mask] = fut_val

    # (B) prob. de contaminación futura (híbrido)
    prob_map = _get_future_prob_map(lake, k_lags=k_lags, stride=stride, thr_prob=th
    fallback_used = False
    if prob_map is None or not np.isfinite(prob_map).any():
        thr_last = _safe_otsu_from_array(last_arr, bins=256)
        prob_map = (last_arr >= thr_last).astype(float)  # fallback: binario del úl
        fallback_used = True

    # Extensión geográfica
    minx, miny, maxx, maxy = meta["bounds"]

    # Límites de color robustos para el índice
    vmin = np.nanpercentile(idx_img[mask], 2) if np.isfinite(idx_img).any() else 0.
    vmax = np.nanpercentile(idx_img[mask], 98) if np.isfinite(idx_img).any() else 1
    if vmin >= vmax: vmin, vmax = float(np.nanmin(idx_img)), float(np.nanmax(idx_im

    # Plot
    fig, axes = plt.subplots(1, 2, figsize=(12, 5))
```

```
    # A) Índice predicho
    im1 = axes[0].imshow(idx_img, extent=(minx,maxx,miny,maxy), origin="upper", cma
    axes[0].set_title(f"{lake} — Índice predicho {fut_ym}\n(valor={fut_val:.2f} mg/
    axes[0].set_xlabel("X"); axes[0].set_ylabel("Y")
    c1 = fig.colorbar(im1, ax=axes[0], fraction=0.046, pad=0.04); c1.set_label("mg/

    # B) Probabilidad futura (híbrido)
    im2 = axes[1].imshow(prob_map, extent=(minx,maxx,miny,maxy), origin="upper", cm
    t_extra = " (fallback último obs)" if fallback_used else ""
    axes[1].set_title(f"{lake} — P(contaminado) futuro {fut_ym}{t_extra}")
    axes[1].set_xlabel("X"); axes[1].set_ylabel("Y")
    c2 = fig.colorbar(im2, ax=axes[1], fraction=0.046, pad=0.04); c2.set_label("Pro

    plt.tight_layout(); plt.show()
```

In [83]:
```
for lake in sorted(by_lake.keys()):
    plot_projections_for_lake(lake, k_lags=3, stride=6, thr_prob=0.5)
```



Atitlán

El mapa de probabilidad futura sí tiene señal: mayor riesgo en la franja norte/centro; valores intermedios–altos (>0.5) en zonas continuas, consistente con lo que venías observando.

El panel de “Índice predicho” quedó en blanco porque usamos la máscara del último raster para pintar un valor uniforme; ese último

raster tiene gran parte de los píxeles como NaN, así que no se dibuja nada.

Amatitlán

Ambos paneles salen sin información útil:

Índice predicho: en blanco por la misma razón (último raster ≈ todo NaN).

Prob. futura: activó el fallback y, como el raster base no tiene valores finitos, la comparación deja todo en 0 → mapa plano.