

MÓDULO 3

Selección del modelo óptimo para programación

Coste · Latencia · Calidad

Plan de Formación — OpenAI Codex para Desarrolladores

Nivel: Intermedio · Duración estimada: 3–4 horas

Versión 1.0 — Febrero 2026

Índice

1. Introducción

Elegir el modelo correcto es una de las decisiones con mayor impacto en la productividad, el coste y la calidad del trabajo con Codex. OpenAI ofrece una familia creciente de modelos especializados para programación, cada uno optimizado para un perfil de tarea distinto. Usar siempre el modelo más potente es un desperdicio; usar siempre el más barato compromete la calidad.

En este módulo construiremos un marco de decisión fundamentado: el alumno aprenderá a evaluar cada modelo según tres ejes (coste, latencia, calidad), entenderá cómo funciona el razonamiento adaptativo, y creará una política interna de selección de modelos para su equipo basada en datos reales.

2. Objetivos de aprendizaje

| # | Objetivo | Evidencia de logro |
|----|---|--|
| O1 | Elegir el modelo adecuado según el tipo de tarea: generación, refactor, debugging, tareas largas agentic. | Completa la tabla de decisión modelo→tarea con justificación. |
| O2 | Entender la diferencia práctica entre modelos de razonamiento y no razonamiento, incluyendo reasoning effort. | Explica cuándo usar cada nivel de esfuerzo con ejemplo concreto. |
| O3 | Conocer el catálogo completo de modelos Codex y sus características diferenciales. | Identifica correctamente las capacidades de cada modelo de la familia. |
| O4 | Aplicar la estrategia “escalar cuando se atasque” en un benchmark real. | Completa L3 con tabla comparativa de 3 perfiles. |

3. Contenidos teóricos

3.1 Catálogo de modelos Codex (febrero 2026)

Codex opera sobre una familia de modelos optimizados específicamente para tareas de ingeniería de software. Estos modelos son versiones de la familia GPT-5 entrenadas adicionalmente con reinforcement learning sobre tareas reales de codificación, revisión de código, debugging y operaciones de terminal.

3.1.1 Modelos actuales y sus características

| Modelo | Descripción | Puntos fuertes | Reasoning effort |
|--------------------|---|--|-----------------------------|
| GPT-5.3-Codex | Modelo más avanzado. Combina coding frontier + razonamiento + conocimiento profesional. 25% más rápido que GPT-5.2-Codex. | Tareas largas con investigación, uso de herramientas y ejecución compleja. Mejor colaboración en tiempo real (steering). | low / medium / high / xhigh |
| GPT-5.2-Codex | Modelo anterior de referencia. Optimizado para coding agentic de larga duración con context compaction. | Refactors y migraciones grandes, entornos Windows, ciberseguridad. Excelente para API. | low / medium / high / xhigh |
| GPT-5.1-Codex-Max | Primer modelo nativo con compaction multi-contexto. Puede trabajar 7+ horas autónomamente. | Tareas de horizonte largo que desbordan el contexto de otros modelos. | low / medium / high / xhigh |
| GPT-5.1-Codex-Mini | Versión pequeña y económica de GPT-5.1-Codex. ~4x más mensajes dentro del límite del plan. | Tareas simples, bien definidas, con validación automática (tests). Maximiza límites de uso. | low / medium / high |

3.1.2 Modelos legacy (aún disponibles)

| Modelo | Estado | Sucesor | Notas |
|------------------|----------|--------------------|--|
| GPT-5.1-Codex | Sucedido | GPT-5.2-Codex | Sigue disponible; usar GPT-5.2-Codex para nuevos proyectos |
| GPT-5-Codex | Sucedido | GPT-5.1-Codex | Primer modelo Codex dedicado. API: \$1.25/1M input, \$10/1M output |
| GPT-5-Codex-Mini | Sucedido | GPT-5.1-Codex-Mini | Versión económica original |
| o4-mini | Sucedido | GPT-5.1-Codex | Modelo de razonamiento previo a la familia GPT-5 |

3.1.3 Modelos general-purpose compatibles

Además de los modelos Codex, puedes apuntar Codex a cualquier modelo que soporte Chat Completions o Responses API. Opciones relevantes:

- GPT-5.2: modelo flagship general. Ideal para tareas que combinan código con análisis, redacción o investigación.
- GPT-5.2-Pro: mayor compute de razonamiento. Para tareas donde la calidad justifica el coste extra.

- GPT-4.1: modelo económico para tareas sencillas que no requieren razonamiento profundo.
- Modelos locales (via --oss): Codex soporta modelos locales vía Ollama con el flag codex --oss.

 **Consejo**

No puedes cambiar el modelo por defecto de las **tareas cloud** (chatgpt.com/codex). Para control total del modelo, usa el **CLI** o la **extensión IDE**.

3.2 Razonamiento adaptativo: reasoning effort

Los modelos Codex son modelos de razonamiento: antes de generar código, producen una “cadena de pensamiento” interna (chain of thought) donde descomponen el problema, planifican la solución y verifican su lógica. El parámetro reasoning effort controla cuánto tiempo y tokens dedica el modelo a esta fase de pensamiento.

3.2.1 Niveles de reasoning effort

| Nivel | Comportamiento | Consumo de tokens | Caso de uso óptimo |
|----------------------|---|-------------------|--|
| none (solo GPT-5.2) | Sin razonamiento interno. Respuesta directa. | Mínimo | Autocompletado, respuestas triviales, baja latencia |
| low | Razonamiento mínimo. Respuesta rápida. | Bajo | Q&A sobre código, explicaciones simples, tareas mecánicas |
| medium (recomendado) | Balance óptimo inteligencia/velocidad. | Moderado | Desarrollo interactivo diario, generación de código, tests |
| high | Razonamiento profundo. Más lento. | Alto | Bugs complejos, refactors multiarchivo, migraciones |
| xhigh | Máximo razonamiento. Puede pensar mucho tiempo. | Muy alto | Tareas de días, auditorías de seguridad, arquitectura compleja |

3.2.2 Cómo cambiar el reasoning effort

```
# Vía config.toml
model_reasoning_effort = "medium"

# Vía CLI flag
codex --config 'model_reasoning_effort="high"'

# Vía perfil
[profiles.deep]
model_reasoning_effort = "high"

# Vía extensión IDE
# Selector bajo el input de chat: low / medium / high

# Vía API (Responses API)
# reasoning: { effort: "high" }
```

3.2.3 Context compaction: razonamiento de larga duración

Cuando una sesión se acerca al límite de la ventana de contexto, Codex ejecuta automáticamente context compaction: resume el historial preservando el contexto más relevante para liberar espacio y continuar sin perder progreso. Esto permite tareas que se extienden durante horas con millones de tokens procesados.

Requisito: La compaction es nativa en GPT-5.1-Codex-Max y posteriores. En modelos anteriores puede requerir configuración explícita.

Impacto práctico: Un refactor que antes fallaba al 60% por agotamiento de contexto ahora puede completarse de extremo a extremo.

3.3 Guía de decisión: qué modelo para qué tarea

Esta tabla de decisión sintetiza las recomendaciones oficiales y la experiencia práctica para elegir el modelo adecuado según el tipo de tarea:

| Tipo de tarea | Modelo recomendado | Reasoning effort | Justificación |
|--|-------------------------------|------------------|--|
| Generación de código (features nuevas) | GPT-5.3-Codex | medium | Balance óptimo calidad/velocidad para desarrollo diario |
| Refactorización multiarchivo | GPT-5.3-Codex / GPT-5.2-Codex | high | Requiere comprender dependencias entre módulos |
| Debugging complejo | GPT-5.3-Codex | high | Necesita rastrear causa raíz a través de múltiples capas |
| Migración de lenguaje/framework | GPT-5.2-Codex / 5.1-Max | xhigh | Tareas largas con muchos ficheros y patrones a transformar |
| Generación de tests unitarios | GPT-5.3-Codex | medium | Tarea bien definida con validación inmediata |
| Documentación técnica (docstrings, README) | GPT-5.3-Codex / Mini | medium / low | Tarea mecánica; Mini ahorra límites de uso |
| Code review de PR | GPT-5.3-Codex | medium – high | Necesita comprender contexto y detectar errores sutiles |
| Hotfix urgente con scope acotado | GPT-5.3-Codex | medium | Rápido, scope reducido, validado por tests |
| Exploración de codebase / onboarding | GPT-5.3-Codex (read-only) | low – medium | Solo lectura, no requiere razonamiento profundo |
| Tareas repetitivas bien definidas (renombrar, formatear) | GPT-5.1-Codex-Mini | low | Tarea mecánica con output predecible. Maximiza límites. |
| Tareas de 1+ hora (auditoría, refactor masivo) | GPT-5.1-Codex-Max / Cloud | xhigh | Soporte nativo de compaction, duración extendida |
| Tareas mixtas (código + investigación + redacción) | GPT-5.3-Codex | medium – high | Combina coding frontier con conocimiento profesional |
| <p>💡 Estrategia “Escalar cuando se atasque”</p> <p>La recomendación general de OpenAI es usar medium reasoning effort como daily driver. Solo escalar a high/xhigh cuando la tarea lo requiera o cuando el modelo se atasque. GPT-5.1-Codex-Max con medium reasoning obtiene mejor rendimiento que GPT-5.1-Codex con el mismo esfuerzo, usando 30% menos thinking tokens.</p> | | | |

3.4 El triángulo coste–latencia–calidad

3.4.1 Modelo de costes

Codex opera bajo dos modelos de coste según el método de autenticación:

Autenticación ChatGPT (planes Plus/Pro/Business/Enterprise): Incluido en la suscripción con límites de uso. Al alcanzar el límite, se pueden comprar créditos adicionales. GPT-5.1-Codex-Mini extiende los límites ~4x.

Autenticación API key: Pago por tokens consumidos según pricing del modelo. Los reasoning tokens (pensamiento interno) cuentan como output tokens.

3.4.2 Pricing API (referencia, sujeto a cambios)

| Modelo | Input (1M tokens) | Cached input | Output (1M tokens) | Contexto |
|--------------------------|----------------------|--------------|--------------------|-------------|
| GPT-5-Codex (referencia) | \$1.25 | \$0.125 | \$10.00 | 400K tokens |
| codex-mini-latest | \$1.50 | — | — | 200K tokens |
| GPT-5.2 (general) | Según pricing actual | — | — | 1M tokens |

Nota: los precios de GPT-5.3-Codex y GPT-5.2-Codex para API están pendientes de publicación al momento de redacción. Consultar <https://developers.openai.com/codex/pricing/> para valores actualizados.

3.4.3 Impacto del reasoning effort en el coste

Los reasoning tokens (la “cadena de pensamiento” interna del modelo) se facturan como output tokens. A mayor reasoning effort, más tokens de pensamiento se generan:

| Reasoning effort | Tokens de pensamiento (estimado) | Impacto en coste | Impacto en latencia |
|------------------|----------------------------------|---------------------|------------------------------------|
| none | 0 | Menor coste posible | Mínima latencia |
| low | Pocos cientos | Bajo | Rápido |
| medium | Miles | Moderado | Normal |
| high | Decenas de miles | Alto | Notablemente más lento |
| xhigh | Cientos de miles+ | Muy alto | Puede tardar minutos por respuesta |

⚠ Cuidado con xhigh

El nivel xhigh puede agotar **rate limits rápidamente**. OpenAI recomienda reservarlo para tareas críticas donde la calidad justifica el coste y la espera. Para la mayoría de tareas diarias, medium es suficiente.

3.5 Razonamiento vs. no razonamiento: diferencias prácticas

Todos los modelos Codex son modelos de razonamiento (producen chain of thought interna). Sin embargo, el grado de razonamiento es configurable, y GPT-5.2 (general-purpose) soporta reasoning effort = none que desactiva el razonamiento por completo.

3.5.1 Cuándo activar razonamiento profundo

- La tarea requiere planificación multi-paso (ej. refactor que afecta a 10+ ficheros).
- El bug es sutil y requiere rastrear interacciones entre componentes.
- La migración necesita comprender patrones del framework origen Y destino.
- La tarea de seguridad requiere analizar flujos de datos complejos.

3.5.2 Cuándo reducir o desactivar razonamiento

- La tarea es mecánica: renombrar variable, añadir import, formatear.
- El output es directamente verificable por tests automáticos.
- Necesitas baja latencia (ej. autocompletado interactivo).
- El prompt es muy específico y no hay ambigüedad.

3.5.3 Reasoning summaries

Los modelos Codex producen resúmenes de razonamiento (reasoning summaries) como actualizaciones de progreso mientras trabajan. Estos aparecen como texto efímero en el TUI del CLI. Son generados por un modelo separado y no son controlables via prompt. GPT-5.3-Codex mejora significativamente estas actualizaciones, haciéndolas más frecuentes e informativas.

3.6 Cómo cambiar de modelo en cada superficie

| Superficie | Cómo cambiar | Permanencia |
|-------------------|--|----------------------------|
| CLI (config.toml) | model = "gpt-5.3-codex" en ~/.codex/config.toml | Todas las sesiones futuras |
| CLI (flag) | codex -m gpt-5.2-codex o codex --model gpt-5.2-codex | Solo esa ejecución |
| CLI (perfil) | codex --profile deep-review (donde el perfil define model) | Solo esa ejecución |
| CLI (en sesión) | Comando /model y seleccionar del picker | Resto de la sesión |
| IDE Extension | Dropdown de modelo bajo el input de chat | Resto de la sesión |
| Cloud (web) | No configurable (usa el modelo por defecto del cloud) | N/A |
| API (Responses) | Parámetro model en la request | Por request |

4. Buenas prácticas

4.1 Emparejar modelo con grado de verificación automática

La regla fundamental es: cuanto menos capacidad de verificación automática tengas, más inteligente debe ser el modelo y más revisión humana necesitas.

| Escenario | Tests automáticos | Modelo recomendado | Revisión humana |
|--|-------------------|---------------------------|-----------------------------------|
| Suite de tests completa (>80% cobertura) | Sí, exhaustivos | Mini o medium effort | Mínima (verificar tests pasan) |
| Tests parciales (<50% cobertura) | Parciales | GPT-5.3-Codex, medium | Moderada (revisar diffs) |
| Sin tests | No | GPT-5.3-Codex, high | Exhaustiva (revisar cada línea) |
| Código crítico (seguridad, pagos) | Sí + auditoría | GPT-5.3-Codex, high/xhigh | Revisión de seguridad obligatoria |

4.2 Escalar solo cuando se atasque

Paso 1: Empieza con GPT-5.1-Codex-Mini (o GPT-5.3-Codex con medium) para la tarea.

Paso 2: Si el resultado no es satisfactorio (tests fallan, lógica incorrecta, cambios innecesarios), sube a GPT-5.3-Codex con high.

Paso 3: Si aún se atasca, sube a xhigh o cambia a un modelo más potente.

Paso 4: Documenta el resultado para tu tabla de decisión interna.

4.3 Definir perfiles por tipo de trabajo

```
# ~/.codex/config.toml - Perfiles corporativos

# Daily driver (default)
model = "gpt-5.3-codex"
model_reasoning_effort = "medium"

# Feature grande / refactor
[profiles.heavy]
model = "gpt-5.3-codex"
model_reasoning_effort = "high"

# Tareas simples / repetitivas
[profiles.quick]
model = "gpt-5.1-codex-mini"
model_reasoning_effort = "low"

# Revisión de código
[profiles.review]
model = "gpt-5.3-codex"
model_reasoning_effort = "high"
approval_policy = "never" # No interrumpir durante review
```

```
# Uso:  
codex                         # daily driver (medium)  
codex --profile heavy          # feature grande  
codex --profile quick          # tarea simple  
codex --profile review         # code review
```

4.4 Medir para decidir

- Registrar por tarea: modelo usado, reasoning effort, tiempo total, iteraciones necesarias, tests verdes al primer intento.
- Usar codex exec --json para capturar métricas parseables de ejecuciones automáticas.
- Habilitar OpenTelemetry ([otel] en config.toml) para trazar runs a nivel enterprise.
- Revisar periódicamente: si un modelo inferior resuelve consistentemente un tipo de tarea, bajarlo en la política.

5. Errores comunes y cómo evitarlos

5.1 Usar el modelo más caro para todo

El problema: Configurar siempre xhigh reasoning con el modelo más potente. Resultado: rate limits agotados a media mañana, latencia innecesaria en tareas simples, coste desproporcionado en API.

La solución: Usar medium como default. Configurar perfiles para escalar solo cuando la tarea lo requiera. GPT-5.1-Codex-Mini para tareas mecánicas extiende los límites ~4x.

5.2 No medir: latencia, coste por tarea, tasa de retrabajo

El problema: El equipo “siente” que un modelo es mejor sin datos. Las decisiones de modelo se basan en intuición o inercia en lugar de evidencia. No se detecta cuándo un modelo inferior sería suficiente.

La solución: Implementar un benchmark periódico (como el Lab L3 de este módulo). Registrar métricas por tarea. Habilitar telemetría para análisis agregado.

5.3 Errores adicionales

| Error | Consecuencia | Prevención |
|--|---|---|
| Fijar modelo en config.toml y olvidarse | Se queda en un modelo legacy cuando hay mejores disponibles | Revisar changelog mensualmente; probar modelos nuevos |
| Subir reasoning effort sin cambiar el prompt | Más tokens gastados pero sin mejora real | Mejorar el prompt primero; escalar effort después |
| Usar --yolo con xhigh en desarrollo local | Ejecución sin sandbox con máxima autonomía: alto riesgo | Reservar xhigh+no-sandbox para contenedores CI |
| No considerar Mini para tareas simples | Agota límites de uso 4x más rápido de lo necesario | Incluir Mini en la política de modelos del equipo |
| Ignorar context compaction | Tareas largas fallan al agotar contexto | Usar GPT-5.1-Codex-Max o posterior para tareas >1h |

6. Casos de uso reales

6.1 Política corporativa de modelos por tipo de trabajo

Escenario: Empresa con 30 desarrolladores y plan Enterprise. El equipo de plataforma necesita estandarizar qué modelo usar para cada tipo de tarea, equilibrando calidad y consumo de créditos.

Política implementada

| Tipo de trabajo | Perfil Codex | Modelo | Effort | Justificación |
|---------------------|--------------|--------------------|--------|-------------------------------------|
| Feature nueva | (default) | GPT-5.3-Codex | medium | Bestándar diario, buen balance |
| Hotfix urgente | (default) | GPT-5.3-Codex | medium | Rapidez; scope acotado + tests |
| Refactor grande | heavy | GPT-5.3-Codex | high | Comprender dependencias complejas |
| Migración framework | heavy | GPT-5.2-Codex | xhigh | Duración extendida, muchos ficheros |
| Documentación | quick | GPT-5.1-Codex-Mini | low | Mecánico; maximiza límites |
| Code review de PR | review | GPT-5.3-Codex | high | Detección de errores sutiles |
| Tests unitarios | (default) | GPT-5.3-Codex | medium | Verificable inmediatamente |

Resultado: Reducción del 35% en consumo de créditos sin pérdida de calidad. Los desarrolladores tienen perfiles preconfigurados y solo necesitan elegir el tipo de tarea.

6.2 Model routing en equipos con plantillas por tarea

Escenario: Equipo que usa Codex exec en CI/CD para revisar automáticamente cada PR, generar tests para nuevos ficheros y detectar errores de seguridad.

Implementación con GitHub Actions

```
# .github/workflows/codex-review.yml
jobs:
  review:
    steps:
      - uses: openai/codex-action@v1
        with:
          prompt-file: .github/codex/review.md
          model: gpt-5.3-codex           # Modelo potente para review
          sandbox: read-only            # Solo lectura

  generate-tests:
    steps:
      - uses: openai/codex-action@v1
        with:
          prompt: "Generate tests for new/changed files"
          model: gpt-5.1-codex-mini    # Mini para tarea mecánica
          sandbox: workspace-write
```

Resultado: Cada tipo de tarea en CI usa el modelo óptimo. Las reviews usan el modelo potente; la generación de tests usa Mini para maximizar límites.

7. Laboratorio práctico (L3) — Benchmark interno de 3 modelos

⌚ Objetivo del laboratorio

Comparar 3 perfiles de modelo/effort en la **misma tarea idéntica** y construir una tabla de decisión interna basada en datos reales. Al finalizar, el alumno tendrá evidencia empírica para recomendar modelos según tipo de tarea.

7.1 Prerrequisitos

- Codex CLI instalado y autenticado (Módulos 1 y 2).
- Acceso a al menos 2 modelos diferentes (ej. GPT-5.3-Codex + GPT-5.1-Codex-Mini).
- Python 3.10+ con pytest instalado.
- Git instalado.

7.2 Paso 1 — Crear repositorio con tarea definida

Tiempo estimado: 5 minutos

```
mkdir /tmp/codex-lab03 && cd /tmp/codex-lab03
git init

# Crear estructura
mkdir -p src tests

cat > src/user_service.py << 'EOF'
"""Servicio de gestión de usuarios (intencionalmente incompleto)."""
from dataclasses import dataclass, field
from typing import Optional
import uuid

@dataclass
class User:
    id: str = field(default_factory=lambda: str(uuid.uuid4()))
    name: str = ""
    email: str = ""
    active: bool = True


class UserService:
    """Almacén en memoria de usuarios."""

    def __init__(self):
        self._users: dict[str, User] = {}

    def create_user(self, name: str, email: str) -> User:
        user = User(name=name, email=email)
        self._users[user.id] = user
        return user
```

```

def get_user(self, user_id: str) -> Optional[User]:
    return self._users.get(user_id)

def list_users(self) -> list[User]:
    return list(self._users.values())
EOF

cat > tests/test_user_service.py << 'EOF'
from src.user_service import UserService

def test_create_user():
    svc = UserService()
    user = svc.create_user("Alice", "alice@test.com")
    assert user.name == "Alice"
    assert user.id is not None
EOF

# Verificar que el scaffold funciona
PYTHONPATH=. pytest tests/ -v

git add -A && git commit -m "chore: scaffold lab03"

```

7.3 Paso 2 — Definir la tarea idéntica

La misma tarea se ejecutará con cada perfil. Guardarla en un fichero para asegurar identidad:

```

cat > /tmp/codex-lab03/TASK.md << 'EOF'
# Tarea: Ampliar UserService

## Funcionalidad nueva
1. Añadir método `update_user(user_id, name=None, email=None)` que
actualice solo los campos proporcionados. Retornar el user actualizado.
Si user_id no existe, lanzar `KeyError`.

2. Añadir método `delete_user(user_id)` que desactive el usuario
(`active = False`). Si no existe, lanzar `KeyError`.

3. Añadir método `search_users(query)` que busque por nombre o email
(búsqueda case-insensitive parcial). Retornar lista de User activos.

## Refactorización
4. Extraer validación de email a una función `validate_email(email)`
que verifique formato básico (contiene @). Usarla en create y update.
Lanzar `ValueError` si el email es inválido.

## Tests
5. Añadir tests para TODOS los métodos nuevos incluyendo casos edge:
update existente, update inexistente, delete, search por nombre,
search por email, search sin resultados, email inválido en create,
email inválido en update.

```

```
## Validación
6. Ejecutar `PYTHONPATH=. pytest tests/ -v` y mostrar resultados.
EOF

git add TASK.md && git commit -m "chore: definir tarea benchmark"
```

7.4 Paso 3 — Crear perfiles de benchmark

```
# Crear perfiles temporales en config del proyecto
mkdir -p .codex
cat > .codex/config.toml << 'EOF'
# Perfiles para benchmark L3
approval_policy = "on-request"
sandbox_mode = "workspace-write"

[profiles.alto]
model = "gpt-5.3-codex"
model_reasoning_effort = "high"

[profiles.medio]
model = "gpt-5.3-codex"
model_reasoning_effort = "medium"

[profiles.rapido]
model = "gpt-5.1-codex-mini"
model_reasoning_effort = "low"
EOF

git add .codex/ && git commit -m "chore: perfiles benchmark"
```

7.5 Paso 4 — Ejecutar la tarea con cada perfil

Tiempo estimado: 15–25 minutos (5–10 min por perfil)

Para cada perfil, restaurar el estado original y medir el resultado:

```
# --- PERFIL 1: ALTO (gpt-5.3-codex, high) ---
cd /tmp/codex-lab03
git stash --include-untracked # Guardar estado limpio

# Registrar hora de inicio
echo "ALTO inicio: $(date +%H:%M:%S)" >> /tmp/benchmark_log.txt

# Ejecutar (modo interactivo para observar)
codex --profile alto --full-auto
# Pegar contenido de TASK.md como prompt

# Al terminar, registrar:
echo "ALTO fin: $(date +%H:%M:%S)" >> /tmp/benchmark_log.txt
PYTHONPATH=. pytest tests/ -v >> /tmp/benchmark_log.txt 2>&1
git diff --stat >> /tmp/benchmark_log.txt
```

```
# Guardar resultado y restaurar
git add -A && git stash push -m "resultado-alto"
```

```
# --- PERFIL 2: MEDIO (gpt-5.3-codex, medium) ---
git checkout . # Restaurar scaffold

echo "MEDIO inicio: $(date +%H:%M:%S)" >> /tmp/benchmark_log.txt
codex --profile medio --full-auto
# Pegar contenido de TASK.md

echo "MEDIO fin: $(date +%H:%M:%S)" >> /tmp/benchmark_log.txt
PYTHONPATH=. pytest tests/ -v >> /tmp/benchmark_log.txt 2>&1
git diff --stat >> /tmp/benchmark_log.txt
git add -A && git stash push -m "resultado-medio"
```

```
# --- PERFIL 3: RÁPIDO (gpt-5.1-codex-mini, low) ---
git checkout .

echo "RAPIDO inicio: $(date +%H:%M:%S)" >> /tmp/benchmark_log.txt
codex --profile rapido --full-auto
# Pegar contenido de TASK.md

echo "RAPIDO fin: $(date +%H:%M:%S)" >> /tmp/benchmark_log.txt
PYTHONPATH=. pytest tests/ -v >> /tmp/benchmark_log.txt 2>&1
git diff --stat >> /tmp/benchmark_log.txt
git add -A && git stash push -m "resultado-rapido"
```

7.6 Paso 5 — Completar tabla de resultados

Analizar /tmp/benchmark_log.txt y completar la tabla:

| Métrica | Alto (5.3, high) | Medio (5.3, medium) | Rápido (Mini, low) |
|---|------------------|---------------------|--------------------|
| Tiempo total (min:seg) | _____ | _____ | _____ |
| Tests verdes al primer intento (✓/✗) | _____ | _____ | _____ |
| Nº total de tests generados | _____ | _____ | _____ |
| Iteraciones necesarias (0 = primera vez) | _____ | _____ | _____ |
| Ficheros modificados | _____ | _____ | _____ |
| Cambios innecesarios detectados | _____ | _____ | _____ |
| validate_email implementado correctamente | _____ | _____ | _____ |
| Edge cases cubiertos | _____ | _____ | _____ |
| Calidad percibida (1–5) | _____ | _____ | _____ |

7.7 Paso 6 — Construir tabla de decisión del equipo

Basándose en los resultados, completar:

| Tipo de tarea | Perfil recomendado | Justificación (basada en datos) |
|------------------------------|--------------------|---------------------------------|
| Feature nueva con tests | _____ | _____ |
| Refactorización + validación | _____ | _____ |
| Tareas mecánicas/simples | _____ | _____ |
| Debugging complejo | _____ | _____ |
| Documentación | _____ | _____ |

7.8 Resultado esperado

- Tabla de métricas completada con datos reales de los 3 perfiles.
- Tabla de decisión modelo→tarea con justificación empírica.
- Comprensión práctica de las diferencias entre perfiles.
- El perfil “medio” debería resolver la tarea correctamente en la mayoría de los casos.
- El perfil “rápido” puede fallar en la refactorización o generar menos tests edge.
- El perfil “alto” puede producir código más robusto pero tardar significativamente más.

7.9 Limpieza (OBLIGATORIA)

⚠️ Limpieza de recursos

Ejecutar al finalizar el laboratorio:

```
# 1. Borrar el repositorio
rm -rf /tmp/codex-lab03

# 2. Borrar log de benchmark (puede contener datos sensibles)
rm -f /tmp/benchmark_log.txt

# 3. Si añadiste perfiles de prueba a ~/.codex/config.toml, revertirlos

# 4. Verificar limpieza
ls /tmp/codex-lab03 2>/dev/null && echo 'ERROR' || echo 'OK: limpio'
```

8. Resumen del módulo y siguientes pasos

8.1 Conceptos clave aprendidos

| Concepto | Detalle |
|----------------------------------|---|
| Familia de modelos Codex | GPT-5.3-Codex (actual), GPT-5.2-Codex, GPT-5.1-Codex-Max, GPT-5.1-Codex-Mini + legacy |
| Reasoning effort | none/low/medium/high/xhigh — controla profundidad de pensamiento y coste |
| medium como daily driver | Recomendación oficial de OpenAI para balance óptimo |
| Context compaction | Permite tareas de horas sin agotar el contexto |
| Triángulo coste-latencia-calidad | Equilibrar según tipo de tarea y verificabilidad |
| Estrategia escalar | Empezar con menor modelo/effort; subir solo si falla |
| Perfiles por tarea | Preconfigurar modelos/effort para cada tipo de trabajo |
| Mini para mecánico | GPT-5.1-Codex-Mini ~4x más mensajes en el plan |

8.2 Preparación para el Módulo 4

En el Módulo 4 profundizaremos en AGENTS.md y contexto persistente: cómo dar instrucciones permanentes al agente que sobreviven entre sesiones. Para prepararse:

- Tener claro qué modelo usar por defecto (resultado del Lab 3).
- Revisar la guía de AGENTS.md: <https://developers.openai.com/codex/guides/agents-md/>
- Pensar en qué convenciones y reglas necesita tu proyecto actual.

9. Referencias y recursos

- Modelos Codex: <https://developers.openai.com/codex/models/>
- Pricing Codex: <https://developers.openai.com/codex/pricing/>
- Codex Prompting Guide (GPT-5.2-Codex):
https://developers.openai.com/cookbook/examples/gpt-5/codex_prompting_guide/
- GPT-5.2 Prompting Guide: https://developers.openai.com/cookbook/examples/gpt-5/gpt-5_2_prompting_guide/
- GPT-5.1 Prompting Guide: https://developers.openai.com/cookbook/examples/gpt-5/gpt-5_1_prompting_guide
- Building more with GPT-5.1-Codex-Max: <https://openai.com/index/gpt-5-1-codex-max/>
- Presentación GPT-5.3-Codex: <https://openai.com/index/introducing-gpt-5-3-codex/>
- Using GPT-5.2 (API guide): <https://platform.openai.com/docs/guides/latest-model>
- PLANS.md para tareas multi-hora:
https://developers.openai.com/cookbook/articles/codex_exec_plans/
- OpenAI for Developers 2025 (resumen): <https://developers.openai.com/blog/openai-for-developers-2025/>
- Codex changelog: <https://developers.openai.com/codex/changelog/>
- CLI Reference: <https://developers.openai.com/codex/cli/reference/>