Analyse et Optimisation de la Consommation fal.ai - ReelGen AI

Date d'analyse : 26 octobre 2025
Période analysée : 25-26 octobre 2025 (24h environ)

📊 ANALYSE DE LA CONSOMMATION ACTUELLE

Synthèse de la consommation (période de 24h)

Total dépensé : ~7.57 USD

Répartition par modèle

Modèle	Utilisation	Coût unitaire	Coût total	% du budget
Luma Dream Machine (image-to-video)	13 vidéos	$0.50/vidéo	$6.50	85.9%
FLUX Dev (image-to-image)	34 megapixels	$0.03/MP	$1.02	13.5%
FLUX Dev (text-to-image)	2 megapixels	$0.025/MP	$0.05	0.6%

Observations clés

1. **Le modèle vidéo domine les coûts** : Luma Dream Machine représente 86% des dépenses
2. **Utilisation intensive** : 13 vidéos générées en 24h = forte demande en génération vidéo
3. **Coût par génération complète** (1 image + 1 vidéo) : ~$0.53 en moyenne
4. **Projection mensuelle** (à ce rythme) : ~227 USD/mois

💡 STRATÉGIES D'OPTIMISATION DE LA CONSOMMATION

1. Optimisation des Modèles Vidéo (Impact majeur - 86% des coûts)

Options moins coûteuses pour la génération vidéo

Modèle	Coût	Qualité	Vitesse	Économie vs Luma
Luma Dream Machine (actuel)	$0.50/vidéo	⭐⭐⭐⭐⭐	Rapide	-
Wan 2.5	$0.25/vidéo (5s)	⭐⭐⭐⭐	Rapide	**50%**
LTX-2 Fast	$0.18/vidéo	⭐⭐⭐	Très rapide	**64%**
Kling 2.5 Turbo Standard	$0.30/vidéo	⭐⭐⭐⭐	Rapide	**40%**
Seedance v1 Lite	$0.31/vidéo (720p)	⭐⭐⭐	Moyen	**38%**

Recommandation prioritaire : Utiliser **Wan 2.5** comme alternative principale
- Économie potentielle : **3.25 USD/jour** (50% des coûts vidéo)
- Projection mensuelle : **~113 USD au lieu de 227 USD**
- Qualité : Excellente, open-source avec support audio natif

2. Optimisation des Modèles Image (Impact modéré - 14% des coûts)

Alternatives pour la génération d'images

Modèle	Coût	Qualité	Utilisation recommandée
FLUX.1 [dev] (actuel)	$0.025/MP	⭐⭐⭐⭐⭐	Usage commercial
FLUX.1 [schnell]	$0.015/MP	⭐⭐⭐⭐	Prototypage rapide
Seedream V4	$0.03/image	⭐⭐⭐⭐	Images 4K cohérentes
Qwen-Image	$0.02/MP	⭐⭐⭐⭐	Rendu de texte supérieur

Recommandation : Garder FLUX.1 [dev] pour la qualité, ou passer à **FLUX.1 [schnell]** pour 40% d'économie

3. Optimisation Technique

Configuration API recommandée

```
// Paramètres d'optimisation fal.ai
const optimizationSettings = {
  // Multiplexing : traiter plusieurs requêtes simultanément
  max_multiplexing: 5,

  // Concurrence : instances simultanées
  min_concurrency: 1,
  max_concurrency: 3,

  // Buffer pour gérer les pics de demande
  concurrency_buffer: 1,

  // Résolution optimale (balance qualité/coût)
  image_resolution: '1024x1024', // 1MP au lieu de 2MP
  video_duration: 5, // 5 secondes suffisent pour Instagram
}
```

Meilleures pratiques

1. **Mise en cache intelligente** : Éviter de régénérer les mêmes prompts
2. **Résolution adaptative** :
   - Images : 1024x1024 (1MP) pour prévisualisation
   - Vidéos : 720p pour tests, 1080p pour la version finale
3. **Batch processing** : Grouper les requêtes pour réduire les appels API
4. **Compression** : Optimiser les images avant upload pour réduire les megapixels

---

🎨 CATALOGUE COMPLET DES MODÈLES FAL.AI

Modèles de Génération Vidéo (50+ modèles)

Catégorie Text-to-Video

Modèle	Endpoint	Prix	Caractéristiques
Veo 3.1	`fal-ai/veo3`	$0.20-$0.40/s	Audio natif, 1080p, cinématique ⭐⭐⭐⭐⭐
Veo 3.1 Fast	`fal-ai/veo3/fast`	$0.10-$0.15/s	60-80% économie, qualité maintenue
Sora 2	`fal-ai/sora/v2`	$0.30/s	Clips dynamiques avec audio
Kling 2.5 Turbo Pro	`fal-ai/kling/v2.5/turbo/pro`	$0.30/vidéo	Fluidité motion exceptionnelle
Kling 2.5 Turbo Standard	`fal-ai/kling/v2.5/turbo/standard`	$0.30/vidéo	Motion améliorée
Wan 2.5	`fal-ai/wan/v2.5`	$0.25/vidéo	Open-source, audio natif ⭐ Recommandé
Hunyuan Video	`fal-ai/hunyuan/video`	$0.25/vidéo	Haute stabilité visuelle
LTX-2 Fast	`fal-ai/ltx-2/fast`	$0.18/vidéo	Très rapide, haute fidélité
Mochi 1	`fal-ai/mochi/v1`	$0.22/vidéo	Haute fidélité motion, open-source

Catégorie Image-to-Video

Modèle	Endpoint	Prix	Caractéristiques
Luma Dream Machine (actuel)	`fal-ai/luma-dream-machine`	$0.50/vidéo	Qualité supérieure, 5s
Veo 3.1	`fal-ai/veo3/image-to-video`	$0.20-$0.40/s	Motion réaliste + audio
Kling 2.5 Turbo Pro	`fal-ai/kling/v2.5/image-to-video`	$0.30/vidéo	Fluidité cinématique ⭐
Seedance 1.0 Pro	`fal-ai/bytedance/seedance/v1/pro`	$0.62/vidéo	1080p, mouvement naturel
Seedance 1.0 Lite	`fal-ai/bytedance/seedance/v1/lite`	$0.31/vidéo	720p, économique
Wan 2.5	`fal-ai/wan/v2.5/image-to-video`	$0.25/vidéo	Diversité motion ⭐ Économique
Vidu Q2	`fal-ai/vidu/q2`	$0.35/vidéo	Qualité améliorée
PixVerse v5	`fal-ai/pixverse/v5`	$0.28/vidéo	Bon équilibre qualité/prix

Catégorie Video-to-Video (Editing)

Modèle	Endpoint	Prix	Caractéristiques
Sora 2 Remix	`fal-ai/sora/v2/remix`	$0.35/s	Transformation style + audio
Vidu Q2 Remix	`fal-ai/vidu/q2/remix`	$0.40/vidéo	Contrôle amélioré
Sync Lipsync	`fal-ai/sync/lipsync`	$0.15/s	Synchronisation labiale

Modèles de Génération d'Images (200+ modèles)

Catégorie Text-to-Image (Haute Qualité)

Modèle	Endpoint	Prix	Caractéristiques
FLUX.1 [pro]	`fal-ai/flux-pro`	$0.055/MP	Qualité ultime, usage pro
FLUX.1 [dev] (actuel)	`fal-ai/flux/dev`	$0.025/MP	Excellent rapport qualité/prix ⭐
FLUX.1 [schnell]	`fal-ai/flux/schnell`	$0.015/MP	Rapide, bonne qualité
FLUX Kontext Pro	`fal-ai/flux-pro/kontext`	$0.04/image	Édition contextuelle
FLUX General	`fal-ai/flux/general`	$0.075/MP	Usage général
Imagen 4	`fal-ai/imagen4`	$0.05/MP	Google, photorealistic
HiDream-I1 Full	`fal-ai/hidream-i1-full`	$0.04/MP	Rendu ultra-réaliste
Qwen-Image	`fal-ai/qwen-image`	$0.02/MP	Excellent rendu de texte ⭐
Seedream V4	`fal-ai/seedream/v4`	$0.03/image	4K rapide, cohérent

Catégorie Image-to-Image (Édition)

Modèle	Endpoint	Prix	Caractéristiques
FLUX.1 [dev] Image-to-Image (actuel)	`fal-ai/flux/dev/image-to-image`	$0.03/MP	Transformation fidèle
Revé (Gemini 2.5 Flash)	`fal-ai/reve`	$0.04/image	Édition par prompt de texte
Nanobanana	`fal-ai/nanobanana`	$0.0398/image	Google, édition État de l'art
FLUX LoRA	`fal-ai/flux/lora`	$0.035/MP	Style personnalisé
Codeformer	`fal-ai/codeformer`	$0.0021/MP	Restauration visage
DDColor	`fal-ai/ddcolor`	$0.001/MP	Colorisation N&B ⭐ Économique
AuraFlow	`fal-ai/aura-flow`	$0.00111/s compute	Édition intensive

Catégorie Upscaling et Enhancement

Modèle	Endpoint	Prix	Caractéristiques
ESRGAN	`fal-ai/esrgan`	$0.002/MP	Super-résolution 4x
RealESRGAN	`fal-ai/real-esrgan`	$0.003/MP	Photo-réaliste
Clarity Upscaler	`fal-ai/clarity-upscaler`	$0.005/MP	Netteté avancée

Catégorie Utilities

Modèle	Endpoint	Prix	Caractéristiques
NSFW Detector	`fal-ai/nsfw-detector`	$0.001/image	Détection contenu
Bria RMBG 2.0	`fal-ai/bria/rmbg-2.0`	$0.005/image	Suppression arrière-plan
Finegrain Eraser	`fal-ai/finegrain-eraser`	$0.18/requête	Effacement précis

Modèles de Génération Audio (50+ modèles)

Text-to-Speech

Modèle	Endpoint	Prix	Caractéristiques
Minimax Speech-02 HD	`fal-ai/minimax/speech-02-hd`	$0.08/1k chars	Voix HD, émotions
ElevenLabs TTS	`fal-ai/elevenlabs/tts`	$0.05-$0.10/1k chars	Voix ultra-réalistes
Chatterbox	`fal-ai/chatterbox`	$0.06/1k chars	Voix conversation-nelles

Music Generation

Modèle	Endpoint	Prix	Caractéristiques
Stable Audio 2.5	`fal-ai/stable-audio/2.5`	$0.15/30s	Musique haute-fidél-ité
Beatoven	`fal-ai/beatoven`	$0.12/30s	Musique + effets son-ores
Sonauto v2.2	`fal-ai/sonauto/v2.2`	$0.10/30s	Génération vocale

Speech-to-Text

Modèle	Endpoint	Prix	Caractéristiques
Whisper	`fal-ai/whisper`	$0.006/minute	Transcription précise
Wizper	`fal-ai/wizper`	$0.005/minute	Transcription rapide

Modèles LoRA Training (Personnalisation)

Modèle	Endpoint	Prix	Caractéristiques
FLUX LoRA Trainer	`fal-ai/flux/lora-trainer`	$1.50/training	<5 min, style custom
SDXL LoRA Trainer	`fal-ai/sdxl/lora-trainer`	$1.20/training	Style personnalisé

🚀 PLAN D’IMPLÉMENTATION : SÉLECTION DE MODÈLES PAR L’UTILISATEUR

Phase 1 : Architecture Backend (Prioritaire)

1.1 Extension du Schéma Base de Données

```
// Ajout au schema.prisma
model ModelPreferences {
  id                String   @id @default(cuid())
  userId            String
  user              User     @relation(fields: [userId], references: [id], onDelete: C
ascade)

  // Modèles préférés
  imageModel        String   @default("fal-ai/flux/dev")
  imageToVideoModel String   @default("fal-ai/luma-dream-machine")
  textToVideoModel  String?

  // Paramètres avancés
  imageResolution   String   @default("1024x1024")
  videoQuality      String   @default("1080p")
  videoDuration     Int      @default(5)

  // Optimisation budget
  prioritizeSpeed   Boolean  @default(false)
  prioritizeCost    Boolean  @default(false)
  prioritizeQuality Boolean  @default(true)

  createdAt         DateTime @default(now())
  updatedAt         DateTime @updatedAt

  @@unique([userId])
}

model ModelCatalog {
  id          String   @id @default(cuid())
  endpoint    String   @unique
  name        String
  category    String   // "image", "video", "audio", "utility"
  subcategory String   // "text-to-image", "image-to-video", etc.
  provider    String   // "fal-ai", "google", "openai", etc.

  // Pricing
  pricePerUnit  Float
  priceUnit     String   // "megapixel", "video", "second", etc.

  // Caractéristiques
  maxResolution String?
  hasAudio      Boolean  @default(false)
  avgSpeed      Int?     // secondes de traitement
  qualityRating Int      @default(3) // 1-5

  // Metadata
  description   String?
  features      Json?
  isActive      Boolean  @default(true)

  createdAt     DateTime @default(now())
  updatedAt     DateTime @updatedAt
}
```

1.2 API Routes

```typescript
// app/api/models/route.ts - Liste des modèles disponibles
export async function GET(request: Request) {
  const { searchParams } = new URL(request.url)
  const category = searchParams.get('category')

  const models = await prisma.modelCatalog.findMany({
    where: category ? { category } : {},
    orderBy: { qualityRating: 'desc' }
  })

  return Response.json({ models })
}

// app/api/models/preferences/route.ts - Préférences utilisateur
export async function GET(request: Request) {
  const session = await getSession()
  const preferences = await prisma.modelPreferences.findUnique({
    where: { userId: session.user.id }
  })

  return Response.json({ preferences })
}

export async function PUT(request: Request) {
  const session = await getSession()
  const data = await request.json()

  const preferences = await prisma.modelPreferences.upsert({
    where: { userId: session.user.id },
    create: { userId: session.user.id, ...data },
    update: data
  })

  return Response.json({ preferences })
}

// app/api/models/compare/route.ts - Comparaison de modèles
export async function POST(request: Request) {
  const { modelIds } = await request.json()

  const models = await prisma.modelCatalog.findMany({
    where: { id: { in: modelIds } }
  })

  // Calculer le coût estimé pour chaque modèle
  const comparison = models.map(model => ({
    ...model,
    estimatedCostPer100: calculateEstimatedCost(model, 100)
  }))

  return Response.json({ comparison })
}
```

1.3 Service de Gestion des Modèles

```typescript
// lib/model-manager.ts
export class ModelManager {
  async getOptimalModel(
    category: string,
    criteria: 'cost' | 'speed' | 'quality' | 'balanced'
  ) {
    const models = await prisma.modelCatalog.findMany({
      where: { category, isActive: true }
    })

    switch (criteria) {
      case 'cost':
        return models.sort((a, b) => a.pricePerUnit - b.pricePerUnit)[0]
      case 'speed':
        return models.sort((a, b) => (a.avgSpeed || 999) - (b.avgSpeed || 999))[0]
      case 'quality':
        return models.sort((a, b) => b.qualityRating - a.qualityRating)[0]
      case 'balanced':
        return this.calculateBalancedScore(models)[0]
    }
  }

  private calculateBalancedScore(models: ModelCatalog[]) {
    return models.map(model => ({
      ...model,
      score: (model.qualityRating * 0.4) +
             ((5 - (model.pricePerUnit / 0.1)) * 0.3) +
             ((5 - (model.avgSpeed || 30) / 6) * 0.3)
    })).sort((a, b) => b.score - a.score)
  }

  async estimateJobCost(
    imageModel: string,
    videoModel: string,
    variations: number = 3
  ) {
    const imgModel = await prisma.modelCatalog.findUnique({
      where: { endpoint: imageModel }
    })
    const vidModel = await prisma.modelCatalog.findUnique({
      where: { endpoint: videoModel }
    })

    if (!imgModel || !vidModel) return null

    // Calcul basé sur les prix réels
    const imageCost = imgModel.pricePerUnit * (imgModel.priceUnit === 'megapixel' ?
1 : 1)
    const videoCost = vidModel.pricePerUnit * variations

    return {
      imageCost,
      videoCost,
      totalCost: imageCost + videoCost,
      breakdown: {
        image: { model: imgModel.name, cost: imageCost },
        video: { model: vidModel.name, cost: videoCost, count: variations }
      }
    }
  }
}
```

Phase 2 : Interface Utilisateur

2.1 Panel de Sélection de Modèles

```
// components/model-selector.tsx
export function ModelSelector({ category, value, onChange }) {
  const [models, setModels] = useState([])
  const [sortBy, setSortBy] = useState<'cost' | 'quality' | 'speed'>('quality')

  useEffect(() => {
    fetch(`/api/models?category=${category}`)
      .then(res => res.json())
      .then(data => setModels(data.models))
  }, [category])

  const sortedModels = useMemo(() => {
    return models.sort((a, b) => {
      switch (sortBy) {
        case 'cost': return a.pricePerUnit - b.pricePerUnit
        case 'quality': return b.qualityRating - a.qualityRating
        case 'speed': return (a.avgSpeed || 999) - (b.avgSpeed || 999)
      }
    })
  }, [models, sortBy])

  return (
    <div className="space-y-fluid-sm">
      <div className="flex items-center justify-between">
        <Label>Modèle {category}</Label>
        <Select value={sortBy} onValueChange={setSortBy}>
          <SelectTrigger className="w-[180px]">
            <SelectValue />
          </SelectTrigger>
          <SelectContent>
            <SelectItem value="quality">Meilleure qualité</SelectItem>
            <SelectItem value="cost">Plus économique</SelectItem>
            <SelectItem value="speed">Plus rapide</SelectItem>
          </SelectContent>
        </Select>
      </div>

      <Select value={value} onValueChange={onChange}>
        <SelectTrigger>
          <SelectValue />
        </SelectTrigger>
        <SelectContent>
          {sortedModels.map(model => (
            <SelectItem key={model.id} value={model.endpoint}>
              <div className="flex items-center justify-between w-full">
                <span>{model.name}</span>
                <div className="flex items-center gap-2 text-xs text-muted-fore-
ground">
                  <Badge variant={
                    model.qualityRating >= 4 ? 'default' : 'secondary'
                  }>
                    {'⭐'.repeat(model.qualityRating)}
                  </Badge>
                  <span className="font-mono">
                    ${model.pricePerUnit}/{model.priceUnit}
                  </span>
                </div>
              </div>
            </SelectItem>
          ))}
        </SelectContent>
      </Select>
```

```
    </div>
  )
}
```

2.2 Panneau de Paramètres Avancés

```tsx
// components/advanced-settings.tsx
export function AdvancedSettings() {
  const [preferences, setPreferences] = useState(null)
  const [isLoading, setIsLoading] = useState(true)

  useEffect(() => {
    fetch('/api/models/preferences')
      .then(res => res.json())
      .then(data => {
        setPreferences(data.preferences)
        setIsLoading(false)
      })
  }, [])

  const handleSave = async () => {
    await fetch('/api/models/preferences', {
      method: 'PUT',
      headers: { 'Content-Type': 'application/json' },
      body: JSON.stringify(preferences)
    })
    toast.success('Préférences enregistrées')
  }

  return (
    <Card>
      <CardHeader>
        <CardTitle>Paramètres Avancés des Modèles</CardTitle>
        <CardDescription>
          Personnalisez vos modèles préférés pour la génération de contenu
        </CardDescription>
      </CardHeader>
      <CardContent className="space-y-fluid-md">
        <div className="space-y-fluid-sm">
          <Label>Priorités</Label>
          <div className="flex gap-fluid-xs">
            <Button
              variant={preferences?.prioritizeQuality ? 'default' : 'outline'}
              onClick={() => setPreferences({
                ...preferences,
                prioritizeQuality: !preferences?.prioritizeQuality,
                prioritizeCost: false,
                prioritizeSpeed: false
              })}
            >
              Qualité
            </Button>
            <Button
              variant={preferences?.prioritizeCost ? 'default' : 'outline'}
              onClick={() => setPreferences({
                ...preferences,
                prioritizeCost: !preferences?.prioritizeCost,
                prioritizeQuality: false,
                prioritizeSpeed: false
              })}
            >
              Économie
            </Button>
            <Button
              variant={preferences?.prioritizeSpeed ? 'default' : 'outline'}
              onClick={() => setPreferences({
                ...preferences,
                prioritizeSpeed: !preferences?.prioritizeSpeed,
```

```
                prioritizeQuality: false,
                prioritizeCost: false
              })}
            >
              Vitesse
            </Button>
          </div>
        </div>

        <ModelSelector
          category="image"
          value={preferences?.imageModel || 'fal-ai/flux/dev'}
          onChange={(value) => setPreferences({
            ...preferences,
            imageModel: value
          })}
        />

        <ModelSelector
          category="video"
          value={preferences?.imageToVideoModel || 'fal-ai/luma-dream-machine'}
          onChange={(value) => setPreferences({
            ...preferences,
            imageToVideoModel: value
          })}
        />

        <div className="flex justify-between items-center pt-fluid-sm">
          <Button variant="outline" onClick={() => window.location.reload()}>
            Annuler
          </Button>
          <Button onClick={handleSave}>
            Enregistrer
          </Button>
        </div>
      </CardContent>
    </Card>
  )
}
```

2.3 Comparateur de Modèles

```tsx
// components/model-comparator.tsx
export function ModelComparator() {
  const [selectedModels, setSelectedModels] = useState<string[]>([])
  const [comparison, setComparison] = useState(null)

  const handleCompare = async () => {
    const res = await fetch('/api/models/compare', {
      method: 'POST',
      headers: { 'Content-Type': 'application/json' },
      body: JSON.stringify({ modelIds: selectedModels })
    })
    const data = await res.json()
    setComparison(data.comparison)
  }

  return (
    <Dialog>
      <DialogTrigger asChild>
        <Button variant="outline">Comparer les modèles</Button>
      </DialogTrigger>
      <DialogContent className="max-w-4xl">
        <DialogHeader>
          <DialogTitle>Comparateur de Modèles</DialogTitle>
          <DialogDescription>
            Sélectionnez jusqu'à 3 modèles pour comparer leurs performances
          </DialogDescription>
        </DialogHeader>

        {comparison && (
          <Table>
            <TableHeader>
              <TableRow>
                <TableHead>Modèle</TableHead>
                <TableHead>Qualité</TableHead>
                <TableHead>Coût</TableHead>
                <TableHead>Vitesse</TableHead>
                <TableHead>Coût estimé / 100</TableHead>
              </TableRow>
            </TableHeader>
            <TableBody>
              {comparison.map((model) => (
                <TableRow key={model.id}>
                  <TableCell>{model.name}</TableCell>
                  <TableCell>{'⭐'.repeat(model.qualityRating)}</TableCell>
                  <TableCell>${model.pricePerUnit}/{model.priceUnit}</TableCell>
                  <TableCell>{model.avgSpeed}s</TableCell>
                  <TableCell>${model.estimatedCostPer100.toFixed(2)}</TableCell>
                </TableRow>
              ))}
            </TableBody>
          </Table>
        )}
      </DialogContent>
    </Dialog>
  )
}
```

Phase 3 : Intégration avec le Générateur

```
// Modification de lib/media-generator.ts
export class MediaGenerator {
  private modelManager: ModelManager

  constructor() {
    this.modelManager = new ModelManager()
  }

  async generateContent(prompt: string, userId: string) {
    // Récupérer les préférences utilisateur
    const preferences = await prisma.modelPreferences.findUnique({
      where: { userId }
    })

    // Utiliser les modèles préférés ou les valeurs par défaut
    const imageModel = preferences?.imageModel || 'fal-ai/flux/dev'
    const videoModel = preferences?.imageToVideoModel || 'fal-ai/luma-dream-machine'

    // Estimer le coût avant de générer
    const costEstimate = await this.modelManager.estimateJobCost(
      imageModel,
      videoModel,
      3 // nombre de variations
    )

    // Vérifier le budget utilisateur
    const user = await prisma.user.findUnique({
      where: { id: userId },
      select: { manualBudget: true }
    })

    if (user?.manualBudget && costEstimate.totalCost > user.manualBudget) {
      throw new Error(`Budget insuffisant. Coût estimé: ${costEstim-
ate.totalCost.toFixed(2)}`)
    }

    // Générer avec les modèles sélectionnés
    return await this.generate(prompt, imageModel, videoModel)
  }
}
```

📈 PRÉVISIONS D’ÉCONOMIES

Scénario 1 : Configuration Actuelle (Baseline)

- **Coût journalier** : $7.57
- **Coût mensuel** : $227
- **Coût par génération complète** : $0.53

Scénario 2 : Optimisation Modérée (Wan 2.5 pour vidéo)

- **Coût journalier** : $4.32 (-43%)
- **Coût mensuel** : $130 (-43%)
- **Coût par génération complète** : $0.28 (-47%)

- **Économie mensuelle** : $97

Scénario 3 : Optimisation Maximale (LTX-2 Fast + FLUX schnell)

- **Coût journalier** : $2.91 (-62%)
- **Coût mensuel** : $87 (-62%)
- **Coût par génération complète** : $0.19 (-64%)
- **Économie mensuelle** : $140

Scénario 4 : Configuration Premium (Veo 3.1 + FLUX Pro)

- **Coût journalier** : $9.15 (+21%)
- **Coût mensuel** : $275 (+21%)
- **Coût par génération complète** : $0.61 (+15%)
- **Valeur ajoutée** : Audio natif, qualité cinématique

🎯 RECOMMANDATIONS FINALES

Priorité Immédiate (Impact majeur)

1. **Remplacer Luma par Wan 2.5** pour économiser 50% sur les vidéos
2. **Implémenter le cache de prompts** pour éviter les générations dupliquées
3. **Ajouter l'estimation de coût** avant chaque génération

Court terme (1-2 semaines)

1. **Déployer le sélecteur de modèles** dans les paramètres
2. **Créer le catalogue de modèles** en base de données
3. **Afficher les coûts réels** dans l'historique

Moyen terme (1 mois)

1. **Système de comparaison de modèles** pour aider au choix
2. **Alertes budget** en temps réel
3. **Statistiques de consommation** détaillées

Long terme (2-3 mois)

1. **IA de recommandation** de modèles selon le type de contenu
2. **Tests A/B automatiques** sur les modèles
3. **Optimisation automatique** selon le budget restant

📚 RESSOURCES ET DOCUMENTATION

Documentation fal.ai

- **Explorer les modèles** : https://fal.ai/explore/models
- **Pricing** : https://fal.ai/pricing
- **API Documentation** : https://docs.fal.ai/
- **Optimisation scaling** : https://docs.fal.ai/serverless/deployment-operations/scale-your-application

Catalogue de modèles complet

- **GitHub Gist (tous les modèles)** : https://gist.github.com/azer/6e8f-fa228cb5d6f5807cd4d895b191a4
- **Comparaison de prix** : https://pricepertoken.com/image-models

Alternatives et benchmarks

- **Comparaison fal.ai vs Replicate** : Performance et coûts
- **Alternatives moins chères** : Modal.com, Runpod.io, Pollo AI

Document généré le : 26 octobre 2025
Auteur : ReelGen AI - Analyse d’optimisation
Version : 1.0